MindIE
1.0.RC3

MindIE LLM 开发指南

文档版本 01

发布日期 2026-06-05

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

目 录

1 简介

概述

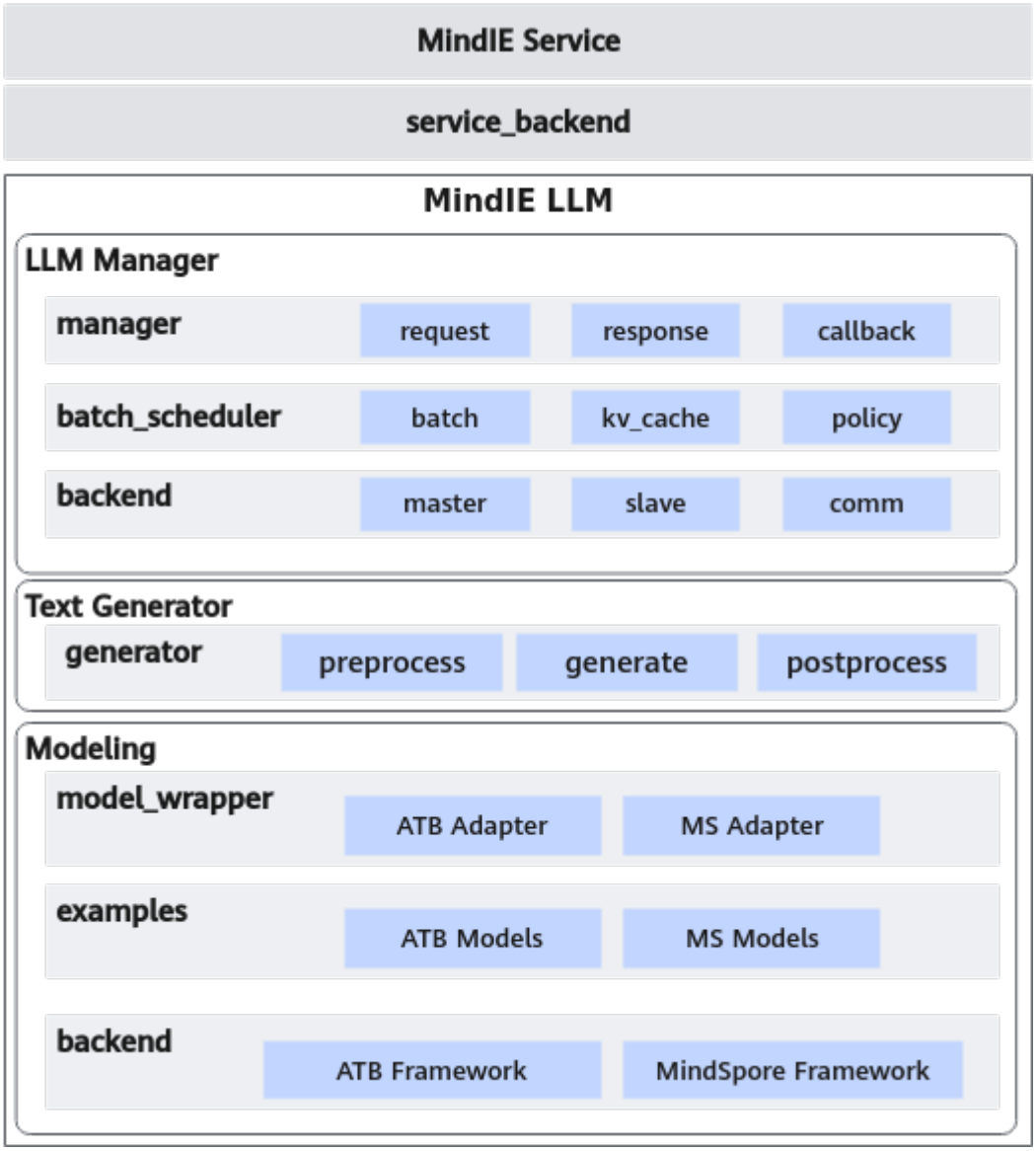
MindIE LLM（Mind Inference Engine Large Language Model，大语言模型）是MindIE下的大语言模型推理组件，基于昇腾硬件提供业界通用大模型推理能力，同时提供多并发请求的调度功能，支持Continuous Batching、Page Attention、FlashDecoding等加速特性，使能用户高性能推理需求。

MindIE LLM主要提供大模型推理Python API和大模型调度C++ API。

本手册有助于用户快速了解MindIE LLM，完成大模型推理的部署测试。

MindIE LLM 架构

图 1-1 MindIE LLM 架构图



MindIE LLM总体架构分为三层：LLM Manager、Text Generator和Modeling。

- LLM Manager：负责状态管理及任务调度，基于调度策略实现用户请求组batch，统一内存池管理kv缓存，返回推理结果，提供状态监控接口。
- Text Generator：负责模型配置、初始化、加载、自回归推理流程、后处理等，向LLM Manager提供统一的自回归推理接口，支持并行解码插件化运行。
- Modeling：提供深度定制优化的模块和内置模型，支持ATB Models（Ascend Transformer Boost Models）和MindSpore Models两种框架。
  - 内置模块包括Attention、Embedding、ColumnLinear、RowLinear、MLP（multilayer perceptron），支持Weight在线Tensor切分加载。
  - 内置模型使用内置模块进行组网拼接，支持Tensor切分，支持多种量化方式，用户亦可参照样例通过内置模块组网自定义模型。

- 组网后的模型经过编译优化后，会生成能在昇腾NPU设备上加速推理的可执行图。

功能特性

MindIE LLM功能特性主要分为模型相关（包括量化）能力和调度相关能力两个维度。

表 1-1 特性清单

类别	特性列表
模型相关（包括量化）能力	支持量化特性，请参见**6.1 量化特性介绍**。
	支持长序列特性，请参见**6.2 长序列特性介绍**。
	支持多机特性，请参见**6.3 多机特性介绍**。
	支持MoE特性，请参见**6.4 MoE特性介绍**。
调度相关能力	支持Multi-LoRA特性，请参见**6.5 Multi-LoRA特性介绍**。
	支持PD分离特性，请参见**6.6 PD分离特性介绍**。
	支持SplitFuse特性，请参见**6.7 SplitFuse特性介绍**。
	支持并行解码特性，请参见**6.8 并行解码特性介绍**。
	支持Prefix Cache特性，请参见**6.9 Prefix Cache特性介绍**。

- **模型相关（包括量化）能力简介**
  a. **基础能力**包括浮点、量化、并行。

表 1-2 浮点特性

浮点特性	浮点能力
float16	√
bfloat16	√

MindIE LLM主打高性能推理，当前仅支持float16、bfloat16浮点格式。可通过配置模型config.json中'torch_dtype'字段进行类型修改。

表 1-3 量化特性

量化特性	per channel	per token	per group
w8a8	√	√	×
w8a16	√	×	√

量化特性	per channel	per token	per group
KV Cache int8	√	×	×
w8a8稀疏量化	√	×	×

MindIE LLM提供多种量化选择进行推理加速，用户可根据自己的需要进行选择，具体量化权重获取、量化推理运行请参考6.1 量化特性介绍章节进行配置。

表 1-4 并行特性

并行特性	并行能力
TP（Tensor Parallelism）	√
DP（Data Parallelism）	×
PP（Pipeline Parallelism）	×
EP（Expert Parallelism）	×

MindIE LLM提供TP并行策略。

b. **模型能力**

MindIE LLM提供如下所示模型预置能力，用户可根据需要进行使用，也可以对模型进行自定义开发迁移。

- LLaMa
- CodeLLaMa
- Baichuan
- Mixtral
- Qwen
- Bloom
- DeepSeek
- Glm
- CodeGeex
- Starcoder
- Gemma

- **调度相关能力简介**

表 1-5 服务化特性

服务化特性	服务化能力
MindIE Service	√
TGI（Text Generation Inference）	√
vLLM	√
Triton	√

MindIE LLM支持自研MindIE Service服务化框架，MindIE Service运行指导请参考《MindIE Service开发指南》的“MindIE Service组件介绍 > **MindIE Server**”章节。

MindIE LLM支持对接第三方的服务化框架，如TGI、vLLM和Triton，提供model backend接口，模型加载、推理以及后处理能力。

2 安装与配置

2.1 安装 MindIE LLM

请参考《**MindIE安装指南**》安装MindIE，并参考《MindIE安装指南》中“配置MindIE > **配置MindIE LLM**”章节配置MindIE LLM。

2.2 配置参数说明

MindIE LLM适用的配置文件config.json格式如下：

```
{
    "Version": "1.0.0",
    "LogConfig" :
    {
        "logLevel" : "Info",
        "logFileSize" : 20,
        "logFileNum" : 20,
        "logPath" : "logs/mindservice.log"
    },
    "BackendConfig": {
        "backendName" : "mindieservice_llm_engine",
        "modelInstanceNumber" : 1,
        "npuDeviceIds" : [[0,1,2,3]],
        "tokenizerProcessNumber" : 8,
        "multiNodesInferEnabled": false,
        "multiNodesInferPort": 1120,
        "interNodeTLSEnabled": true,
        "interNodeTlsCaFile": "security/grpc/ca/ca.pem",
        "interNodeTlsCert": "security/grpc/certs/server.pem",
        "interNodeTlsPk": "security/grpc/keys/server.key.pem",
        "interNodeTlsPkPwd": "security/grpc/pass/mindie_server_key_pwd.txt",
        "interNodeKmcKsfMaster": "tools/pmt/master/ksfa",
        "interNodeTlsCrl" : "security/grpc/certs/server_crl.pem",
        "interNodeKmcKsfStandby": "tools/pmt/standby/ksfb",
        "ModelDeployConfig":
        {
            "maxSeqLen" : 2560,
            "maxInputTokenLen" : 2048,
            "truncation" : false,
            "ModelConfig" : [
```



```
            {
                "modelInstanceType": "Standard",
                "modelName" : "llama_65b",
                "modelWeightPath" : "/data/atb_testdata/weights/llama1-65b-safetensors",
                "worldSize" : 4,
                "cpuMemSize" : 5,
                "npuMemSize" : -1,
                "backendType": "atb"
            }
        ]
    },

    "ScheduleConfig":
    {
        "templateType": "Standard",
        "templateName" : "Standard_LLM",
        "cacheBlockSize" : 128,

        "maxPrefillBatchSize" : 50,
        "maxPrefillTokens" : 8192,
        "prefillTimeMsPerReq" : 150,
        "prefillPolicyType" : 0,

        "decodeTimeMsPerReq" : 50,
        "decodePolicyType" : 0,

        "maxBatchSize" : 200,
        "maxIterTimes" : 512,
        "maxPreemptCount" : 0,
        "supportSelectBatch" : false,
        "maxQueueDelayMicroseconds" : 5000
    }
  }
}
```

配置文件参数说明

配置项	取值类型	取值范围	配置说明
Version	std::string	"1.0.0"	标注配置文件版本，当前版本指定为1.0.0，不支持修改。
LogConfig	map	-	日志相关配置。详情请参见**LogConfig参数**。
BackendConfig	map	-	模型后端相关配置，包含调度、模型相关配置。详情请参见**BackendConfig参数**。

LogConfig 参数

配置项	取值类型	取值范围	配置说明
logLevel	std::string	• "Verbose" • "Info" • "Warning" • "Error" • "None"	• "Verbose"：打印Verbose、Info、Warning和Error级别的日志。 • "Info"：打印Info、Warning和Error级别的日志。 • "Warning"：打印Warning和Error级别的日志。 • "Error"：打印Error级别的日志。 • "None"：不打印日志。 必填，默认值："Info"。
logPath	std::string	日志文件路径，长度<=4096。 日志路径设置限制与操作系统有关，且会受到spdlog三方库约束。因此建议配置的最大长度为1024，且符合目录规范。	支持绝对和相对路径。如果配置为相对路径，则代码中会取安装目录，最后拼接而成。 例如，假设MindIE Service的安装路径为“/opt/Ascend-mindie-service_{version}_linux-x86_64/”，当logPath="logs/mindservice.log"，则其实际日志生成路径为：“/opt/Ascend-mindie-service_{version}_linux-x86_64/logs/mindservice.log”。 若配置路径不满足要求，则使用默认路径。 必填，默认值："logs/mindservice.log"。
logFileSize	uint32_t	日志单个文件大小 范围为：0-500。	日志文件大小，单位MB。
logFileNum	uint32_t	日志文件数量 范围为：0-64。	日志文件数量。

BackendConfig 参数

配置项	取值类型	取值范围	配置说明
backendName	std::string	长度1~50，只支持小写字母和下划线。且不以下划线作为开头和结尾。	后端名称。 必填，默认值："mindieservice_llm_engine"。

配置项	取值类型	取值范围	配置说明
modelInstanceNumber	uint32_t	[1, 10]	模型实例个数。 必填，默认值：1。
npuDeviceIds	std::vector<std::set<size_t>>	根据模型和环境的实际情况来决定。	表示启用哪几张卡。对于每个模型实例分配的npuIds，使用芯片逻辑ID表示。 ● 在未配置ASCEND_RT_VISIBLE_DEVICES环境变量时，每张卡对应的逻辑ID可使用"npu-smi info -m"指令进行查询。 ● 若配置ASCEND_RT_VISIBLE_DEVICES环境变量时，可见芯片的逻辑ID按照ASCEND_RT_VISIBLE_DEVICES中配置的顺序从0开始计数。 例如： ASCEND_RT_VISIBLE_DEVICES=7,1,6 则以上可见芯片的逻辑ID按顺序依次为0,1,2。 多机推理场景下该值无效，每个节点上使用的npuDeviceIds根据ranktable计算获得。 必填，默认值：[[0,1,2,3]]。
tokenizerProcessNumber	uint32_t	[1, 32]	tokenizer进程数。 必填，默认值：8。
multiNodesInferEnabled	bool	● true ● false	● false：单机推理。 ● true：多机推理。 选填，默认值：false。
multiNodesInferPort	int32_t	[1024, 65535]	跨机通信的端口号，多机推理场景使用。 选填，默认值：1120。
interNodeTLSEnabled	bool	● true ● false	多机推理时，跨机通信是否开启证书安全认证。 ● true：开启证书安全认证。 ● false：关闭证书安全认证。 选填，默认值：true。取值为false时，忽略后续参数。

配置项	取值类型	取值范围	配置说明
interNodeTlsCaFile	std::string	建议文件路径长度<=4096。实际路径为工程路径+interNodeTlsCaFile，上限限制与操作系统有关，最小值为1。	根证书名称路径，只支持软件包安装路径下的相对路径。 “interNodeTLSEnabled”=true生效，生效后必填，默认值："security/grpc/ca/ca.pem"。
interNodeTlsCert	std::string	建议文件路径长度<=4096。实际路径为工程路径+interNodeTlsCert，上限限制与操作系统有关，最小值为1。	服务证书文件路径，只支持软件包安装路径下的相对路径。 “interNodeTLSEnabled”=true生效，生效后必填，默认值："security/grpc/certs/server.pem"。
interNodeTlsPk	std::string	建议文件路径长度<=4096。实际路径为工程路径+interNodeTlsPk，上限限制与操作系统有关，最小值为1。	服务证书私钥文件路径，只支持软件包安装路径下的相对路径。 “interNodeTLSEnabled”=true生效，生效后必填，默认值："security/grpc/keys/server.key.pem"。
interNodeTlsPkPwd	std::string	建议文件路径长度<=4096。支持为空；若非空，则实际路径为工程路径+interNodeTlsPkPwd，上限限制与操作系统有关，最小值为1。	服务证书私钥加密密钥文件路径，只支持软件包安装路径下的相对路径。 “interNodeTLSEnabled”=true生效，生效后必填，默认值："security/grpc/pass/mindie_server_key_pwd.txt"。
interNodeTlsCrl	std::string	建议文件路径长度<=4096。实际路径为工程路径+interCommTlsCrl，上限与操作系统有关，最小值为1。	集群内部实例间的通信如果启用TLS，则使用这里指定的文件作为证书吊销列表。选填，默认值："security/grpc/certs/server_crl.pem"。
interNodeKmcKsfMaster	std::string	建议文件路径长度<=4096。实际路径为工程路径+interNodeKmcKsfMaster，上限限制与操作系统有关，最小值为1。	KMC密钥库文件路径，只支持软件包安装路径下的相对路径。 “interNodeTLSEnabled”=true生效，生效后必填，默认值："tools/pmt/master/ksfa"。

配置项	取值类型	取值范围	配置说明
interNodeKmcKsfStandby	std::string	建议文件路径长度<=4096。实际路径为工程路径+interNodeKmcKsfStandby，上限限制与操作系统有关，最小值为1。	KMC密钥库备份文件路径，只支持软件包安装路径下的相对路径。 “interNodeTLSEnabled”=true生效，生效后必填，默认值："tools/pmt/standby/ksfb"。
ModelDeployConfig	map	-	模型部署相关配置。详情请参见**ModelDeployConfig参数**。
ScheduleConfig	map	-	调度相关配置。详情请参见**ScheduleConfig参数**。

ModelDeployConfig 参数

配置项	取值类型	取值范围	配置说明
maxSeqLen	uint32_t	上限根据显存和用户需求来决定，最小值需大于0。	最大序列长度。用户根据自己的推理场景选择maxSeqLen。 如果maxSeqLen大于模型支持的最大序列长度，可能会影响推理精度。 必填，默认值：2560。
maxInputTokenLen	uint32_t	(0, 4294967295]	输入token id最大长度。 必填，默认值：2048。 maxInputTokenLen = min(maxInputTokenLen, maxSeqLen -1) • 当truncation=true时，请求的输入长度inputLen会自动截断，请求的实际输入长度inputLen = min(inputLen, maxInputLen)。 • 当truncation=false时，若请求的输入长度inputLen > maxInputTokenLen，会返回Error。
truncation	bool	• true • false	是否进行参数合理化校验拦截。 • false：校验 • true：不校验 选填，默认值：false。

配置项	取值类型	取值范围	配置说明
ModelConfig	map	-	模型相关配置，包括后处理参数。详情请参见**ModelConfig参数**。

ModelConfig 参数

配置项	取值类型	取值范围	配置说明
modelInstanceType	std::string	• "Standard" • "StandardMock"	模型类型。 • "Standard"：普通推理 • "StandardMock"：假模型 选填，默认值："Standard"。
modelName	std::string	由大写字母、小写字母、数字、中划线、点和下划线组成，且不以中划线、点和下划线作为开头和结尾，字符串长度小于或等于256。	模型名称。 必填，默认值："llama_65b"。
modelWeightPath	std::string	文件绝对路径长度的上限与操作系统有关，最小值为1。	模型权重路径。程序会读取该路径下的config.json中torch_dtype和vocab_size字段的值，需保证路径和相关字段存在。 必填，默认值："/data/atb_testdata/weights/llama1-65b-safetensors"。 该路径会进行安全校验，需要和执行用户的属组和权限保持一致。 模型权重路径及文件的权限需保证其他用户无写权限。
worldSize	uint32_t	根据模型实际情况来决定。每一套模型参数中worldSize必须与使用的NPU数量相等。	启用几张卡推理。目前LLaMa-65B至少启用四张NPU卡。 多机推理场景下该值无效，worldSize根据ranktable计算获得。 必填，默认值：4。
cpuMemSize	uint32_t	上限根据显存和用户需求来决定。只有当maxPreemptCount为0时，才可以取值为0。	CPU中可以用来申请KV Cache的size上限。 必填，默认值：5，建议值：5，单位：GB。

配置项	取值类型	取值范围	配置说明
npuMemSize	int32_t	• -1 • 整型数字，取值范围：(0, 4294967295]	NPU中可以用来申请KV Cache的size上限。 必填，默认值：-1，建议值：-1，单位：GB。 1. 自动分配KV Cache：当配置值为-1时，KV Cache会根据可用显存自动进行分配。 KV Cache快速计算公式：npuMemSize=单卡总内存*内存分配比例-单卡权重占用内存-运行时相关变量占用内存-系统占用内存。 • 单卡总显存：可通过npu-smi info进行查看总显存。 • 内存分配比例：默认值0.8；可通过环境变量NPU_MEMORY_FRACTION控制；**当出现权重加载OOM情况时，可适当调高分配比例或使用更多显卡进行推理。** • 单卡权重占用内存：约等于权重大小*类型大小（浮点为2/int8类型为1）/ 卡数；以实际加载权重为准。 • 运行时相关变量占用内存：模型输入变量、输出变量、中间变量等内存。 • 系统占用内存：可通过npu-smi info进行查看静息状态下使用显存。 2. 手动分配KV Cache：当配置值大于0时，根据设置值会固定分配KV Cache大小。
backendType	std::string	• "atb" • "ms"	对接的后端类型。 必填，默认值："atb"。

ScheduleConfig 参数

配置项	取值类型	取值范围	配置说明
templateType	std::string	"Standard"	推理类型。 必填，默认值："Standard"。

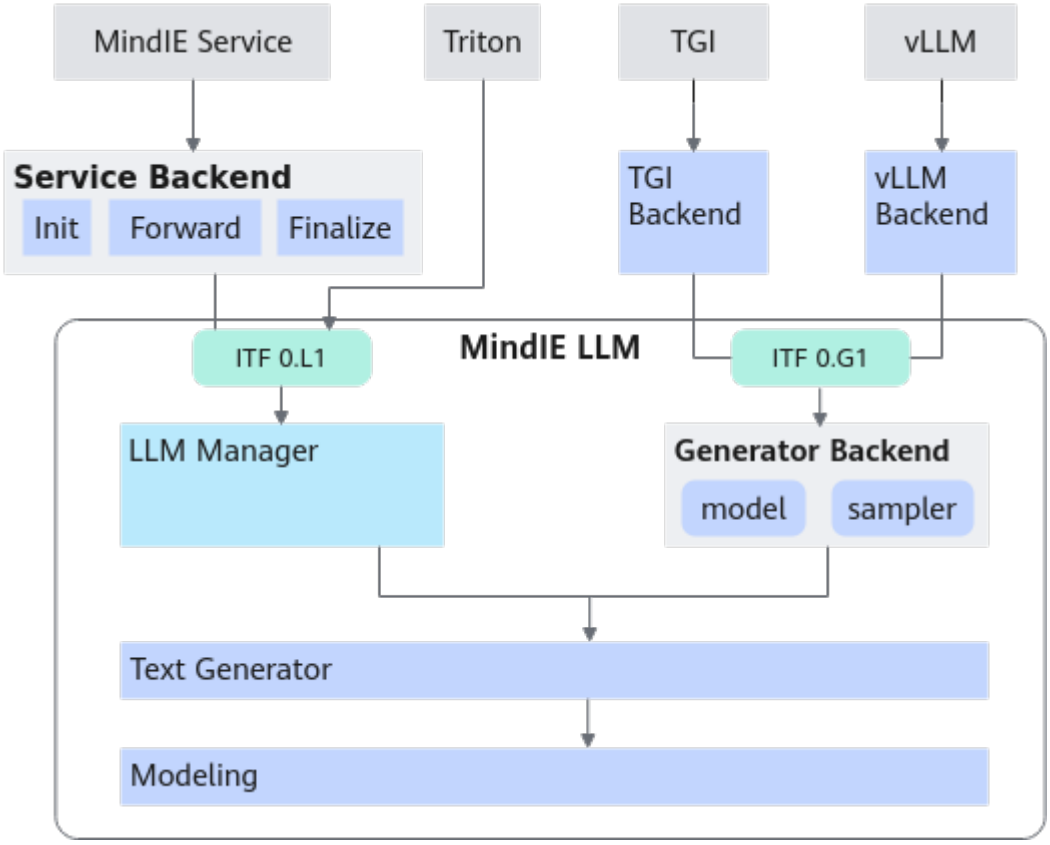
配置项	取值类型	取值范围	配置说明
templateName	std::string	由字母和下划线组成，且不以下划线作为开头和结尾，字符串长度小于或等于256。	工作流名称。 必填，默认值："Standard_LLM"。
cacheBlockSize	uint32_t	[1, 128]	KV Cache Block的size大小。 必填，默认值：128，建议值：128，其他值建议取为2的n次幂。
maxPrefillBatchSize	uint32_t	[1, maxBatchSize]	最大prefill batch size。maxPrefillBatchSize和maxPrefillTokens谁先达到各自的取值就完成本次组batch。 该参数主要是在明确需要限制prefill阶段batch size的场景下使用，否则可以设置为0（此时引擎将默认取maxBatchSize值）或与maxBatchSize值相同。 必填，默认值：50。
maxPrefillTokens	uint32_t	[5120, 512000]，且必须大于或等于maxInputTokenLen的取值。	每次prefill时，当前batch中所有input token总数，不能超过maxPrefillTokens。maxPrefillTokens和maxPrefillBatchSize谁先达到各自的取值就完成本次组batch。 必填，默认值：8192。
prefillTimeMsPerReq	uint32_t	[0, 1000]	与decodeTimeMsPerReq比较，计算当前应该选择prefill还是decode。单位：ms，当“supportSelectBatch”=true时有效。 必填，默认值：150。
prefillPolicyType	uint32_t	• 0 • 1 • 2 • 3	prefill阶段的调度策略。 • 0：FCFS，先来先服务。 • 1：STATE，prefill阶段等同于FCFS策略。 • 2：PRIORITY，优先级队列。 • 3：MLFQ，多级反馈队列。 其中，3是0/1的组合。 必填，默认值：0。

配置项	取值类型	取值范围	配置说明
decodeTimeMsPerReq	uint32_t	[0, 1000]	与prefillTimeMsPerReq比较，计算当前应该选择prefill还是decode。单位：ms，当“supportSelectBatch”=true时有效。 必填，默认值：50。
decodePolicyType	uint32_t	• 0 • 1 • 2 • 3	decode阶段的调度策略。 • 0：FCFS，先来先服务。 • 1：STATE，decode阶段优先执行未被抢占和换出的请求。 • 2：PRIORITY，优先级队列。 • 3：MLFQ，多级反馈队列。 其中，3是0/1的组合。 必填，默认值：0。
maxBatchSize	uint32_t	[1, 5000]，且必须大于或等于maxPreemptCount参数的取值。 **说明** Atlas 300I Duo推理卡的取值范围为[1, 2000]。	最大decode batch size。 1. 首先计算block_num：Total Block Num = Floor(NPU显存/(模型网络数*cacheBlockSize*模型注意力头数*注意力头大小*Cache类型字节数*Cache数))，其中，Cache数=2；在tensor并行的情况下，block_num*world_size为实际的分配block数。 如果是多卡，公式中的模型注意力头数*注意力大小的值需要均摊在每张卡上，即“模型注意力头数*注意力大小/卡数”。 公式中的Floor表示计算结果向下取整。 2. 为每个请求申请的block数量Block Num=Ceil(输入Token数/Block Size)+Ceil(最大输出Token数/Block Size)。输入Token数：输入（字符串）做完tokenizer后的tokenID个数；最大输出Token数：模型推理最大迭代次数和最大输出长度之间取较小值。 公式中的Ceil表示计算结果向上取整。 3. maxBatchSize=Total Block Num/Block Num。 必填，默认值：200。

配置项	取值类型	取值范围	配置说明
maxIterTimes	uint32_t	[1, maxSeqLen-1]	模型全局最大输出长度。与请求级最大输出token个数max_tokens（或max_new_tokens）取较小值作为最大可生成长度。 必填，默认值：512。 请求的最大输出长度maxOutputLen=min(maxIterTimes, max_tokens， max_new_tokens) 请求的实际输出长度outputLen = min(maxSeqLen - inputLen, maxOutputLen)
maxPreemptCount	uint32_t	[0, maxBatchSize]，当取值大于0时，cpuMemSize取值不可为0。	每一批次最大可抢占请求的上限，即限制一轮调度最多抢占请求的数量，最大上限为maxBatchSize，取值大于0则表示开启可抢占功能。 必填，默认值：0。
supportSelectBatch	bool	• true • false	batch选择策略。 • false：表示每一轮调度时，优先调度和执行prefill阶段的请求。 • true：表示每一轮调度时，根据当前prefill与decode请求的数量，自适应调整prefill和decode阶段请求调度和执行的先后顺序。 必填，默认值：false。
maxQueueDelayMicroseconds	uint32_t	[500, 1000000]	队列等待时间，单位：us。 必填，默认值：5000。

3 使用场景

图 3-1 MindIE LLM 接口示意图



MindIE LLM 接口和场景说明

- **服务化接入场景**

表 3-1 服务化场景

服务化框架	MindIE LLM子组件范围	能力说明	接口示意	接口形式
MindIE Service	LLM Manager +TextGenerator +Modeling	LLM Manager +TextGenerator提供完整的CB+PA自回归推理调度能力	ITF 0.L1	C++ API
Triton	LLM Manager +TextGenerator +Modeling	LLM Manager +TextGenerator提供完整的CB+PA自回归推理调度能力	ITF 0.L1	C++ API
TGI	TextGenerator +Modeling	提供TextGenerator前后处理+Modeling推理能力	ITF 0.G1	Python API
vLLM	TextGenerator +Modeling	提供TextGenerator前后处理+Modeling推理能力	ITF 0.G1	Python API

说明

- MindIE Service Backend为MindIE LLM到MindIE Service的适配层，避免MindIE LLM与MindIE Service有编译依赖，接口实现参考《MindIE Service开发指南》。
- TGIBackend为MindIE LLM到第三方服务化TGI的适配层。
- vLLMBackend为MindIE LLM到第三方服务化vLLM的适配层。
- 接口编号ITF 0.L1：ITF 0表示Interface 0，最外层接口；L1表示LLM Manager的第一个接口。
- 接口编号ITF 0.G1：ITF 0表示Interface 0，最外层接口；G1表示Generator的第一个接口。
- 接口权限请找技术支持获取。

- **模型能力测试场景**

MindIE LLM Modeling后端	测试入口	能力说明	使用接口	测试脚本形式
ATB Models	run_pa.py	支持静态场景下模型对话、精度、性能测试	ITF 0.M1	Python API
MindFormers	参考MindFormers社区	参考MindFormers社区	无	参考MindFormers社区

说明

- MindIE LLM Modeling底层提供两种形式的模型后端，满足不同用户的使用需求。
- 接口编号ITF 0.M1：ITF 0表示Interface 0，最外层接口；M1表示Modeling对接TextGenerator的内部接口。

4 模型推理使用流程

4.1 ATB Models 使用

4.1.1 ATB Models 纯模型使用

前提条件

已在环境上安装CANN、PyTorch、Torch-NPU和ATB Models，详情请参见《**MindIE安装指南**》。

说明

本次样例参考以下安装路径进行：

安装ATB Models并初始化ATB Models环境变量。

模型仓set_env.sh脚本中有初始化“${ATB_SPEED_HOME_PATH}”环境变量的操作，所以source模型仓中set_env.sh脚本时会同时初始化“${ATB_SPEED_HOME_PATH}”环境变量。

约束

- 使用ATB Models进行推理，模型初始化失败时，模型初始过程中用户自定义修改导致的失败，需要手动结束进程。
- 使用ATB Models进行推理，权重路径及文件的权限需保证其他用户无写权限。

Readme 文档解读

当前ATB Models包含三类Readme文档指导您执行推理流程，了解模型支持特性以及提供基础的调测和问题定位手段。

图 4-1 ATB Models Readme 文档关系示意图

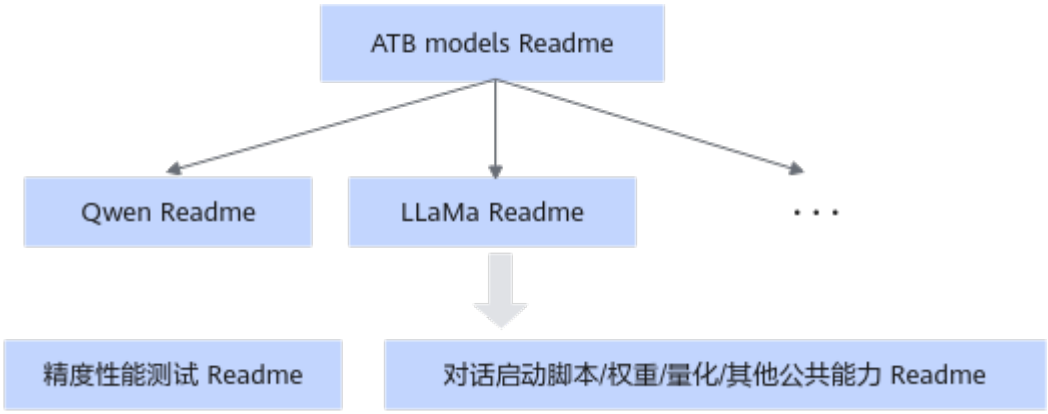


表 4-1 Readme 文档介绍

文档名称	作用	内容
“${ATB_SPEED_HOME_PATH}/README.md”	为ATB Models 所有文档的总入口。	• 运行ATB Models 依赖的硬件和软件版本。 **须知** 每个模型所依赖的软件版本不同，请根据对应的${ATB_MODELS_HOME_PATH}/requirement进行安装，详细信息见“${ATB_SPEED_HOME_PATH}/README.md”文档。 • 基本调测和问题定位手段： – 算子库、加速库和模型仓日志开启方式； – 性能分析方法； – 精度分析方法。 • 预置模型列表： – 此处会链接至模型的README文档。
“${ATB_SPEED_HOME_PATH}/examples/models/{模型名称}/README.md”	为 ATB Models 每个模型各自的文档，例如：“${ATB_SPEED_HOME_PATH}/examples/models/llama/README.md”中为LLaMa模型的文档，其中涵盖了LLaMa系列和LLaMa2系列模型的介绍和运行指导。	• 模型特性支持矩阵，即不同参数量的模型对各类硬件，各种量化方式，各种特性的支持情况。 • 模型开源权重下载地址。 • 模型量化权重生成介绍。 • 对话测试、精度测试和性能测试脚本执行方式。

文档名称	作用	内容
“${ATB_SPEED_HOME_PATH}/examples/README.md”	汇总了对于公共能力和接口的介绍。	• bin格式的权重转safetensor格式脚本的介绍。 • 量化权重生成脚本的介绍。 • Flash Attention和Paged Attention启动脚本参数介绍。 • 可选择性配置的环境变量介绍。 • 特殊场景注意事项。
“${ATB_SPEED_HOME_PATH}/tests/modeltest/README.md”	ModelTest为大模型的性能和精度提供测试功能。	• 性能和精度测试脚本支持矩阵。 • 脚本依赖说明。 • 脚本运行指令。

使用示例

下面以LLaMa2_7B模型为例，展示对话推理、精度测试以及性能测试的执行步骤。

步骤1 配置环境变量。

```
# 配置CANN环境，默认安装在/usr/local目录下
source /usr/local/Ascend/ascend-toolkit/set_env.sh
# 配置加速库环境
source /usr/local/Ascend/nnal/atb/set_env.sh
# 配置模型仓环境变量
source /usr/local/Ascend/llm_model/set_env.sh
```

步骤2 准备模型权重：可从Hugging Face官网直接下载，将下载的权重保存在“/data/Llama-2-7b-hf”。

说明

MindIE LLM中部分模型权重使用Hugging Face提供的接口进行加载，其中默认开启trust_remote_code参数，此时Hugging Face会执行用户权重路径下的代码文件，这些代码文件的功能的安全性需由用户保证。

步骤3 （可选）当前ATB Models 推理仅支持加载safetensor格式的权重文件，若下载的权重文件是safetensor格式文件，则无需进行权重转换，若下载的权重文件是bin格式文件，则需要按照如下方式进行转换。

```
# 进入ATB Models 所在路径
cd ${ATB_SPEED_HOME_PATH}
# 执行脚本生成safetensor格式的权重
python examples/convert/convert_weights.py --model_path /data/Llama-2-7b-hf
```

输出结果会保存在bin格式的权重文件同目录下。

步骤4 测试对话推理。

```
cd ${ATB_SPEED_HOME_PATH}
bash examples/models/llama/run_pa.sh /data/Llama-2-7b-hf
```

如上命令调用的run_pa.sh脚本是对run_pa.py脚本的封装，默认推理内容为"What's deep learning?"，batch size为1，可以通过**步骤5**修改推理内容。

步骤5 自定义推理内容。

- 用户可以通过以下方式直接调用run_pa.py脚本，通过传入参数的方式自定义推理内容及推理方式。

例如：使用/data/Llama-2-7b-hf路径下的权重，使用8卡推理"What's deep learning?"和"Hello World."，推理时batch size为2。

```
# 指定当前机器上可用的逻辑NPU核心，多个核心间使用逗号相连
export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
# 执行推理
torchrun --nproc_per_node 8 --master_port 20030 -m examples.run_pa --model_path /data/
Llama-2-7b-hf --input_texts "What's deep learning?" "Hello World." --max_batch_size 2
```

说明

环境变量说明见**9.3 环境变量说明**。

- 用户可以通过传入Token id的方式进行推理。

  新建一个py脚本（如test.py）用于生成Token id：

  ```
  # 生成Token id的方式
  from transformers import AutoTokenizer
  tokenizer = AutoTokenizer.from_pretrained(
      pretrained_model_name_or_path={tokenizer所在的文件夹路径},
      use_fast=False,
      padding_side='left',
      trust_remote_code=True)
  inputs = tokenizer("What's deep learning?", return_tensors="pt")
  token_id = inputs.data["input_ids"]
  print(token_id)
  ```

  执行如下命令，生成Token id：

  ```
  python test.py
  ```

  执行如下命令进行推理，如下以生成的第一个推理内容对应的Token id为"1,15043,2787"，第二个推理内容对应的Token id为"1,306,626,2691"为例，其中推理内容间以空格分开。

  ```
  # 执行推理
  torchrun --nproc_per_node 8 --master_port 20030 -m examples.run_pa --model_path /data/
  Llama-2-7b-hf --input_ids 1,15043,2787 1,306,626,2691 --max_batch_size 2
  ```

表 4-2 run_pa.py 脚本参数说明

参数名称	是否为必选	类型	默认值	描述
--model_path	是	string	""	模型权重路径。 该路径会进行安全校验，必须使用绝对路径，且和执行推理用户的属组和权限保持一致。
--input_texts	否	string	"What's deep learning?"	推理文本或推理文本路径，多条推理文本间使用空格分割。

参数名称	是否为必选	类型	默认值	描述
--input_ids	否	string	None	推理文本经过模型分词器处理后得到的token id列表，多条推理请求间使用空格分割，单个推理请求内每个token使用逗号隔开。
--input_file	否	string	None	仅支持jsonl格式文件，每一行必须为List[Dict]格式的按时间顺序排序的对话数据，每个Dict字典中需要至少包含"role"和"content"两个字段。
--input_dict	否	parse_list_of_json	None	推理文本以及对应的adapter名称。格式形如：'[{"prompt": "A robe takes 2 bolts of blue fiber and half that much white fiber. How many bolts in total does it take?", "adapter": "adapter1"}, {"prompt": "What is deep learning?", "adapter": "base"}]'
--max_position_embeddings	否	int或者None	None	模型可接受的最大上下文长度。当此值为None时，则从模型权重文件中读取。
--max_input_length	否	int	1024	推理文本最大token数。

参数名称	是否为必选	类型	默认值	描述
--max_output_length	否	int	20	推理结果最大token数。
--max_prefill_tokens	否	int	-1	模型Prefill推理阶段最大可接受的token数。若输入为-1，则max_prefill_tokens = max_batch_size * (max_input_length + max_output_length)
--max_batch_size	否	int	1	模型推理最大batch size。
--block_size	否	int	128	KV Cache分块存储，每块存储的最大token数，默认为128。
--chat_template	否	string或者None	None	对话模型的prompt模板。
--ignore_eos	否	bool	store_true	当推理结果中遇到eos token（句子结束标识符）时，是否结束推理。若传入此参数，则忽略eos token。
--is_chat_model	否	bool	store_true	是否支持对话模式。若传入此参数，则进入对话模式。
--is_embedding_model	否	bool	store_true	是否为embedding类模型。默认为因果推断类模型，若传入此参数，则为embedding类模型。

参数名称	是否为必选	类型	默认值	描述
--load_tokenizer	否	bool	True	是否加载tokenizer。若传入False，则必须传入input_ids参数，且推理输出为token id。
--enable_atb_torch	否	bool	store_true	是否使用Python组图。默认使用C++组图，若传入此参数，则使用Python组图。
--kw_args	否	string	""	扩展参数，支持用户通过扩展参数进行功能扩展。

说明

此章节中的run_pa.py脚本用于纯模型快速测试，脚本中未增加强校验，出现异常情况时，会直接抛出异常信息。例如：

- input_texts、input_ids、input_file、input_dict参数包含推理内容，程序进行数据处理的时间和传入数据量成正比。同时这些输入会被转换成token id搬运至NPU，传入数据量过大可能会导致这些NPU tensor占用显存过大，而出现由out of memory导致的报错信息，例如："req: xx input length: xx is too long, max_prefill_tokens: xx"等报错信息。
- chat_template参数可以使用两种形式输入：模板文本或模板文件的路径。当以模板文本输入时，若文本长度过大，可能会导致运行缓慢。
- 脚本会基于max_batch_size、max_input_length、max_output_length、max_prefill_batch_size和max_prefill_tokens等参数申请推理输入及KV Cache，若用户传入数值过大，会出现由out of memory导致的报错信息，例如："RuntimeError: NPU out of memory. Tried to allocate xxx GiB."。
- 脚本会基于max_position_embeddings参数，申请旋转位置编码和attention mask等NPU tensor，若用户传入数值过大，会出现由out of memory导致的报错信息，例如："RuntimeError: NPU out of memory. Tried to allocate xxx GiB."。
- block_size参数若小于张量并行场景下每张卡实际分到的注意力头个数，会出现由shape不匹配导致的报错（"Setup fail, enable log: export ASDOPS_LOG_LEVEL=ERROR, export ASDOPS_LOG_TO_STDOUT=1 to find the first error. For more details, see the MindIE official document."），需开启日志查看详细信息。

步骤6 测试性能。

开启ATB_LLM_BENCHMARK_ENABLE环境变量后，将统计模型首Token、增量Token及端到端推理时延。

```
# 环境变量开启方式
export ATB_LLM_BENCHMARK_ENABLE=1
# 启动推理方式见步骤4、步骤5
```

耗时结果会显示在Console中，并保存在./benchmark_result/benchmark.csv文件里。

----结束

4.1.2 ATB Models 服务化使用

前提条件

已在环境上安装CANN、PyTorch、Torch-NPU、ATB Models、MindIE LLM和MindIE Motor，详情请参见《**MindIE安装指南**》。

使用实例

步骤1 设置环境变量。

若安装路径为默认路径，可以运行以下命令初始化各组件环境变量。

```
# 配置CANN环境，默认安装在/usr/local目录下
source /usr/local/Ascend/ascend-toolkit/set_env.sh
# 配置加速库环境
source /usr/local/Ascend/nnal/atb/set_env.sh
# 配置模型仓环境变量
source /usr/local/Ascend/llm_model/set_env.sh
# MindIE
source /usr/local/Ascend/mindie/latest/mindie-llm/set_env.sh
source /usr/local/Ascend/mindie/latest/mindie-service/set_env.sh
```

步骤2 启动服务化并发送请求。

MindIE服务化使用方法请参考《MindIE Service开发指南》的“快速开始 > **启动服务**”章节。服务化参数配置请参考《MindIE Service开发指南》的“MindIE Service组件 > MindIE Server > **配置参数说明**”章节。

服务化配置中默认使用ATB Models作为模型后端。

```
vim /usr/local/Ascend/mindie/1.0.RC3/mindie-service/conf/config.json
# ModelDeployConfig.ModelConfig.backendType字段默认值为"atb"
"backendType": "atb"
```

服务化API接口请参考《MindIE Service开发指南》的“**服务化接口**”章节。

用户可使用HTTPS客户端（Linux curl命令，Postman工具等）发送HTTPS请求，此处以Linux curl命令为例进行说明。重开一个窗口，使用以下命令发送请求。

```
curl -H "Accept: application/json" -H "Content-type: application/json" -X POST --cacert {Server服务端证书的验签证书/根证书路径} --cert {客户端证书文件路径} --key {客户端证书私钥路径} -d '{"inputs": "hi","stream":false}' https://{ip}:{port}/generate
```

----结束

4.2 MindSpore Models 使用

4.2.1 MindSpore Models 服务化使用

MindIE LLM不仅支持ATB Models，同时支持MindSpore作为框架后端，MindSpore Models覆盖MindFormers社区下的开源模型，模型列表和使用方式参考**MindSpore Transformers**。

前提条件

- 已在环境上安装CANN详情请参见《**MindIE安装指南**》。

- 已在环境上安装MindFormers，详情请参见**MindFormers官方安装指南**。

权重转换

执行推理前，需将权重格式转为MindFormers所使用的格式（ckpt格式）。
MindFormers提供了统一的权重转换工具，请参考**权重格式转换**。

以Qwen1.5-72B为例，转换后的模型权重目录结构如下：

```
mf_model
└── qwen1_5_72b
├── config.json                # 模型json配置文件
├── vocab.json                 # 模型vocab文件，hf上对应模型下载
├── merges.txt                 # 模型merges文件，hf上对应模型下载
├── predict_qwen1_5_72b.yaml    # 模型yaml配置文件
├── qwen1_5_tokenizer.py        # 模型tokenizer文件，从mindformers仓中research目录下找到对应模型复制
└── qwen1_5_72b_ckpt_dir        # 模型分布式权重文件夹
```

predict_qwen1_5_72b.yaml需要关注以下配置：

```
load_checkpoint: '/mf_model/qwen1_5_72b/qwen1_5_72b_ckpt_dir' # 为存放模型分布式权重文件夹路径
use_parallel: True
auto_trans_ckpt: False    # 是否开启自动权重转换，离线切分设置为False
parallel_config:
  data_parallel: 1
  model_parallel: 4       # 多卡推理配置模型切分，一般与使用卡数一致
  pipeline_parallel: 1
processor:
  tokenizer:
    vocab_file: "/mf_model/qwen1_5_72b/vocab.json"  # vocab文件路径
    merges_file: "/mf_model/qwen1_5_72b/merges.txt"  # merges文件路径
```

模型的config.json文件可以使用save_pretrained接口生成，示例如下：

```
from mindformers import AutoConfig

model_config = AutoConfig.from_pretrained("/mf_model/qwen1_5_72b/predict_qwen1_5_72b.yaml")
model_config.save_pretrained(save_directory="./json/qwen1_5_72b/", save_json=True)
```

使用实例

运行MindSpore Models需配合服务化使用。

步骤1 设置环境变量。

若安装路径为默认路径，可以运行以下命令初始化各组件环境变量。

```
# Ascend
source /usr/local/Ascend/ascend-toolkit/set_env.sh
# MindIE
source /usr/local/Ascend/mindie/latest/mindie-llm/set_env.sh
source /usr/local/Ascend/mindie/latest/mindie-service/set_env.sh
# MindSpore
export LCAL_IF_PORT=8129
# 组网配置
export MS_SCHED_HOST=127.0.0.1  # scheduler节点ip地址
export MS_SCHED_PORT=8090  # scheduler节点服务端口
```

说明

若机器上有其他卡已启动MindIE，需要注意MS_SCHED_PORT参数是否冲突。若日志打印中该参数报错，替换为其他端口号重新尝试即可。

步骤2 启动服务化并发送请求。

MindIE服务化使用方法请参考《 MindIE Service开发指南 》的“快速开始 > **启动服务**”章节，服务化参数配置请参考《 MindIE Service开发指南 》的“MindIE Service组件 > MindIE Server > **配置参数说明**”章节。

若要启用MindSpore Models作为模型后端，服务化配置中需将ModelDeployConfig.ModelConfig.backendType字段设置为"ms"。

```
vim /usr/local/Ascend/mindie/latest/mindie-service/conf/config.json
# 修改ModelDeployConfig.ModelConfig.backendType
"backendType": "ms"
```

服务化API接口请参考《 MindIE Service开发指南 》的“**服务化接口**”章节。

用户可使用HTTPS客户端（ Linux curl命令，Postman工具等 ）发送HTTPS请求，此处以Linux curl命令为例进行说明。重开一个窗口，使用以下命令发送请求。

```
curl -H "Accept: application/json" -H "Content-type: application/json" -X POST --cacert {Server服务端证书的验签证书/根证书路径} --cert {客户端证书文件路径} --key {客户端证书私钥路径} -d '{"inputs": "I love Beijing, because","stream": false}' https://{ip}:{port}/generate
```

须知

MindSpore场景下，请求体中的seed字段限制在[0, 2^32)范围内，若超过则按照默认值seed = 0设置。

----结束

5 API 接口说明

5.1 LLM Manager API 参考（C++）

5.1.1 LLM Manger 提供的 C++接口

5.1.1.1 使用场景

针对自研服务化MindIE Service、三方服务化Triton，提供模型及调度能力，支持Continuous Batching动态调度。

5.1.1.2 类参考

5.1.1.2.1 InferRequest

类说明

InferenceRequest推理引擎功能请求实现类，提供Request初始化，Tensor设置，获取请求信息等功能入口。

说明

以下章节的中的P节点表示：prefill节点，D节点表示：decode节点。

InferRequest 接口

接口功能

构造函数。

接口格式

```
explicit InferRequest(InferRequestId requestId);
```

接口参数

参数	是否必选	说明	取值要求
requestId	是	请求ID。	InferRequestId 类型，请参考 **5.1.1.2.5 InferRequestId**。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
```

返回值

InferRequest对象。

SetTensor 接口

接口功能

请求的设置Tensor接口。

接口格式

```
void SetTensor(const std::string& tensorName, TensorPtr &tensor)
```

接口参数

参数	是否必选	说明	取值要求
tensorName	是	tensor名字。	合法的字符串string类型。
tensor	是	tensor内容。	合法的InferTensor类型，参考 **5.1.1.2.3 InferTensor**。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType = mindie_llm::InferDataType::TYPE_INT64;
std::vector<int64_t> dataShape = {1, 2};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
runtimeRequest->SetTensor(name, inputsTensor);
```


返回值

无。

AddTensor 接口

接口功能

添加Tensor。

接口格式

```
Status AddTensor(const std::string& tensorName, TensorPtr &tensor)
```

接口参数

参数	是否必选	说明	取值要求
tensorName	是	tensor名字。	合法的字符串String类型。
tensor	是	tensor内容。	合法的InferTensor类型，请参考**5.1.1.2.3 InferTensor**。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType = mindie_llm::InferDataType::TYPE_INT64;
std::vector<int64_t> dataShape = {1, 2};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
runtimeRequest->AddTensor(name, inputsTensor);
```

返回值

Status对象。

GetTensorByName 接口

接口功能

根据Name获取Tensor。

接口格式

```
Status GetTensorByName(const std::string& tensorName, TensorPtr &tensor)
```


接口参数

参数	是否必选	说明	取值要求
tensorName	是	tensor名字。	合法的字符串类型。
tensor	是	tensor内容。	合法的InferTensor类型，请参考 **5.1.1.2.3 InferTensor**。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType = mindie_llm::InferDataType::TYPE_INT64;
std::vector<int64_t> dataShape = {1, 2};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
runtimeRequest->AddTensor(name, inputsTensor);
TensorPtr tensor = std::make_shared<mindie_llm::InferTensor>();
Status ret = request->GetTensorByName("INPUT_IDS", tensor);
```

返回值

Status对象。

DelTensorByName 接口

接口功能

根据Name删除Tensor。

接口格式

```
Status DelTensorByName(const std::string &name)
```

接口参数

参数	是否必选	说明	取值要求
name	是	tensor名字。	合法的字符串类型。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType = mindie_llm::InferDataType::TYPE_INT64;
std::vector<int64_t> dataShape = {1, 2};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
runtimeRequest->AddTensor(name, inputsTensor);
```



```
TensorPtr tensor = std::make_shared<mindie_llm::InferTensor>();
Status ret = request->DelTensorByName("INPUT_IDS");
```

返回值

Status对象。

GetRequestId 接口

接口功能

获取请求ID。

接口格式

```
InferRequestId GetRequestId() const
```

接口参数

无。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
mindie_llm::InferRequestId RequestId = runtimeRequest->GetRequestId();
```

返回值

InferRequestId对象。

SetMaxOutputLen 接口

接口功能

设置最大输出长度。

接口格式

```
Status SetMaxOutputLen(uint32_t maxOutputLen)
```

接口参数

参数	是否必选	说明	取值要求
maxOutputLen	是	最大输出长度。	uint32类型。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
uint32_t maxOutputLen = 1;
Status ret = runtimeRequest->SetMaxOutputLen(maxOutputLen);
```


返回值

Status对象。

GetMaxOutputLen 接口

接口功能

获取最大输出长度。

接口格式

```
uint32_t GetMaxOutputLen() const
```

接口参数

无。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
uint32_t maxOutputLen = 1;
Status ret = runtimeRequest->SetMaxOutputLen(maxOutputLen);
uint32_t maxOutputLen1 = runtimeRequest->GetMaxOutputLen();
```

返回值

返回请求最大输出长度 类型为uint32_t。

GetRequestInner 接口

接口功能

获取InferRequestImpl对象。

接口格式

```
std::shared_ptr<InferRequestImpl> GetRequestInner() const
```

接口参数

无。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
std::shared_ptr<InferRequestImpl> request = runtimeRequest->GetRequestInner();
```

返回值

返回指向InferRequestImpl的指针对象。

ImmutableInputs 接口

接口功能

获取TensorMap对象。

接口格式

```
const TensorMap &ImmutableInputs() const
```

接口参数

无。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType = mindie_llm::InferDataType::TYPE_INT64;
std::vector<int64_t> dataShape = {1, 2};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
runtimeRequest->AddTensor(name, inputsTensor);
TensorMap tensorMap = runtimeRequest->ImmutableInputs();
```

返回值

返回TensorMap对象。

SetReqType 接口

接口功能

设置请求类型，prefill或decode。

接口格式

```
void SetReqType(mindie_llm::InferReqType reqType);
```

接口参数

参数	是否必选	说明	取值要求
reqType	是	请求类型。	• REQ_STAND_INFER = 0, • REQ_PREFILL = 1, • REQ_DECODE = 2。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
runtimeRequest->SetReqType(mindie_llm::InferReqType::REQ_STAND_INFER);
```


返回值

无。

GetReqType 接口

接口功能

获取请求类型。

接口格式

```
mindie_llm::InferReqType GetReqType() const;
```

接口参数

无。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
runtimeRequest->SetReqType(mindie_llm::InferReqType::REQ_STAND_INFER);
mindie_llm::InferReqType type = runtimeRequest->GetReqType();
```

返回值

enum class InferReqType {

REQ_STAND_INFER = 0,

REQ_PREFILL = 1,

REQ_DECODE = 2

};

IsPrefillReq 接口

接口功能

请求是否是Prefill类型。

接口格式

```
bool IsPrefillReq() const;
```

接口参数

无。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
runtimeRequest->SetReqType(mindie_llm::InferReqType::REQ_STAND_INFER);
bool isPrefill = runtimeRequest->IsPrefillReq();
```


返回值

bool值。

IsDecodeReq 接口

接口功能

请求是否是Decode类型。

接口格式

```
bool IsDecodeReq() const;
```

接口参数

无。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
runtimeRequest->SetReqType(mindie_llm::InferReqType::REQ_STAND_INFER);
bool isDecode = runtimeRequest->IsDecodeReq();
```

返回值

bool值。

SetDTarget 接口

接口功能

设置请求发往哪个D节点的server IP。

接口格式

```
void SetDTarget(std::string &dTarget);
```

接口参数

参数	是否必选	说明	取值要求
dTarget	是	设置做完全量推理发往D节点的ip地址。	字符串类型，server ip地址。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
string serverIp = "";
runtimeRequest->SetDTarget(serverIp);
```


返回值

无。

GetDTarget 接口

接口功能

获取请求发往哪个D节点的IP。

接口格式

```
std::string GetDTarget() const;
```

接口参数

无。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
string serverIp = "";
runtimeRequest->SetDTarget(serverIp);
std::string ipTarget = runtimeRequest->GetDTarget();
```

返回值

目标server IP。

SetPrefillAddr 接口

接口功能

设置请求在P节点上的server IP。

接口格式

```
void SetPrefillAddr(std::string &prefillAddr);
```

接口参数

参数	是否必选	说明	取值要求
prefillAddr	是	全量推理阶段的ip地址。	字符串类型， server ip地址。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
string prefillAddr= "";
runtimeRequest->SetPrefillAddr(prefillAddr);
```


返回值

无。

GetPrefillAddr 接口

接口功能

获取P节点的Server IP。

接口格式

```
std::string GetPrefillAddr() const;
```

接口参数

无。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
string prefillAddr= "";
runtimeRequest->SetPrefillAddr(prefillAddr);
string addr = runtimeRequest->GetPrefillAddr();
```

返回值

P节点的Server IP。

SetSrcBlockTable 接口

接口功能

设置请求在P节点上的block table。

接口格式

```
void SetSrcBlockTable(const std::vector<int64_t> &srcBlockTable);
```

接口参数

参数	是否必选	说明	取值要求
srcBlockTable	是	全量推理节点上的block table。	元素为int64_t类型的vector，元素值大于等于0。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
std::vector<int64_t> srcBlockTable{0, 1, 2};
runtimeRequest->SetSrcBlockTable(srcBlockTable);
```


返回值

无。

GetSrcBlockTable 接口

接口功能

获取请求在P节点上的block table下标。

接口格式

```
std::vector<int64_t> GetSrcBlockTable() const;
```

接口参数

无。

使用样例

```
mindie_llm::InferRequestId runtimeReqId(0);
std::shared_ptr<mindie_llm::InferRequest> runtimeRequest(runtimeReqId);
std::vector<int64_t> srcBlockTable{0, 1, 2};
runtimeRequest->SetSrcBlockTable(srcBlockTable);
auto blockTable = runtimeRequest->GetSrcBlockTable();
```

返回值

源block table。

~InferRequest

接口功能

默认析构函数。

接口格式

```
~InferRequest()
```

接口参数

无。

返回值

无。

5.1.1.2.2 LlmManager

类说明

Llmmanager推理引擎功能管理类，提供Manager模块的初始化和终止。

LlmManager 接口

接口功能

默认构造函数。

接口格式

```
LlmManager(const std::string &llmConfigPath, mindie_llm::GetRequestsCallback getRequest,
        mindie_llm::SendResponsesCallback sendResponse, mindie_llm::ControlSignalCallback controlCallback,
        mindie_llm::LlmManagerStatsCallback statusCallback,
        mindie_llm::SendStatusResponseCallback statusResponseCallback,
std::map<std::string, std::string> ipInfo = std::map<std::string, std::string>())
```

接口参数

5.1.1.4 函数中Callback定义的五个函数指针类型，以及如下参数

参数	是否必选	说明	取值要求
ipInfo	是	将PD场景模型初始化需要的节点身份信息相关参数传递给模型。	std::map<std::string, std::string>
llmConfigPath	是	将llmmanager需要的配置路径传递给llmmanager的构造函数。	const std::string

使用样例

```
mindie_llm::GetRequestsCallback getRequestCallback;
mindie_llm::SendResponsesCallback sendResponsesCallback;
mindie_llm::ControlSignalCallback stopSignalCallback;
mindie_llm::LlmManagerStatsCallback statsCallback;
mindie_llm::SendStatusResponseCallback sendStatusCallback;
const std::string llmConfigPath;
std::map<std::string, std::string> ipInfo;
std::shared_ptr<mindie_llm::LlmManager> llmManager =
std::make_shared<mindie_llm::LlmManager>(llmConfigPath, getRequestCallback, sendResponsesCallback,
stopSignalCallback, statsCallback, sendStatusCallback,ipInfo);
```

返回值

返回LlmManager对象。

GetModelParams 接口

接口功能

提供获取模型参数的功能。

接口格式

```
std::map<std::string, std::string> GetModelParams() const
```


接口参数

无。

使用样例

```
mindie_llm::GetRequestsCallback getRequestCallback;
mindie_llm::SendResponsesCallback sendResponsesCallback;
mindie_llm::ControlSignalCallback stopSignalCallback;
mindie_llm::LlmManagerStatsCallback statsCallback;
mindie_llm::SendStatusResponseCallback sendStatusCallback;
const std::string llmConfigPath;
std::map<std::string, std::string> ipInfo;
std::shared_ptr<mindie_llm::LlmManager> llmManager =
std::make_shared<mindie_llm::LlmManager>(llmConfigPath, getRequestCallback, sendResponsesCallback,
stopSignalCallback, statsCallback, sendStatusCallback,ipInfo);
std::map<std::string, std::string> modeparams = llmManager->GetModelParams();
```

返回值

返回map格式的模型参数。

Shutdown 接口

接口功能

提供终止LLMManager的功能。

接口格式

```
void Shutdown()
```

接口参数

无。

使用样例

```
mindie_llm::GetRequestsCallback getRequestCallback;
mindie_llm::SendResponsesCallback sendResponsesCallback;
mindie_llm::ControlSignalCallback stopSignalCallback;
mindie_llm::LlmManagerStatsCallback statsCallback;
mindie_llm::SendStatusResponseCallback sendStatusCallback;
const std::string llmConfigPath;
std::map<std::string, std::string> ipInfo;
std::shared_ptr<mindie_llm::LlmManager> llmManager =
std::make_shared<mindie_llm::LlmManager>(llmConfigPath, getRequestCallback, sendResponsesCallback,
stopSignalCallback, statsCallback, sendStatusCallback,ipInfo);
llmManager->Shutdown();
```

返回值

无。

GetMaxPositionEmbeddings 接口

接口功能

获取maxPositionEmbeddings数值。

接口格式

```
uint32_t GetMaxPositionEmbeddings() const
```

接口参数

无。

使用样例

```
mindie_llm::GetRequestsCallback getRequestCallback;
mindie_llm::SendResponsesCallback sendResponsesCallback;
mindie_llm::ControlSignalCallback stopSignalCallback;
mindie_llm::LlmManagerStatsCallback statsCallback;
mindie_llm::SendStatusResponseCallback sendStatusCallback;
const std::string llmConfigPath;
std::map<std::string, std::string> ipInfo;
std::shared_ptr<mindie_llm::LlmManager> llmManager =
std::make_shared<mindie_llm::LlmManager>(llmConfigPath, getRequestCallback, sendResponsesCallback,
stopSignalCallback, statsCallback, sendStatusCallback,ipInfo);
uint32_t maxposition = llmManager->GetMaxPositionEmbeddings();
```

返回值

返回值为uint32_t类型的maxPositionEmbeddings数值。

UpdateEngineInfo 接口

接口功能

下发切换身份请求。

接口格式

```
bool UpdateEngineInfo(std::shared_ptr<mindie_llm::InferRequest> &runtimeRequest, bool isForceRelease);
```

接口参数

参数	是否必选	说明	取值要求
runtimeRequest	是	切换身份请求。	std::shared_ptr<MindIE_LLM::InferRequest>。
isForceRelease	是	是否强制释放节点间连接。	bool。

使用样例

```
mindie_llm::GetRequestsCallback getRequestCallback;
mindie_llm::SendResponsesCallback sendResponsesCallback;
mindie_llm::ControlSignalCallback stopSignalCallback;
mindie_llm::LlmManagerStatsCallback statsCallback;
mindie_llm::SendStatusResponseCallback sendStatusCallback;
const std::string llmConfigPath;
std::map<std::string, std::string> ipInfo;
std::shared_ptr<mindie_llm::LlmManager> llmManager =
std::make_shared<mindie_llm::LlmManager>(llmConfigPath, getRequestCallback, sendResponsesCallback,
```



```
stopSignalCallback, statsCallback, sendStatusCallback,ipInfo);
std::shared_ptr<MindIE_LLM::InferRequest> request;
llmManager->UpdateEngineInfo(request, false);
```

返回值

bool值。

Init 接口

接口功能

LlmManager的初始化接口。

接口格式

```
Status Init(uint32_t modelInstanceId, std::set<size_t> npuDeviceIds)
```

接口参数

参数	是否必选	说明	取值要求
modelInstanceId	是	模型实例Id。	uint32类型。
npuDeviceIds	是	npu设备的ids。	std::set<size_t>。

使用样例

```
// 需要定义6.1.1.4的函数
mindie_llm::GetRequestsCallback getRequestCallback;
mindie_llm::SendResponsesCallback sendResponsesCallback;
mindie_llm::ControlSignalCallback stopSignalCallback;
mindie_llm::LlmManagerStatsCallback statsCallback;
mindie_llm::SendStatusResponseCallback sendStatusCallback;
const std::string llmConfigPath;
std::map<std::string, std::string> ipInfo;
std::shared_ptr<mindie_llm::LlmManager> llmManager =
std::make_shared<mindie_llm::LlmManager>(llmConfigPath, getRequestCallback, sendResponsesCallback,
stopSignalCallback, statsCallback, sendStatusCallback,ipInfo);
uint32 modelInstanceId = 0;
std::set<size_t> npuDeviceIds = {0};
llmManager->Init(modelInstanceId,npuDeviceIds);
```

返回值

Status对象。

~LlmManager 接口

接口功能

默认析构函数。

接口格式

```
~LlmManager()
```


接口参数

无。

返回值

无。

5.1.1.2.3 InferTensor

类说明

Llmmanager推理引擎功能数据类，提供输入和输出的基类。

InferTensor 接口

接口功能

默认构造函数。

接口格式

```
InferTensor() = default;
InferTensor(std::string name, InferDataType dataType, std::vector<int64_t> dataShape);
```

接口参数

参数	是否必选	说明	取值要求
name	是	tensor名字。	合法的字符串类型string类型。
dataType	是	tensor数据类型。	合法的数据类型,请参考**5.1.1.3.1 InferDataType**。
dataShape	是	tensor数据维度。	合法的数据维度，类型为std::vector<int64_t>。

使用样例

```
auto runtimeTensor = std::make_shared<mindie_llm::InferTensor>();
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType = mindie_llm::InferDataType::TYPE_INT64;
std::vector<int64_t> dataShape = {1, 2};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
```

返回值

InferTensor对象。

GetShape 接口

接口功能

获取张量对象shape。

接口格式

```
const std::vector<int64_t> &GetShape() const
```

接口参数

无。

使用样例

```
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType = mindie_llm::InferDataType::TYPE_INT64;
std::vector<int64_t> dataShape = {1, 2};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
std::vector<int64_t> shape = inputsTensor->GetShape();
```

返回值

tensor的数据维度，类型为vector<int64_t>。

GetSize 接口

接口功能

获取张量对象数据大小。

接口格式

```
size_t GetSize() const
```

接口参数

无。

使用样例

```
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType = mindie_llm::InferDataType::TYPE_INT64;
std::vector<int64_t> dataShape = {1, 2};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
size_t size = inputsTensor->GetSize();
```

返回值

张量对象数据大小，类型为size_t。

GetDataType 接口

接口功能

获取张量对象数据类型。

接口格式

```
InferDataType GetDataType() const
```

接口参数

无。

使用样例

```
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType = mindie_llm::InferDataType::TYPE_INT64;
std::vector<int64_t> dataShape = {1, 2};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
mindie_llm::InferDataType dataType2 = inputsTensor->GetDataType();
```

返回值

张量对象数据类型，类型为InferDataType。

GetMemType 接口

接口功能

获取张量对象的MemType。

接口格式

```
MemType GetMemType() const
```

接口参数

无。

使用样例

```
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType = mindie_llm::InferDataType::TYPE_INT64;
std::vector<int64_t> dataShape = {1, 2};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
mindie_llm::MemType type = inputsTensor->GetMemType();
```

返回值

张量对象的MemType，类型为MemType。

GetData 接口

接口功能

获取张量对象的Data。

接口格式

```
void* GetData() const;
```


接口参数

无。

使用样例

```
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType = mindie_llm::InferDataType::TYPE_INT64;
int64_t dataSize = 2;
std::vector<int64_t> dataShape = {1, dataSize};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
size_t bufferSize = dataSize * sizeof(int64_t);
inputsTensor->Allocate(bufferSize);
auto data = inputsTensor->GetData();
```

返回值

张量对象的数据的首地址。

GetName 接口

接口功能

获取张量对象的名字。

接口格式

```
const std::string &GetName() const
```

接口参数

无。

使用样例

```
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType = mindie_llm::InferDataType::TYPE_INT64;
std::vector<int64_t> dataShape = {1, 2};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
const std::string name = inputsTensor->GetName();
```

返回值

张量对象的名字，类型为字符串类型。

Truncate 接口

接口功能

对输入张量对象进行截断。

接口格式

```
bool Truncate(const size_t truncLen)
```


接口参数

参数	是否必选	说明	取值要求
truncLen	是	合法的数据长度大小。	合法的数据长度大小，类型为size_t。

使用样例

```
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType = mindie_llm::InferDataType::TYPE_INT64;
std::vector<int64_t> dataShape = {1, 2};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
bool ret = inputsTensor->Truncate(1);
```

返回值

无。

Allocate 接口

接口功能

为输入张量对象分配内存空间，用于接收输入数据。

接口格式

```
bool Allocate(size_t size)
```

接口参数

参数	是否必选	说明	取值要求
size	是	合法的内存空间大小。	size_t类型。

使用样例

```
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType= mindie_llm::InferDataType::TYPE_INT64;
int64_t dataSize = 2;
std::vector<int64_t> dataShape = {1, dataSize};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
size_t bufferSize = dataSize * sizeof(int64_t);
inputsTensor->Allocate(bufferSize);
```

返回值

张量对象内存空间分配是否成功，返回类型为bool类型。

SetBuffer 接口

接口功能

为输入张量对象设置数据，数据大小和是否需要释放内存的标识。

接口格式

```
void SetBuffer(const void *buffer, size_t tensorbyteSize, bool tensorNeedRelease)
```

接口参数

参数	是否必选	说明	取值要求
buffer	是	张量数据地址指针。	合法的地址指针：需要指向由malloc、calloc和realloc等函数动态分配的内存地址。
tensorbyteSize	是	张量数据内存大小。	合法的张量内存空间大小，类型为size_t 。
tensorNeedRelease	是	是否需要释放内存的标识。	合法的bool类型。

使用样例

```
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType= mindie_llm::InferDataType::TYPE_INT64;
int64_t dataSize = 2;
std::vector<int64_t> dataShape = {1, dataSize};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
uint64_t bufferSize = dataSize * sizeof(int64_t);
void *data = (void *)malloc(bufferSize);
inputsTensor->SetBuffer(data, bufferSize, false);
```

返回值

无。

SetRelease 接口

接口功能

为输入张量对象设置是否需要释放数据内存标识。

接口格式

```
void SetRelease(bool releaseFlag)
```


接口参数

参数	是否必选	说明	取值要求
releaseFlag	是	是否需要释放数据内存的标识。	合法的bool类型。

使用样例

```
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType= mindie_llm::InferDataType::TYPE_INT64;
int64_t dataSize = 2;
std::vector<int64_t> dataShape = {1, dataSize};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
uint64_t bufferSize = dataSize * sizeof(int64_t);
void *data = (void *)malloc(bufferSize);
inputsTensor->SetBuffer(data, bufferSize, false);
inputsTensor->SetRelease(true);
```

返回值

无。

Release 接口

接口功能

释放输入张量对象内存。

接口格式

```
void Release()
```

接口参数

无。

使用样例

```
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType = mindie_llm::InferDataType::TYPE_INT64;
std::vector<int64_t> dataShape = {1, 2};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
inputsTensor->Release();
```

返回值

无。

GetTypeByteSize 接口

接口功能

获取对应InferDataType类型的字节数。

接口格式

```
static size_t GetTypeByteSize(InferDataType dataType)
```

接口参数

参数	是否必选	说明	取值要求
dataType	是	数据类型。	InferDataType类型。

使用样例

```
std::string name = "INPUT_IDS";
mindie_llm::InferDataType dataType = mindie_llm::InferDataType::TYPE_INT64;
std::vector<int64_t> dataShape = {1, 2};
auto inputsTensor = std::make_shared<mindie_llm::InferTensor>(name, dataType, dataShape);
size_t size = inputsTensor->GetTypeByteSize();
```

返回值

InferDataType类型的字节数，类型为size_t。

~InferTensor 接口

接口功能

默认析构函数。

接口格式

```
~InferTensor()
```

接口参数

无。

返回值

无。

5.1.1.2.4 Error

类说明

推理引擎错误码类，提供错误码的创建、获取等功能。

Error 接口

接口功能

- 根据枚举Code创建返回信息对象，错误码参考**5.1.1.3.4 Code枚举类**。
- 根据枚举Code、错误信息msg，创建返回信息对象。

接口格式

```
explicit Error(Code code = Code::OK);
explicit Error(Code code, std::string msg);
```

接口参数

参数	是否必选	说明	取值要求
code	必选	错误码。	• Code::OK • Code::ERROR • Code::INVALID • Code::NOT_FOUND
msg	必选	错误信息。	合法字符串string类型。

使用样例

```
Error(Error::Code::OK);
Error(Error::Code::ERROR, "ERROR: Failed to get real path of home.");
```

返回值

Error对象。

ErrorCode 接口

接口功能

获取返回对象内的错误码。

接口格式

```
Code ErrorCode() const;
```

接口参数

无。

使用样例

```
Error(Error::Code::ERROR, "ERROR: Failed to get real path of home.").ErrorCode();
```

返回值

错误码枚举。

Message 接口

接口功能

获取返回对象内的错误码信息。

接口格式

```
const std::string &Message() const;
```

接口参数

无。

使用样例

```
Error(Error::Code::ERROR, "ERROR: Failed to get real path of home.").Message();
```

返回值

错误码信息，string类型。

IsOk 接口

接口功能

获取返回对象状态，是否是正常状态。

接口格式

```
bool IsOk() const;
```

接口参数

无。

使用样例

```
auto error = Error(Code code = Code::OK);
if (!error.IsOk()) {
    return -1;
}
```

返回值

- true：返回正常，代表操作成功。
- false：返回异常，代表操作失败。

ToString 接口

接口功能

将返回对象转为字符串格式，具体为错误码映射为相应字符后，和错误信息以": "进行拼接。

接口格式

```
std::string ToString() const;
```

接口参数

无。

使用样例

```
auto err = Error(Error::Code::ERROR, "ERROR: Failed to get real path of home.");
err.ToString();
```

返回值

返回对象转化出来的字符串。

Error对象	返回字符格式
Error::Code::OK	OK: + msg
Error::Code::ERROR	Error: +msg
Error::Code::INVALID_ARG	Invalid argument: +msg
Error::Code::NOT_FOUND	Not found: +msg

CodeToString 接口

接口功能

将错误码转化为对应字符。

接口格式

```
static const char *CodeToString(const Code code);
```

接口参数

参数	是否必选	说明	取值要求
code	必选	错误码枚举。	枚举类Code值，请参考**5.1.1.3.4 Code枚举类**。

使用样例

```
std::string str(CodeToString(code_));
```

返回值

返回错误码转化出来的对应字符。

Code枚举	返回字符格式
Code::OK	OK
Code::ERROR	Error
Code::INVALID_ARG	Invalid argument
Code::NOT_FOUND	Not found

5.1.1.2.5 InferRequestId

类说明

推理引擎请求ID实现类，提供请求ID的初始化。

InferRequestId 接口

接口功能

- InferRequestId的默认构造函数。
- 根据序列字符串requestLabel值创建以string类型值作为唯一序列号的请求体对象。
- 根据序列uint64类型requestIndex值创建以uint64类型值作为唯一序列号的请求体对象。

接口格式

```
explicit InferRequestId();
explicit InferRequestId(const std::string requestLabel);
explicit InferRequestId(uint64_t requestIndex);
```

接口参数

参数	是否必选	说明	取值要求
requestLabel	必选	推理请求对象的唯一性标识，序列号，这里是string类型的。	合法的字符类型，string类型请求序列号。

参数	是否必选	说明	取值要求
requestIndex	必选	推理请求对象的唯一序列号，uint64类型的。	合法的uint64唯一序列号。

使用样例

```
mindie_llm::InferRequestId1 runtimeReqId();
mindie_llm::InferRequestId2 runtimeReqId("Req");
mindie_llm::InferRequestId3 runtimeReqId(0);
```

返回值

InferRequestId对象。

InferRequestId 操作符重载接口

接口功能

- 重载推理请求体对象的=号运算符，用于给请求体初始化为以uint64_t类型作为序列号类型，序列号值是rhs的推理请求体对象，用于标识一个请求体。
- 重载推理请求体对象的=号运算符，用于给请求体初始化为以string类型作为序列号类型，序列号值是字符rhs的推理请求体对象，用于标识一个请求体。
- 重载推理请求体对象的=号运算符，用于从已有的请求体对象初始化一个新的请求体对象。

接口格式

```
InferRequestId &operator = (const uint64_t rhs);
InferRequestId &operator = (const std::string &rhs);
InferRequestId &operator = (const InferRequestId &rhs);
```

接口参数

参数	是否必选	说明	取值要求
rhs	必选	请求体序列号，唯一标识请求体。	合法的uint64_t数值。

参数	是否必选	说明	取值要求
rhs	必选	请求体序列号，唯一标识请求体。	合法的字符串。

参数	是否必选	说明	取值要求
rhs	必选	推理请求体对象。	合法的请求体对象。

使用样例

```
mindie_llm::InferRequestId runtimeReqId1(0);
uint64_t reqId = 0;
mindie_llm::InferRequestId runtimeReqId2 = reqId ;
```



```
mindie_llm::InferRequestId runtimeReqId3 = "Req";
mindie_llm::InferRequestId runtimeReqId4 = runtimeReqId1;
```

返回值

InferRequestId对象。

InferRequestId 接口

接口功能

使用已有的InferRequestId初始化InferRequestId对象。

接口格式

```
InferRequestId(const InferRequestId &other)
```

接口参数

参数	是否必选	说明	取值要求
other	必选	已有的InferRequestId对象。	合法的请求体对象。

使用样例

```
mindie_llm::InferRequestId runtimeReqId1(0);
mindie_llm::InferRequestId runtimeReqId2(runtimeReqId1);
```

返回值

InferRequestId对象。

Type 接口

接口功能

获取请求体对象唯一的序列号标识的数据类型。

接口格式

```
DataType Type() const;
```

接口参数

无。

使用样例

```
mindie_llm::InferRequestId runtimeReqId1(0);
mindie_llm::DataType dataType = runtimeReqId1.Type();
```


返回值

请求体对象唯一的序列号标识的数据类型。

StringValue 接口

接口功能

获取请求体字符类型的序列号，多为请求体序列号类似为string类型请求体使用。

接口格式

```
const std::string &StringValue() const;
```

接口参数

无。

使用样例

```
mindie_llm::InferRequestId runtimeReqId1("0");
std::string value = runtimeReqId1.Type();
```

返回值

返回请求体对象字符类型序列号，若请求体对象为uint64_t类型，会返回空序列号。

UnsignedIntValue 接口

接口功能

获取请求体数字类型的序列号，多为请求体序列号类似为uint64_t类型请求体使用。

接口格式

```
uint64_t UnsignedIntValue() const;
```

接口参数

无。

使用样例

```
mindie_llm::InferRequestId runtimeReqId1(0);
uint64_t value = runtimeReqId1.Type();
```

返回值

返回请求体对象数字类型序列号，若请求体对象为string类型，会返回空默认值0。

GetRequsetIdString 接口

接口功能

获取请求ID的STRING类型。

接口格式

```
const std::string GetRequsetIdString() const;
```

接口参数

无。

使用样例

```
mindie_llm::InferRequestId runtimeReqId1(0);
string value = runtimeReqId1.GetRequsetIdString();
```

返回值

获取请求ID的STRING类型。

5.1.1.2.6 Status

类说明

状态对象接口。

Status 接口

接口功能

- 通过错误码创建状态对象，错误码参考**5.1.1.3.4 Code枚举类**。
- 通过错误码和string类型的msg创建状态对象。
- 通过Error对象创建状态，Error对象参考**5.1.1.2.4 Error**。

接口格式

```
explicit Status(Error::Code code = Error::Code::OK) noexcept;
explicit Status(Error::Code code, const std::string &msg);
explicit Status(const Error &error);
```

接口参数

参数	是否必选	说明	取值要求
code	必选	错误码。	• Code::OK • Code::ERROR • Code::INVALID • Code::NOT_FOUND

参数	是否必选	说明	取值要求
code	必选	错误码。	• Code::OK • Code::ERROR • Code::INVALID • Code::NOT_FOUND
msg	必选	错误信息。	合法字符串类型。

参数	是否必选	说明	取值要求
error	必选	错误信息返回对象。	合法错误信息返回对象，Error类请参考5.1.1.2.4 Error。

使用样例

```
Status(Error::Code::OK);

Status(Error::Code::ERROR, "Engine init model failed: new modelBackend failed");

Status CommonErrorToStatus(const Error &error)
{
    return Status(error);
}
```

返回值

返回状态对象。

IsOk 接口

接口功能

判断状态对象状态是否是成功的。

接口格式

```
bool IsOk() const;
```

接口参数

无。

使用样例

```
status = Status(Error::Code::ERROR, "Engine init model failed: new modelBackend failed");
if (!status.IsOk()) {
    return;
}
```


返回值

- true：返回正常，代表返回状态成功。
- false：返回异常，代表返回状态失败。

StatusCode 接口

接口功能

获取状态对象的错误码。

接口格式

```
Error::Code StatusCode() const;
```

接口参数

无。

使用样例

```
auto status = Status(Error::Code::ERROR, "Engine init model failed: new modelBackend failed");
auto code = status.StatusCode();
```

返回值

错误码枚举。

StatusMsg 接口

接口功能

获取状态对象的错误信息。

接口格式

```
std::string StatusMsg() const;
```

接口参数

无。

使用样例

```
auto status = Status(Error::Code::ERROR, "Engine init model failed: new modelBackend failed");
string string = status.StatusMsg();
```

返回值

错误信息，string类型。

operator 操作符重载接口

接口功能

重载状态信息对象==操作符，实现若两个状态对象判断是否相等时，只需比较错误码是否相同即可。

接口格式

```
bool operator==(const Status& other) const;
```

接口参数

参数	是否必选	说明	取值要求
other	必选	与本状态对象比较的另外一个状态对象。	Status类，状态对象。

使用样例

```
auto status1 = Status(Error::Code::ERROR, "Engine init model failed: new modelBackend failed");
auto status2 = Status(Error::Code::ERROR, "Engine init model failed: new modelBackend failed");
bool isEqual = status1 == status2 ;
```

返回值

bool类型，两个对象是否相等。

5.1.1.3 结构体和枚举说明

5.1.1.3.1 InferDataType

接口功能

提供数据类型定义。

接口格式

```
enum class InferDataType {
    TYPE_INVALID = 0,
    TYPE_BOOL = 1,
    TYPE_UINT8 = 2,
    TYPE_UINT16 = 3,

    TYPE_UINT32 = 4,
    TYPE_UINT64 = 5,
    TYPE_INT8 = 6,
    TYPE_INT16 = 7,
    TYPE_INT32 = 8,
    TYPE_INT64 = 9,
    TYPE_FP16 = 10,
    TYPE_FP32 = 11,
    TYPE_FP64 = 12,
    TYPE_STRING = 13,
    TYPE_BF16 = 14,
    TYPE_BUTT,
};
```


5.1.1.3.2 StatusResponseType 枚举类

枚举类功能

StatusResponseType类型枚举。

枚举类格式

```
enum class StatusResponseType {
    CONTROL_SIGNAL_STATUS = 0,
    REQUEST_ENQUEUE_STATUS = 1,
};
```

枚举值

枚举名	说明
CONTROL_SIGNAL_STATUS	控制信号状态。
REQUEST_ENQUEUE_STATUS	请求入队状态。

5.1.1.3.3 BYTE_SIZE_MAP

枚举类功能

提供数据类型大小。

枚举类格式

```
const std::unordered_map<InferDataType, size_t> BYTE_SIZE_MAP = {
    {InferDataType::TYPE_INVALID, 0},
    {InferDataType::TYPE_BOOL, sizeof(bool)},
    {InferDataType::TYPE_UINT8, sizeof(uint8_t)},
    {InferDataType::TYPE_UINT16, sizeof(uint16_t)},
    {InferDataType::TYPE_UINT32, sizeof(uint32_t)},
    {InferDataType::TYPE_UINT64, sizeof(uint64_t)},
    {InferDataType::TYPE_INT8, sizeof(int8_t)},
    {InferDataType::TYPE_INT16, sizeof(int16_t)},
    {InferDataType::TYPE_INT32, sizeof(int32_t)},
    {InferDataType::TYPE_INT64, sizeof(int64_t)},
    {InferDataType::TYPE_FP16, sizeof(int16_t)},    // float16 类型不一定支持
    {InferDataType::TYPE_FP32, sizeof(float)},
    {InferDataType::TYPE_FP64, sizeof(double)},
    {InferDataType::TYPE_STRING, 0},                // 长度不确定
    {InferDataType::TYPE_BF16, sizeof(int16_t)},    // bfloat16 类型不一定支持
};
```

5.1.1.3.4 Code 枚举类

枚举类功能

错误码类型枚举。

枚举类格式

```
enum class Code {
      OK,
```



```
    ERROR,
    INVALID_ARG,
    NOT_FOUND,
};
```

枚举值

枚举名	说明
OK	操作成功。
ERROR	操作失败。
INVALID_ARG	非法参数。
NOT_FOUND	无法找到指定操作。

5.1.1.3.5 NodeHealthStatus 枚举类

枚举类功能

节点的链接状态类枚举。

枚举类格式

```
enum class NodeHealthStatus {
    READY,
    ABNORMAL
};
```

枚举值

枚举名	说明
READY	正常
ABNORMAL	异常

5.1.1.3.6 Compare 结构体

结构体功能

比较请求体对象的序列号大小，根据第一个参数请求体数据类型进行区分。若请求体序列号类似为string类型，则比较请求体对象序列号的hash值大小；若请求体序列号对象为数字类型，则比较其数学大小。最后返回两个请求体的数学大小关系，若结果为true，第一个参数的请求对象序列号值大；若为false，第一个参数的请求体对象序列号值小于等于第二个参数请求体对象序列号值。

结构体格式

```
struct Compare {
    bool operator () (const InferRequestId &lhs, const InferRequestId &rhs) const
    {
```



```
            if (lhs.Type() == InferRequestId::DataType::STRING) {
                return std::hash<std::string>()(lhs.StringValue()) < std::hash<std::string>()(rhs.StringValue());
            } else {
                return lhs.UnsignedIntValue() < rhs.UnsignedIntValue();
            }
        }
};
```

结构体参数

无。

5.1.1.3.7 Datatype 枚举类

枚举类功能

InferRequestId类中的DataType枚举，表述请求ID是数值还是字符串类型。

枚举类格式

```
enum class DataType {
    UINT64,
    STRING
};
```

枚举值

枚举名	说明
UINT64	正常
STRING	异常

5.1.1.3.8 MemType 枚举类

枚举类功能

MemType枚举类，表示Mem的类型。

枚举类格式

```
enum class MemType {
    HOST_MEM = 0,
};
```

枚举值

枚举名	说明
HOST_MEM	正常

5.1.1.3.9 InferReqType 枚举类

枚举类功能

InferReqType枚举类，表示请求的类型。

枚举类格式

```
enum class InferReqType {
    REQ_STAND_INFER = 0,
    REQ_PREFILL = 1,
    REQ_DECODE = 2
};
```

枚举值

枚举名	说明
REQ_STAND_INFER	正常
REQ_PREFILL	正常
REQ_DECODE	正常

5.1.1.3.10 Operation 枚举类

枚举类功能

Operation枚举类，表示请求停止的类型。

枚举类格式

```
enum class Operation {
    STOP = 1,
    RELEASE_KV = 2,
};
```

枚举值

枚举名	说明
STOP	请求被stopInfer停止推理的类型。
RELEASE_KV	请求在P节点释放完KV Cache的类型。

5.1.1.4 函数

5.1.1.4.1 GetRequestsCallback

函数功能

定义了一个函数对象类型GetRequestsCallback, 返回值为std::shared_ptr<InferRequest>。

函数格式

```
using GetRequestsCallback = std::function<std::vector<std::shared_ptr<InferRequest>>()>;
```

5.1.1.4.2 SendResponsesCallback

函数功能

定义了一个函数对象类型SendResponseCallback，它具有四个参数，类型为InferRequestId，TensorMap， bool和std::string&。

函数格式

```
using SendResponsesCallback = std::function<void(InferRequestId, const TensorMap&, bool, const std::string&)>;
```

5.1.1.4.3 SendStatusResponseCallback

函数功能

定义了一个函数对象类型SendStatusResponseCallback，它接受三个参数，类型为InferRequestId, Status和StatusResponseType并且没有返回值。

函数格式

```
using SendStatusResponseCallback = std::function<void(InferRequestId, Status, StatusResponseType)>;
```

5.1.1.4.4 ControlSignalCallback

函数功能

定义了一个函数对象类型ControlSignalCallback, 它接受一个参数，返回值类型为std::vector<std::pair<InferRequestId, Operation>>。

函数格式

```
using ControlSignalCallback = std::function<std::vector<std::pair<InferRequestId, Operation>>()>;
```

5.1.1.4.5 LlmManagerStatsCallback

函数功能

定义了一个函数对象类型LlmManagerStatsCallback，它接受一个参数，类型为const std::string&，并且没有返回值。

函数格式

```
using LlmManagerStatsCallback = std::function<void(const std::string&)>;
```


5.1.1.5 类型别名

5.1.1.5.1 TensorPtr

类型别名功能

定义了一个TensorPtr类型别名。

类型别名格式

```
using TensorPtr = std::shared_ptr<InferTensor>;
```

5.1.1.5.2 TensorMap

类型别名功能

定义了一个TensorMap的类型别名。

类型别名格式

```
using TensorMap = std::unordered_map<std::string, std::shared_ptr<InferTensor>>;
```

5.2 LLM Manager API 参考（Python）

5.2.1 LLM Manger 提供的 Python 接口

5.2.1.1 使用场景

针对自研服务化MindIE Service、三方服务化Triton，提供模型+调度能力，支持Continuous Batching动态调度。

5.2.1.2 类参考

5.2.1.2.1 InferRequest

类说明

InferenceRequest推理引擎功能请求实现类，提供Request初始化，Tensor设置，获取请求信息等功能入口。

InferRequest 接口

接口功能

构造函数。

C++函数原型

```
explicit InferRequest(InferRequestId requestId);
```


Python 函数

```
InferRequest(request_id)
```

接口参数

参数	是否必选	说明	取值要求
request_id	是	请求ID。	合法的InferRequestId类型，请参考5.2.1.2.5 InferRequestId。

使用样例

```
id = '123'
request_id = llm_manager_python.InferRequestId(id)
infer_request = llm_manager_python.InferRequest(request_id)
```

add_tensor 接口

接口功能

添加Tensor。

C++函数原型

```
Status AddTensor(const std::string& tensorName, TensorPtr &tensor)
```

Python 函数

```
add_tensor(tensor_name, tensor)
```

接口参数

参数	是否必选	说明	取值要求
tensor_name	是	tensor名字。	合法的字符串类型string类型。
tensor	是	tensor内容。	合法的InferTensor类型，请参考5.2.1.2.3 InferTensor。

使用样例

```
id = '123'
request_id = llm_manager_python.InferRequestId(id)
infer_request = llm_manager_python.InferRequest(request_id)
name = "INPUT_IDS"
shape = np.array([1,1], dtype=np.int64)
runtime_tensor = llm_manager_python.InferTensor(name, shape,
llm_manager_python.InferDataType.TYPE_INT64)
infer_request.add_tensor(name, runtime_tensor)
```


set_tensor 接口

接口功能

请求的设置Tensor接口。

C++函数原型

```
void SetTensor(const std::string& tensorName, TensorPtr &tensor)
```

Python 函数

```
set_tensor(tensor_name, tensor)
```

接口参数

参数	是否必选	说明	取值要求
tensor_name	是	tensor名字。	合法的字符串类型string类型。
tensor	是	tensor内容。	合法的InferTensor类型，请参考**5.2.1.2.3 InferTensor**。

使用样例

```
id = '123'
request_id = llm_manager_python.InferRequestId(id)
infer_request = llm_manager_python.InferRequest(request_id)
name = "INPUT_IDS"
shape = np.array([1,1], dtype=np.int64)
runtime_tensor = llm_manager_python.InferTensor(name, shape,
llm_manager_python.InferDataType.TYPE_INT64)
infer_request.set_tensor(name, runtime_tensor)
```

get_tensor_by_name 接口

接口功能

根据Name获取Tensor。

C++函数原型

```
Status GetTensorByName(const std::string& tensorName, TensorPtr &tensor)
```

Python 函数

```
get_tensor_by_name(tensor_name, tensor)
```


接口参数

参数	是否必选	说明	取值要求
tensor_name	是	tensor名字。	合法的字符串类型string类型。
tensor	是	获取得到的tensor。	合法的InferTensor类型，请参考 **5.2.1.2.3 InferTensor**。

使用样例

```
name = "INPUT_IDS"
shape = np.array([1,1], dtype=np.int64)
runtime_tensor = llm_manager_python.InferTensor(name, shape,
llm_manager_python.InferDataType.TYPE_INT64)
status = infer_request.get_tensor_by_name(name, runtime_tensor)
```

del_tensor_by_name 接口

接口功能

根据Name删除Tensor。

C++函数原型

```
Status DelTensorByName(const std::string &name)
```

Python 函数

```
del_tensor_by_name(tensor_name)
```

接口参数

参数	是否必选	说明	取值要求
name	是	tensor名字。	合法的字符串类型string类型。

使用样例

```
name = "INPUT_IDS"
status = infer_request.del_tensor_by_name(name)
```

get_request_id 接口

接口功能

获取请求ID。

C++函数原型

```
InferRequestId GetRequestId() const
```

Python 函数

```
get_request_id()
```

接口参数

无。

使用样例

```
request_id = infer_request.get_request_id()
```

set_max_output_len 接口

接口功能

设置最大输出长度。

C++函数原型

```
Status SetMaxOutputLen(uint32_t maxOutputLen)
```

Python 函数

```
set_max_output_len(max_output_len)
```

接口参数

参数	是否必选	说明	取值要求
max_output_len	是	最大输出长度。	满足C++中uint32_t范围的int类型。

使用样例

```
max_output_len = 512
status = infer_request.set_max_output_len(max_output_len)
```

get_max_output_len 接口

接口功能

获取最大输出长度。

C++函数原型

```
uint32_t GetMaxOutputLen() const
```

Python 函数

```
get_max_output_len()
```


接口参数

无。

使用样例

```
output_len = infer_request.get_max_output_len()
```

immutable_inputs 接口

接口功能

获取request的输入。

C++函数原型

```
const TensorMap &ImmutableInputs() const
```

Python 函数

```
immutable_inputs()
```

接口参数

无。

使用样例

```
tensor_input = infer_request.immutable_inputs()
```

5.2.1.2.2 LlmManager

类说明

Llmmanager推理引擎功能管理类，提供Manager模块的初始化和终止。

LlmManager 接口

接口功能

默认构造函数。

C++函数原型

```
LlmManager(const std::string &llmConfigPath, MindIE_LLM::GetRequestsCallback getRequest,
    MindIE_LLM::SendResponsesCallback sendResponse, MindIE_LLM::ControlSignalCallback
controlCallback,
    MindIE_LLM::LlmManagerStatsCallback statusCallback,
    MindIE_LLM::SendStatusResponseCallback statusResponseCallback)
```

Python 函数

```
LlmManager(config_path, get_request, send_response, control_callback, status_callback,
status_response_callback)
```


接口参数

参数	是否必选	说明	取值要求
config_path	是	config路径。	合法的config路径,字符类型参数。
get_request	是	GetRequests回调函数。	Callable[[], List['InferRequest']] 输出为InferRequest类型的列表。
send_response	是	SendResponses回调函数。	Callable[[InferRequestId, TensorMap, bool, str], None] 输入依次为InferRequestId类型，TensorMap类型，bool类型，字符类型；无输出。
control_callback	是	ControlSignal回调函数。	Callable[[InferRequestId, Status, int], None] 输入依次为InferRequestId类型，字符类型，Status类型，StatusResponseType类型；无输出。
status_callback	是	LlmManagerStats回调函数。	Callable[[], Set[InferRequestId]] 无输入；输出为InferRequestId的集合。
status_response_callback	是	SendStatusResponse回调函数。	Callable[[str], None] 输入为字符类型；无输出。

使用样例

```
config_path = '/path/to/config.json'
def get_requests_callback():
    ...
    pass
def send_response_callback():
    ...
    pass
def control_callback():
    ...
    pass
```



```
def status_callback():
    ...
    pass
def status_response_callback():
    ...
    pass

llm_manager = LlmManager(config_path, get_requests_callback, send_response_callback, control_callback,
status_callback, status_response_callback)
```

shutdown 接口

接口功能

提供终止LLMManager的功能。

C++函数原型

```
void Shutdown()
```

Python 函数

```
shutdown()
```

接口参数

无。

使用样例

```
llm_manager.shutdown()
```

get_max_position_embeddings 接口

接口功能

获取maxPositionEmbeddings数值。

C++函数原型

```
uint32_t GetMaxPositionEmbeddings() const
```

Python 函数

```
get_max_position_embeddings()
```

接口参数

无。

使用样例

```
nums = llm_manager.get_max_position_embeddings()
```


init 接口

接口功能

LlmManager的初始化接口。

C++函数原型

```
Status Init(uint32_t modelInstanceId, std::set<size_t> npuDeviceIds)
```

Python 函数

```
init(model_instance_id, npu_device_ids)
```

接口参数

参数	是否必选	说明	取值要求
model_instance_id	是	模型实例Id。	满足C++中uint32范围的int类型。
npu_device_ids	是	npu设备的ids。	合法的set类型。

使用样例

```
model_instance_id = 0
npu_device_ids = set([0,1,2,3])
llm_manager.init(model_instance_id, npu_device_ids)
```

返回值

Status对象。

5.2.1.2.3 InferTensor

类说明

Llmmanager推理引擎功能数据类，提供输入和输出的基类。

InferTensor 接口

接口功能

构造函数。

C++函数原型

```
InferTensor() = default;
InferTensor(std::string name, InferDataType dataType, std::vector<int64_t> dataShape);
```

Python 函数

```
InferTensor()
InferTensor(name, data_type, data_shape)
```


接口参数

参数	是否必选	说明	取值要求
name	是	tensor名字。	合法的字符串类型string类型。
data_type	是	tensor数据类型。	合法的InferDataType类型。
data_shape	是	tensor数据维度。	合法的满足C++ int64范围的int列表。

使用样例

```
runtime_tensor1 = llm_manager_python.InferTensor()
name = "INPUT_IDS"
shape = np.array([1,1], dtype=np.int64)
runtime_tensor2 = llm_manager_python.InferTensor(name, shape,
llm_manager_python.InferDataType.TYPE_INT64)
```

get_shape 接口

接口功能

获取张量对象shape。

C++函数原型

```
const std::vector<int64_t> &GetShape() const
```

Python 函数

```
get_shape()
```

接口参数

无。

使用样例

```
shape = infer_tensor.get_shape()
```

get_size 接口

接口功能

获取张量对象数据大小。

C++函数原型

```
size_t GetSize() const
```


Python 函数

```
get_size()
```

接口参数

无。

使用样例

```
tensor_size = infer_tensor.get_size()
```

get_data_type 接口

接口功能

获取张量对象数据类型。

C++函数原型

```
InferDataType GetDataType() const
```

Python 函数

```
get_data_type()
```

接口参数

无。

使用样例

```
data_type = infer_tensor.get_data_type()
```

get_mem_type 接口

接口功能

获取MemType。

C++函数原型

```
MemType GetMemType() const
```

Python 函数

```
get_mem_type()
```

接口参数

无。

使用样例

```
mem_type = infer_tensor.get_mem_type()
```


get_data 接口

接口功能

获取Data。

C++函数原型

```
void* GetData() const;
```

Python 函数

```
get_data()
```

接口参数

无。

使用样例

```
data = infer_tensor.get_data()
```

get_name 接口

接口功能

获取tensor名字。

C++函数原型

```
const std::string &GetName() const
```

Python 函数

```
get_name()
```

接口参数

无。

使用样例

```
name = infer_tensor.get_name()
```

allocate 接口

接口功能

为输入张量对象分配内存空间，用于接收输入数据。

C++函数原型

```
bool Allocate(size_t size)
```

Python 函数

```
allocate(size)
```


接口参数

参数	是否必选	说明	取值要求
size	是	合法的内存空间大小。	合法的内存空间大小。

使用样例

```
status = infer_tensor.allocate(512)
```

set_buffer 接口

接口功能

为输入张量对象设置数据，数据大小和是否需要释放内存的标识。

C++函数原型

```
void SetBuffer(const void* buffer, size_t tensorbyteSize, bool tensorNeedRelease)
```

Python 函数

```
set_buffer(tensor_data, tensor_need_release)
```

接口参数

参数	是否必选	说明	取值要求
tesnor_data	是	张量数据。	合法的满足python缓冲区协议的类型对象。
tensor_need_release	是	是否需要释放内存的标识。	合法的bool类型。

使用样例

```
data = [0,1,2,3]
infer_tensor.set_buffer(data, False)
```

set_release 接口

接口功能

为输入张量对象设置是否需要释放数据内存标识。

C++函数原型

```
void SetRelease(bool releaseFlag)
```


Python 函数

```
set_release(release_flag)
```

接口参数

参数	是否必选	说明	取值要求
release_flag	是	是否需要释放数据内存的标识。	合法的bool类型。

使用样例

```
infer_tensor.set_release(False)
```

release 接口

接口功能

释放输入张量对象内存。

C++函数原型

```
void Release()
```

Python 函数

```
release()
```

接口参数

无。

使用样例

```
infer_tensor.release(
```

5.2.1.2.4 Error

类说明

推理引擎错误码类，提供错误码的创建、获取等功能。

Error(code)接口

接口功能

根据枚举Code创建返回信息对象。

C++函数原型

```
explicit Error(Code code = Code::OK);
```


Python 函数

```
Error(code)
```

接口参数

参数	是否必选	说明	取值要求
code	必选	错误码。	Code枚举类型。

使用样例

```
code = llm_manager_python.Code.OK
error = Error(code)
```

Error(code, msg)接口

接口功能

根据枚举Code、错误信息msg，创建返回信息对象。

C++函数原型

```
explicit Error(Code code, std::string msg);
```

Python 函数

```
Error(code, msg)
```

接口参数

参数	是否必选	说明	取值要求
code	必选	错误码。	Code枚举类型。
msg	必选	错误信息。	合法字符串string类型。

使用样例

```
msg = "message"
error = Error(code, msg)
```

error_code 接口

接口功能

获取返回对象内的错误码。

C++函数原型

```
Code ErrorCode() const;
```


Python 函数

```
error_code()
```

接口参数

无。

使用样例

```
error_code = error.error_code()
```

message 接口

接口功能

获取返回对象内的错误码信息。

C++函数原型

```
const std::string &Message() const;
```

Python 函数

```
message()
```

接口参数

无。

使用样例

```
message = error.message()
```

is_ok 接口

接口功能

获取返回对象状态，是否是正常状态。

C++函数原型

```
bool IsOk() const;
```

Python 函数

```
is_ok()
```

接口参数

无。

使用样例

```
status = error.is_ok()
```


5.2.1.2.5 InferRequestId

类说明

推理引擎请求ID实现类，提供请求ID的初始化。

InferRequestId 接口

接口功能

默认构造函数，根据序列requestLabel值作为唯一序列号的创建请求体对象。

C++函数原型

```
explicit InferRequestId();
```

Python 函数

```
InferRequestId()
```

使用样例

```
request_id = llm_manager_python.InferRequestId()
```

InferRequestId(request_label)接口

接口功能

构造函数，根据序列request_label值作为唯一序列号的创建请求体对象。

C++函数原型

```
explicit InferRequestId(const std::string requestLabel);
```

Python 函数

```
InferRequestId(request_label)
```

接口参数

参数	是否必选	说明	取值要求
request_label	非必选	推理请求对象的唯一性标识，序列号。	合法的字符类型string类型请求序列号。

使用样例

```
id = '123'
request_id = llm_manager_python.InferRequestId(id)
```


InferRequestId(request_index)接口

接口功能

构造函数，根据序列request_index值作为唯一序列号的创建请求体对象。

C++函数原型

```
explicit InferRequestId(uint64_t requestIndex);
```

Python 函数

```
InferRequestId(request_index)
```

接口参数

参数	是否必选	说明	取值要求
request_index	非必选	推理请求对象的唯一序列号，uint64类型的。	合法的符合C++ uint64类型范围的int类型请求序列号。

使用样例

```
request_id = llm_manager_python.InferRequestId(123)
```

type 接口

接口功能

获取请求体对象唯一的序列号标识的数据类型。

C++函数原型

```
DataType Type() const;
```

Python 函数

```
type()
```

接口参数

无。

使用样例

```
type = request_id.type()
```

string_value 接口

接口功能

获取请求体字符类型的序列号，多为请求体序列号为string类型请求体使用。

C++函数原型

```
const std::string &StringValue() const;
```

Python 函数

```
string_value()
```

接口参数

无。

使用样例

```
str_val = request_id.string_value()
```

unsigned_int_value 接口

接口功能

获取请求体数字类型的序列号，多为请求体序列号为uint64_t类型请求体使用。

C++函数原型

```
uint64_t UnsignedIntValue() const;
```

Python 函数

```
unsigned_int_value()
```

接口参数

无。

使用样例

```
val = request_id.unsigned_int_value()
```

5.2.1.2.6 Status

类说明

状态对象接口。

Status(code)接口

接口功能

根据枚举Code创建返回信息对象。

C++函数原型

```
explicit Status(Error::Code code = Error::Code::OK) noexcept
```

Python 函数

```
Status(code)
```


接口参数

参数	是否必选	说明	取值要求
code	必选	错误码。	Code枚举类型，请参考5.2.1.3.4 Code枚举类。

使用样例

```
code = llm_manager_python.Code.OK
status = Status(code)
```

Status(code, msg)接口

接口功能

根据枚举Code、错误信息msg，创建返回信息对象。

C++函数原型

```
explicit Status(Error::Code code, const std::string &msg)
```

Python 函数

```
Status(code, msg)
```

接口参数

参数	是否必选	说明	取值要求
code	必选	错误码。	Code枚举类型，请参考5.2.1.3.4 Code枚举类。
msg	必选	错误信息。	合法字符串string类型。

使用样例

```
msg = "message"
status = Status(code, msg)
```

Status(error)接口

接口功能

获取返回对象内的错误码。

C++函数原型

```
explicit Status(const Error &error)
```


Python 函数

```
Status(error)
```

接口参数

参数	是否必选	说明	取值要求
error	必选	错误码。	Error类对象。

使用样例

```
code = llm_manager_python.Code.OK
error = Error(code)
status = Status(error)
```

is_ok 接口

接口功能

判断状态对象状态是否是成功的。

C++函数原型

```
bool IsOk() const;
```

Python 函数

```
is_ok()
```

接口参数

无。

使用样例

```
ret = status.is_ok()
```

status_code 接口

接口功能

获取状态对象的错误码。

C++函数原型

```
Error::Code StatusCode() const;
```

Python 函数

```
status_code()
```

接口参数

无。

使用样例

```
ret_code = status.status_code()
```

status_msg 接口

接口功能

获取状态对象的错误信息。

C++函数原型

```
std::string StatusMsg() const;
```

Python 函数

```
status_msg()
```

接口参数

无。

使用样例

```
ret_msg = status.status_msg()
```

5.2.1.3 结构体和枚举说明

5.2.1.3.1 InferDataType 枚举类

接口功能

提供数据类型定义。

C++枚举类格式

```
enum class InferDataType {
    TYPE_INVALID = 0,
    TYPE_BOOL = 1,
    TYPE_UINT8 = 2,
    TYPE_UINT16 = 3,

    TYPE_UINT32 = 4,
    TYPE_UINT64 = 5,
    TYPE_INT8 = 6,
    TYPE_INT16 = 7,
    TYPE_INT32 = 8,
    TYPE_INT64 = 9,
    TYPE_FP16 = 10,
    TYPE_FP32 = 11,
    TYPE_FP64 = 12,
    TYPE_STRING = 13,
    TYPE_BF16 = 14,
    TYPE_BUTT,
};
```

Python 接口格式

```
InferDataType.TYPE_INVALID
InferDataType.TYPE_BOOL
```



```
InferDataType.TYPE_UINT8
InferDataType.TYPE_UINT16
InferDataType.TYPE_UINT32
InferDataType.TYPE_UINT64
InferDataType.TYPE_INT8
InferDataType.TYPE_INT16
InferDataType.TYPE_INT32
InferDataType.TYPE_INT64
InferDataType.TYPE_FP16
InferDataType.TYPE_FP32
InferDataType.TYPE_FP64
InferDataType.TYPE_STRING
InferDataType.TYPE_BF16
InferDataType.TYPE_BUTT
```

5.2.1.3.2 StatusResponseType 枚举类

枚举类功能

StatusResponseType类型枚举。

C++枚举类格式

```
enum class StatusResponseType {
    CONTROL_SIGNAL_STATUS = 0,
    REQUEST_ENQUEUE_STATUS = 1,
};
```

Python 接口格式

```
StatusResponseType.CONTROL_SIGNAL_STATUS
StatusResponseType.REQUEST_ENQUEUE_STATUS
```

枚举值

枚举名	说明
CONTROL_SIGNAL_STATUS	控制信号状态。
REQUEST_ENQUEUE_STATUS	请求入队状态。

5.2.1.3.3 MemType 枚举类

枚举类功能

内存类型。

C++枚举类格式

```
enum class MemType {
        HOST_MEM = 0,
};
```

Python 接口格式

```
MemType.HOST_MEM
```


5.2.1.3.4 Code 枚举类

枚举类功能

错误码类型枚举。

C++枚举类格式

```
enum class Code {
    OK,
    ERROR,
    INVALID_ARG,
    NOT_FOUND,
};
```

Python 接口格式

```
Code.OK
Code.ERROR
Code.INVALID_ARG
Code.NOT_FOUND
```

枚举值

枚举名	说明
OK	操作成功。
ERROR	操作失败。
INVALID_ARG	非法参数。
NOT_FOUND	无法找到指定操作。

5.2.1.3.5 DataType 枚举类

枚举类功能

InferRequestId中数据类型。

C++枚举类格式

```
enum class DataType {
    UINT64,
    STRING
};
```

Python 接口格式

```
DataType.UINT64
DataType.STRING
```

5.2.1.4 函数

5.2.1.4.1 tensor_to_numpy

函数功能

定义了类型转换函数，接受一个InferTensor 类型参数，进行类型转换，返回值为py::array。

C++函数原型

```
py::array TensorToNumpy(const InferTensor &tensor);
```

Python 函数

```
tensor_to_numpy(tensor)
```

5.3 Text Generator API 参考（Python）

5.3.1 generator_torch.py 提供 Python 接口

5.3.1.1 使用场景

- generator接口主要面向TGI、vLLM框架，提供模型管理、加载、推理和后处理功能。
- 提供对接开源三方服务化TGI、vLLM能力，基于pytorch框架，仅支持backendType='atb'场景，实现大模型推理、后处理功能。
- 文件路径：mindie_llm/text_generator/adapter/generator_torch.py。

5.3.1.2 __init__接口

接口功能

提供Generator初始化，包括Model和Sampler初始化，以及权重加载、KV Cache分配等功能。

接口实现

```
class GeneratorTorch(GeneratorBackend):
    cache_pool: CachePool = None
    def __init__(self, model_config):
        super().__init__(model_config)
        self.tokenizer = self.model_wrapper.tokenizer
        self.device = self.model_wrapper.device
        self.rank = self.model_wrapper.rank
```

参数说明

model_config为模型配置，支持字典格式输入，主要配置信息参考如下：

参数名称	是否必选	配置名称	类型	默认值	描述
model_config	必选	backend_type	string	None	模型后端类型，支持'atb'和'ms'。
		world_size	int	None	TP并行数。
		rank	int	None	当前进程在TP并行进程中的序号，从0开始计数，此值应小于world_size。
		npu_device_id	int	None	当前进程所使用的可见npu设备序号，此值应小于环境变量ASCEND_RT_VISIBLE_DEVICES设置的可见设备数量。（未设置ASCEND_RT_VISIBLE_DEVICES时，默认所有设备可见。）
		num_threads	int	8	后处理线程数。
	可选	load_tokenizer	bool	True	是否加载tokenizer，设为False则不加载默认的tokenizer。
		tokenizer_path	string	None	自定义加载tokenizer的路径，为None时默认使用模型权重路径。

安全说明

调用API 出现异常情况时，会直接抛出异常信息。例如：

- 对于加载的权重文件较大的情况，会直接抛NPU Out Of Memory(OOM)异常, 异常信息为：RuntimeError:NPU out of memory. Tried to allocate XXX MiB。
- 模型权重文件中未配置torch_dtype字段，会直接抛出异常，异常信息为：unsupported type: XXX, 此类型从权重文件config.json中的`torch_dtype`字段中获取；若config.json中不存在此字段，请新增；当前此字段仅接受`float16`和`bfloat16`两种类型，各模型具体支持的类型不同，请参考模型README文件。
- 模型权重文件中配置的torch_dtype字段不支持时，会直接抛出异常，异常信息为：unsupported type: XXX, 当前仅支持`float16`类型，请修改权重文件config.json中的`torch_dtype`字段。

5.3.1.3 to_tensor 接口

接口功能

将数据转换为torch.tensor。

接口实现

```
def to_tensor(self, data):
    return torch.tensor(data, device=self.device)
```

参数说明

参数名称	是否必选	类型	默认值	描述
data	是	ndarry	-	待转为torch.tensor的数据。

5.3.1.4 forward_tensor 接口

接口功能

模型推理接口，采用Page Attention算子实现高性能推理，依赖上层调度模块实现对模型KV Cache管理能力，支持Continuous Batching。

接口实现

```
def forward_tensor(
            self,
            input_ids: torch.Tensor,
            position_ids: torch.Tensor,
            is_prefill: bool,
            kv_cache: List[Tuple[torch.Tensor, torch.Tensor]],
            block_tables: torch.Tensor,
            slots: torch.Tensor,
            input_lengths: torch.Tensor,
            max_seq_len: int,
            lm_head_indices: Optional[torch.Tensor] = None,
            **kwargs):
        logits = self.model_wrapper.forward_tensor(
            input_ids=input_ids,
            position_ids=position_ids,
            is_prefill=is_prefill,
            kv_cache=kv_cache,
            block_tables=block_tables,
            slots=slots,
            input_lengths=input_lengths,
            max_seq_len=max_seq_len,
            lm_head_indices=lm_head_indices,
            **kwargs,
        )
        return logits
```

参数说明

参数名称	是否必选	类型	默认值	描述	安全声明
input_ids	必选	torch.Tensor	-	输入经过tokenizer后，每个token在词表中的索引。	推理强依赖数据的合法性，需由用户保证。
position_ids	必选	torch.Tensor	-	Token的位置索引。	
is_prefill	必选	bool	-	是否为推理首token阶段。	

参数名称	是否必选	类型	默认值	描述	安全声明
kv_cache	必选	List[Tuple[torch.Tensor, torch.Tensor]]	-	KV 缓存。	
block_tables	必选	torch.Tensor	-	存储每个token和它所使用的KV Cache块之间的映射。	
slots	必选	torch.Tensor	-	存储每个KV Cache块中实际使用的slots的序号。	
input_lengths	必选	torch.Tensor	-	batch中每个query的长度。（输入+当前输出的长度。）	
max_seq_len	必选	int	-	模型支持的最大上下文长度。（即最大可支持的输入+输出的长度。）	
lm_head_indices	可选	Optional[torch.Tensor]	None	设置此值可根据索引选择性地输出logits。	
**kwargs中的q_lens	可选	List[int]	-	在并行解码场景下，单次decode增量输入的token长度。	
**kwargs中的spec_mask	可选	torch.Tensor	-	并行解码场景下生成的mask。	

5.3.1.5 forward 接口

接口功能

第三方服务调用此接口执行模型推理。

接口实现

```
def forward(self, model_inputs, **kwargs):
    logits = self.model_wrapper.forward(model_inputs, self.cache_pool.npu_cache, **kwargs)
    return logits
```


参数说明

参数名称	是否必选	类型	默认值	描述	安全声明
model_inputs	必选	ModelInput	-	模型的输入参数。	推理强依赖数据的合法性，需由用户保证。
**kwargs中的q_lens	可选	List[int]	-	在并行解码场景下，单次decode增量输入的token长度。	
**kwargs中的spec_mask	可选	torch.Tensor	-	并行解码场景下生成的mask。	

补充说明

forward接口中的model_inputs参数通过ModelInput类进行构造。

文件路径：mindie_llm/text_generator/utils/input.py。

ModelInput类的初始化参数说明如下：

参数名称	是否必选	类型	默认值	描述	安全声明
input_ids	必选	np.ndarray	-	输入经过tokenizer后，每个token在词表中的索引。	推理强依赖数据的合法性，需由用户保证。
position_ids	必选	np.ndarray或None	-	Token的位置索引。	
is_prefill	必选	bool	-	是否为推理首token阶段。	
block_tables	必选	np.ndarray	-	存储每个token和它所使用的KV Cache块之间的映射。	
slots	必选	np.ndarray	-	存储每个KV Cache块中实际使用的slots的序号。	
context_length	必选	np.ndarray或List[int]	-	batch中每个query的长度。（输入+当前输出的长度。）	
max_seq_len	必选	int	-	模型支持的最大上下文长度。（即最大可支持的输入+输出的长度。）	
prefill_head_indices	必选	np.ndarray或None	-	设置此值可根据索引选择性地输出logits。	

参数名称	是否必选	类型	默认值	描述	安全声明
query_length	可选	Optional[np.ndarray]	None	请求的长度。	

5.3.1.6 build_inputs 接口

接口功能

基于输入的conversations: List[List[Dict[str, str]]]，基于模型的tokenizer，提供多轮对话token_id生成和拼接。（继承自GeneratorBackend，属于内部调度接口，暂不支持对外功能。）

接口实现

```
def build_inputs(self, conversations: List[List[Dict[str, str]]], **kwargs):
    return [self.model_wrapper.make_context(conversation, **kwargs) for conversation in conversations]
```

5.3.1.7 sample 接口

接口功能

此接口对推理输出的logits进行后处理。

接口实现

```
def sample(self, logits, sampling_data, sampling_param):
        _, next_tokens = self.sampler(logits, sampling_data, sampling_param)
        return next_tokens
```

参数说明

参数名称	是否必选	类型	默认值	描述
logits	是	Any	-	对应后端类型的Tensor，当前支持torch.Tensor或mindspore.Tensor。未经过softmax函数处理的网络输出结果，通常表示每个类别的得分或概率。
sampling_data	否	SamplingData	None	SamplingData类实例，用于传入batch后处理元数据。通过SamplingData类方法。SamplingData.from_numpy可以创建类实例sampling_data。

参数名称	是否必选	类型	默认值	描述
sampling_param	否	SamplingParam	None	SamplingParam类实例，用于传入batch后处理参数。通过SamplingParam类方法。SamplingParam.from_numpy可以创建类实例sampling_param。

补充说明

sampling_data通过SamplingData.from_numpy进行构造。

文件路径：mindie_llm/text_generator/utils/sampling_metadata.py。

表 5-1 SamplingData.from_numpy 接口参数说明

参数名称	是否必选	类型	默认值	描述
all_input_ids	否	np.ndarray	None	二维int数组，包含每个请求所有输入加输出的token id。用于repetition_penalty计算。
output_ids	否	np.ndarray	None	二维int数组，包含每个请求所有输出的token id。用于frequency_penalty和presence_penalty计算。
to_tensor	否	callable	None	to_tensor方法，用于将np.ndarray生成不同后端的张量。若传入的all_input_ids或output_ids不为None，则本项必选。
is_prefill	否	bool	True	用于判断是否在生成第一个token。本项与request_ids成对传入，可触发缓存机制，提升性能。
request_ids	否	np.ndarray	None	一维int数组，批处理中所有请求的唯一标识。本项与is_prefill成对传入，可触发缓存机制，提升性能。

sampling_param通过SamplingParam.from_numpy进行构造。

文件路径：mindie_llm/text_generator/utils/sampling_metadata.py。

表 5-2 SamplingParam.from_numpy 接口参数说明

参数名称	是否必选	类型	默认值	描述
repetition_penalty	否	np.ndarray	None	一维float数组，对应批处理中每个请求的重复惩罚。
frequency_penalty	否	np.ndarray	None	一维float数组，对应批处理中每个请求的频率惩罚。
presence_penalty	否	np.ndarray	None	一维float数组，对应批处理中每个请求的存在惩罚。
temperature	否	np.ndarray	None	一维float数组，对应批处理中每个请求的温度。
top_k	否	np.ndarray	None	一维int数组，对应批处理中每个请求在采样时依次从最高概率选项中选择的数量。
top_p	否	np.ndarray	None	一维float数组，对应批处理中每个请求在采样时的累加概率阈值。
seed	否	np.ndarray	None	一维int数组，对应批处理中每个请求在采样时的随机种子。
do_sample	否	np.ndarray	None	一维bool数组，对应批处理中每个请求是否做采样。
to_tensor	否	callable	None	to_tensor方法，用于将np.ndarray生成不同后端的张量。若传入的任意一个参数不为None，则本项必选。

后处理参数数组元素数值说明：

针对单个request而言，目前支持惩罚参数和采样参数两类后处理参数，如表5-3及表5-4所示。

表 5-3 惩罚参数

参数名称	类型	取值要求	描述
repetition_penalty	float	> 0，建议不超过2	重复惩罚的参数，对输入输出中已存在的token施加除法级惩罚，1.0表示没有惩罚。
frequency_penalty	float	负数表示奖励，建议 > 0	频率惩罚，根据输出中已存在的token的出现频率施加减法级惩罚，0.0表示没有惩罚。
presence_penalty	float	负数表示奖励，建议 > 0	存在惩罚，对输出中已存在的token施加减法级惩罚，0.0表示没有惩罚。

说明

若模型路径下的config.json和generate_config.json中都没有配置pad_token_id时，需要手动添加pad_token_id。

可配置范围：[-1, vocab_size]，建议值：vocab_size。

表 5-4 采样参数

参数名称	类型	取值要求	描述
temperature	float	> 0，建议不超过2	控制生成文本的随机性，值越高文本越随机。
top_k	int	0 < top_k < 词表长度	限制每次生成时候概率最高的k个数量。
top_p	float	0.0 < top_p <= 1.0	选择概率总和达到p的所有选项，用于控制生成的多样性。
seed	int	>= 0	设置随机数种子，以确保结果的可重复性。
do_sample	bool	布尔值	决定是否使用抽样策略生成文本，而非选择概率最高的选项。 输入“False”时，若存在采样参数，则自动变为“True”。

5.3.1.8 update_cache_policy 接口

接口功能

根据cache_manager中的npu_mem和cpu_mem申请NPU和CPU对应的KV Cache。（属于内部调度接口，暂不支持对外功能。）

接口实现

```
def update_cache_policy(self, cache_manager):
    self.cache_pool = CachePool(cache_manager, self.device)
    self.cache_pool.allocate_cpu_cache()
    self.cache_pool.allocate_npu_cache()
```

参数说明

参数名称	是否必选	类型	默认值	描述
cache_manager	是	CacheManager	-	根据模型信息进行npu和cpu的block的计算，接口需配合llm的调度模块，当前作为内部实现。

5.3.1.9 swap_cache 接口

接口功能

NPU和CPU的KV Cache换入换出接口。（属于内部调度接口，暂不支持对外功能。）

接口实现

```
def swap_cache(self, swap_decision):
    swap_decision_tensor = torch.tensor(swap_decision, dtype=torch.int64, device=self.device)
    self.cache_pool.swap_cache(swap_decision_tensor)
```

参数说明

参数名称	是否必选	类型	默认值	描述
swap_decision	是	ndarray	-	换入换出的block的index以及切换方向。

5.3.1.10 clear_cache 接口

接口功能

清理cache。

接口实现

```
def clear_cache(self):
    del self.cache_pool
    torch.npu.empty_cache()
    gc.collect()
```

5.3.1.11 update_cache_after_switch_pd_role 接口

接口功能

更新NPU上的KV Cache。

接口实现

```
def update_cache_after_switch_pd_role(self):
    self.cache_pool.allocate_npu_cache()
```

5.4 Infer Engine API 参考（C++）

5.4.1 说明

须知

- Engine提供的C++接口自本版本起停止演进，且在2024年12月30日后退出并删除。
- 该C++接口不支持多机分布式推理，仅支持单机推理功能。

5.4.2 结构体和枚举说明

5.4.2.1 SamplingParams 结构体

结构体功能

推理采样相关的参数。

结构体格式

```
struct SamplingParams {
    float temperature = 1.0;
    uint32_t topK = 0;
    float topP = 1.0;
    float typicalP = 1.0;
    bool doSample = false;
    uint_64 seed = 1;
    float repetitionPenalty = 1.0;
    bool watermark = false;
    float frequencyPenalty = 0.0f;
    float presencePenalty = 0.0f;
};
```

结构体参数

参数	参数类型	说明	取值要求
temperature	float	控制生成的随机性，较高的值会产生更多样化的输出。	控制生成的随机性，较高的值会产生更多样化的输出。 取值范围大于0，默认值为1.0。 取值越大，结果的随机性越大。推荐使用大于或等于0.001的值，小于0.001可能会导致文本质量不佳。建议最大值取2.0，同时视模型而定。

参数	参数类型	说明	取值要求
topK	uint32_t	控制模型生成过程中考虑的词汇（token）范围，只从概率最高的k个候选词中选择。使用Atlas 300I Duo卡的推理服务场景，推理请求的后参数不支持topK。如果并发发送推理，并且部分推理请求的后处理参数设置了topK，部分请求不设置topK，会造成推理服务异常，导致后续推理请求执行失败，需要重启推理服务。	uint32_t类型，取值范围[0, 2147483647]&&[0, vocabSize)，默认值0。 vocabSize是从modelWeightPath路径下的config.json文件中读取的vocab_size或者padded_vocab_size的值，若不存在则vocabSize取默认值0。建议用户在config.json文件中添加vocab_size或者padded_vocab_size参数，否则可能导致推理失败。
topP	float	控制模型生成过程中考虑的词汇范围，使用累计概率选择候选词，直到累计概率超过给定的阈值。该参数也可以控制生成结果的多样性，它基于累积概率选择候选词，直到累计概率超过给定的阈值为止。	• (0, 1)：兼容TGI 0.9.4版本接口和自研接口对应取值范围。 • (0, 1]：兼容vLLM 0.2.6版本接口、兼容OpenAI接口和兼容Triton接口对应取值范围。
typicalP	float	每个单词被选择为下一个单词的概率大小。	float类型，取值范围(0.0, 1.0]。
doSample	bool	是否做sampling。	-
seed	uint64_t	用于指定推理过程的随机种子，相同的seed值可以确保推理结果的可重现性，不同的seed值会提升推理结果的随机性。	uint64_t类型，取值范围(0, 18446744073709551615]，不传递该参数，系统会产生一个随机seed值。
repetitionPenalty	float	重复惩罚用于减少在文本生成过程中出现重复片段的概率。它对之前已经生成的文本进行惩罚，使得模型更倾向于选择新的、不重复的内容。	float类型，大于0.0，默认值1.0。 • 1.0表示不进行重复度惩罚。 • 大于1.0表示对重复进行惩罚。 建议最大值取2，同时视模型而定。

参数	参数类型	说明	取值要求
watermark	bool	水印。	-
frequencyPenalty	float	它影响模型如何根据文本中词汇的现有频率惩罚新词汇。正值将通过惩罚已经频繁使用的词来降低模型一行中重复用词的可能性。	float类型，取值范围[-2.0, 2.0]，默认值0.0。
presencePenalty	float	它影响模型如何根据到目前为止是否出现在文本中来惩罚新token。正值将通过惩罚已经使用的词，增加模型谈论新主题的可能性。	float类型，取值范围[-2.0, 2.0]，默认值0.0。

5.4.2.2 Compare 结构体

结构体功能

比较请求体对象的序列号大小，根据第一个参数请求体数据类型进行区分。若请求体序列号类似为string类型，则比较请求体对象序列号的hash值大小；若请求体序列号对象为数字类型，则比较其数学大小。最后返回两个请求体的数学大小关系，若结果为true，第一个参数的请求对象序列号值大；若为false，第一个参数的请求体对象序列号值小于等于第二个参数请求体对象序列号值。

结构体格式

```
struct Compare {
    bool operator () (const RequestId &lhs, const RequestId &rhs) const
    {
        if (lhs.Type() == RequestId::DataType::STRING) {
            return std::hash<std::string>()(lhs.StringValue()) < std::hash<std::string>()(rhs.StringValue());
        } else {
            return lhs.UnsignedIntValue() < rhs.UnsignedIntValue();
        }
    }
};
```

结构体参数

无。

5.4.2.3 DataType 枚举类

枚举类功能

标识请求体对象序列号数据类型的枚举类。

枚举类格式

```
enum class DataType {
    UINT64,
    STRING
};
```


枚举值

枚举名	说明
UINT64	请求体对象序列号数据类型为uint64。
STRING	请求体对象序列号数据类型为string。

5.4.2.4 LLMENGINE_memorytype_enum 枚举

枚举功能

内存类型枚举。

枚举格式

```
typedef enum LLMENGINE_memorytype_enum {
    LLMENGINE_MEMORY_CPU,
    LLMENGINE_MEMORY_CPU_PINNED,
    LLMENGINE_MEMORY_GPU
} LLM_ENGINE_MemoryType;
```

枚举值

枚举名	说明
LLMENGINE_MEMORY_CPU	表示内存位于CPU上。
LLMENGINE_MEMORY_CPU_PINNED	表示内存位于CPU上并且是固定的（pinned），即不能被交换到磁盘，这种内存通常用于需要快速且一致的访问速度的情况。
LLMENGINE_MEMORY_GPU	表示内存位于GPU上。

5.4.2.5 LLM_ENGINE_DataType_enum 枚举

枚举功能

数据类型枚举。

枚举格式

```
typedef enum LLM_ENGINE_DataType_enum {
    TYPE_INVALID = 0,
    TYPE_BOOL = 1,
    TYPE_UINT8 = 2,
    TYPE_UINT16 = 3,
    TYPE_UINT32 = 4,
    TYPE_UINT64 = 5,
    TYPE_INT8 = 6,
    TYPE_INT16 = 7,
    TYPE_INT32 = 8,
```



```
    TYPE_INT64 = 9,
    TYPE_FP16 = 10,
    TYPE_FP32 = 11,
    TYPE_FP64 = 12,
    TYPE_STRING = 13,
    TYPE_BF16 = 14,
    TYPE_BUTT,
} LLM_ENGINE_DataType;
```

枚举值

枚举名	说明
TYPE_INVALID	非法类型。
TYPE_BOOL	bool类型。
TYPE_UINT8	uint8类型。
TYPE_UINT16	uint16类型。
TYPE_UINT32	uint32类型。
TYPE_UINT64	uint64类型。
TYPE_INT8	int8类型。
TYPE_INT16	int16类型。
TYPE_INT32	int32类型。
TYPE_INT64	int64类型。
TYPE_FP16	float16类型。
TYPE_FP32	float32类型。
TYPE_FP64	float64类型。
TYPE_STRING	string类型。
TYPE_BF16	bf16类型。
TYPE_BUTT	保留值。

5.4.2.6 LLM_ENGINE_parametertype_enum 枚举

枚举功能

参数类型枚举。

枚举格式

```
typedef enum LLM_ENGINE_parametertype_enum {
    LLM_ENGINE_PARAMETER_STRING,
    LLM_ENGINE_PARAMETER_INT,
    LLM_ENGINE_PARAMETER_BOOL,
    LLM_ENGINE_PARAMETER_BYTES
} LLM_ENGINE_ParameterType;
```


枚举值

枚举名	说明
LLM_ENGINE_PARAMETER_STRING	字符串类型。
LLM_ENGINE_PARAMETER_INT	整数类型。
LLM_ENGINE_PARAMETER_BOOL	布尔类型。
LLM_ENGINE_PARAMETER_BYTES	字节数组类型。

5.4.2.7 InferResponseEndFlag 枚举

枚举功能

响应中EndFlag枚举。

枚举格式

```
enum class InferResponseEndFlag{
    INFER_RESPONSE_CONTINUE = 0,
    INFER_RESPONSE_EOS = 1,
    INFER_RESPONSE_CANCEL = 2,
    INFER_RESPONSE_EXEC_ERROR = 3,
    INFER_RESPONSE_ILLEGAL_INPUT = 4,
    INFER_RESPONSE_REACH_MAX_SEQ_LEN = 5,
    INFER_RESPONSE_REACH_MAX_OUTPUT_LEN = 6,
};
```

枚举值

枚举名	说明
INFER_RESPONSE_CONTINUE	请求继续迭代执行。
INFER_RESPONSE_EOS	请求正常结束。
INFER_RESPONSE_CANCEL	请求被主动CANCEL或STOP，用户不感知，丢弃响应。
INFER_RESPONSE_EXEC_ERROR	请求执行中出错，响应输出为空，err_msg非空。
INFER_RESPONSE_ILLEGAL_INPUT	请求输入校验异常，响应输出为空，err_msg非空。
INFER_RESPONSE_REACH_MAX_SEQ_LEN	请求因达到最大序列长度而结束，响应为最后一轮迭代输出。
INFER_RESPONSE_REACH_MAX_OUTPUT_LEN	请求因达到最大输出长度（包括请求和模型粒度）而结束，响应为最后一轮迭代输出。

5.4.2.8 Code 枚举类

枚举类功能

错误码类型枚举。

枚举类格式

```
enum class Code {
    OK,
    ERROR,
    INVALID_ARG,
    NOT_FOUND,
};
```

枚举值

枚举名	说明
OK	操作成功。
ERROR	操作失败。
INVALID_ARG	非法参数。
NOT_FOUND	无法找到指定操作。

5.4.2.9 Operation 枚举类

枚举类功能

控制操作类型枚举。

枚举类格式

```
enum class Operation {
    STOP = 1,
};
```

枚举值

枚举名	说明
STOP	停止。

5.4.2.10 NodeHealthStatus 枚举类

枚举类功能

节点的链接状态类枚举。

枚举类格式

```
enum class NodeHealthStatus {
    READY,
    ABNORMAL
};
```

枚举值

枚举名	说明
READY	正常。
ABNORMAL	异常。

5.4.3 类参考

5.4.3.1 InferenceEngine

5.4.3.1.1 InferenceEngine 接口

接口功能

创建InferenceEngine对象。

接口格式

```
InferenceEngine();
```

接口参数

无。

使用样例

参考**使用样例**。

返回值

InferenceEngine对象。

5.4.3.1.2 Init 接口

接口功能

推理引擎Engine的初始化函数，用户在声明engine后，需调用Init函数对InferenceEngine初始化。

说明

该函数需与Finalize()成对使用。

接口格式

```
Status Init(const SendResponseCallback &callback = nullptr, const std::string &configPath = "");
```

接口参数

参数	是否必选	说明	取值要求
callback	必选	推理回调函数。	合法的回调函数。
configPath	必选	配置文件地址。	合法地址字符串。

使用样例

```
const std::string dataset = "token_input_gsm.csv";
IOManager manager(dataset); // IOManger 输入输出读写（请按需求完善）

int requestNum = 0;
volatile int completeNum = false;

// 创建engine实例
auto engineConfig = GetEngineConfig();
engineConfig.response_callback = [&manager, &completeNum](std::shared_ptr<InferenceResponse>
&response) {
    InferenceResponse::Output *output;
    response->ImmutableOutput("OUTPUT_IDS", &output);
    manager.SetOutputData(response->GetRequestId().StringValue());

    if (response->IsEOS()) {
        completeNum++;
    }
};

InferenceEngine engine(engineConfig);
engine.Init(GetModelConfig(), GetLoaderConfig()); // 初始化engine
```

返回值

初始化状态。

- Status(Error::Code::OK)：操作成功。
- Status(Error::Code::ERROR)：操作失败。
- Status(Error::Code::INVALID_ARG)：非法参数。
- Status(Error::Code::NOT_FOUND)：无法找到指定操作。

5.4.3.1.3 GetRequestBlockQuotas 接口

接口功能

获取remainBlocks、remainPrefillSlots和remainPrefillTokens的值。

接口格式

```
Status GetRequestBlockQuotas(uint64_t *remainBlocks, uint64_t *remainPrefillSlots, uint64_t
*remainPrefillTokens);
```


接口参数

参数	是否必选	说明	取值要求
remainBlocks	必选	存储获取到的remainBlocks的值。	初始化该变量即可。
remainPrefillSlots	必选	存储获取到的remainPrefillSlots的值。	初始化该变量即可。
remainPrefill	必选	存储获取到的remainPrefill的值。	初始化该变量即可。

使用样例

```
uint64_t remainBlocks;
uint64_t remainPrefillSlots;
uint64_t remainPrefillTokens;
engine.GetRequestBlockQuotas(&remainBlocks, &remainPrefillSlots, &remainPrefillTokens);
```

返回值

返回获取remainBlocks、remainPrefillSlots和remainPrefill的结果。

- Status(Error::Code::OK)：操作成功。
- Status(Error::Code::ERROR)：操作失败。
- Status(Error::Code::INVALID_ARG)：非法参数。
- Status(Error::Code::NOT_FOUND)：无法找到指定操作。

5.4.3.1.4 Forward 接口

接口功能

生成全量token，每生成一个token调用一次回调函数。回调函数通过EngineConfig的response_callback设置。

接口格式

```
Status Forward(std::shared_ptr<InferenceRequest> &request, bool validRequest = false);
```

接口参数

参数	是否必选	说明	取值要求
request	必选	推理请求。	确保有效请求格式。
validRequest	必选	是否校验inputid。	• true • false

使用样例

```
for (size_t i = 0; i < requests.size(); ++i) {
    engine.Forward(requests[i]);
}
```

返回值

返回异步推理的结果。

- Status(Error::Code::OK)：操作成功。
- Status(Error::Code::ERROR)：操作失败。
- Status(Error::Code::INVALID_ARG)：非法参数。
- Status(Error::Code::NOT_FOUND)：无法找到指定操作。

5.4.3.1.5 GetProcessingRequest 接口

接口功能

获取当前正在处理的请求数量。

接口格式

```
Status GetProcessingRequest(uint64_t *num);
```

接口参数

参数	是否必选	说明	取值要求
num	必选	存储获取到的当前正在处理的请求数量。	初始化该变量即可。

使用样例

```
uint64_t processingNum;
engine.GetProcessingRequest(&processingNum);
```

返回值

返回获取num的结果。

- Status(Error::Code::OK)：操作成功。
- Status(Error::Code::ERROR)：操作失败。
- Status(Error::Code::INVALID_ARG)：非法参数。
- Status(Error::Code::NOT_FOUND)：无法找到指定操作。

5.4.3.1.6 ControlRequest 接口

接口功能

对已经发送的异步请求进行控制处理，当前仅支持暂停。

接口格式

```
Status ControlRequest(const RequestId &requestId, Operation operation);
```

接口参数

参数	是否必选	说明	取值要求
requestId	必选	需要处理/干预的请求id。	从request中取出其id。
opertion	必选	需要进行的处理操作类型。	STOP=1。

使用样例

```
for (size_t i = 0; i < requests.size(); ++i) {
    engine.ControlRequest(requests[i]->GetRequestId(), Operation::STOP);
}
```

返回值

对需要处理的请求进行指定操作的处理结果。

- Status(Error::Code::OK)：操作成功。
- Status(Error::Code::ERROR)：操作失败。
- Status(Error::Code::INVALID_ARG)：非法参数。
- Status(Error::Code::NOT_FOUND)：无法找到指定操作。

5.4.3.1.7 Finalize 接口

接口功能

去初始化。

接口格式

```
Status Finalize();
```

接口参数

无。

使用样例

```
engine.Finalize();
```

返回值

去初始化结果。

- Status(Error::Code::OK)：操作成功。
- Status(Error::Code::ERROR)：操作失败。

- Status(Error::Code::INVALID_ARG)：非法参数。
- Status(Error::Code::NOT_FOUND)：无法找到指定操作。

5.4.3.1.8 ~InferenceEngine 接口

接口功能

默认析构函数。

接口格式

```
~InferenceEngine();
```

接口参数

无。

返回值

无。

5.4.3.1.9 GetMaxBatchSize 接口

接口功能

获取max batch size的值。

接口格式

```
Status GetMaxBatchSize(uint64_t *batchSize);
```

接口参数

参数	是否必选	说明	取值要求
batchSize	必选	指向uint64_t类型变量的指针batchSize。	合法的指针。

使用样例

```
uint64_t batchSize;
uint64_t *pBatchSize = &batchSize;
engine.GetMaxBatchSize(pBatchSize);
```

返回值

- Status(Error::Code::OK)：操作成功。
- Status(Error::Code::ERROR)：操作失败。
- Status(Error::Code::INVALID_ARG)：非法参数。
- Status(Error::Code::NOT_FOUND)：无法找到指定操作。

5.4.3.2 InferenceRequest

5.4.3.2.1 InferenceRequest 接口

接口功能

创建InferenceRequest对象。

接口格式

```
explicit InferenceRequest(const RequestId &reqId);
InferenceRequest() = delete;
```

接口参数

参数	是否必选	说明	取值要求
reqId	必选	推理请求对象，请求之间的唯一标识。	合法的请求对象，请参考5.4.3.3 RequestId。

使用样例

```
int reqId = 0;
InferenceRequest *request(RequestID(reqId));
InferenceRequest::Input *input;
data_size = 10;
std::vector<int64_t> shape({1, data_size});
request->AddOriginalInput("INPUT_IDS", TYPE_INT64, &shape[0], shape.size(), &input);
```

返回值

InferenceRequest对象。

5.4.3.2.2 AddOriginalInput 接口

接口功能

- 向推理请求中添加原始输入，默认使用INPUT_IDS作为输入向量名称。
- 向推理请求中添加指定名称的原始输入张量。

接口格式

```
Status AddOriginalInput(LLM_ENGINE_DataType datatype, const int64_t *shape, uint64_t dimCount, Input **input);
Status AddOriginalInput(const std::string &name, LLM_ENGINE_DataType datatype, const int64_t *shape, uint64_t dimCount, Input **input);
```


接口参数

参数	是否必选	说明	取值要求
datatype	必选	输入张量中保存的数据类型。	合法的数据类型，请参考5.4.2.5 LLM_ENGINE_DataType_enum枚举。
shape	必选	输入张量的维度。	非空指针。取值建议为{1, size}，size为输入张量的长度。
dimCount	必选	输入张量的dim。	uint64_类型，小于或等于10000。取值建议为2，与shape长度一致。
input	必选	指针的指针，用于在函数外访问到输入张量对象。	合法的指针的指针。

参数	是否必选	说明	取值要求
name	必选	输入张量的名称。	由大写字母和下划线组成，且不以下划线作为开头和结尾。
datatype	必选	输入张量中保存的数据类型。	合法的数据类型，请参考5.4.2.5 LLM_ENGINE_DataType_enum枚举。
shape	必选	输入张量的维度。	非空指针。取值建议为{1, size}，size为输入张量的长度。
dimCount	必选	输入张量的dim。	uint64_类型，小于或等于10000。取值建议为2，与shape长度一致。
input	必选	指针的指针，用于在函数外访问到输入张量对象。	合法的指针的指针。

使用样例

```
int reqId = 0;
request = std::make_shared<InferenceRequest>(RequestID(reqId));
InferenceRequest::Input *input;
data_size = 10;
std::vector<int64_t> shape({1, data_size});
request->AddOriginalInput(TYPE_INT64, &shape[0], shape.size(), &input);
```


返回值

- Status(Error::Code::OK)：操作成功。
- Status(Error::Code::ERROR)：操作失败。
- Status(Error::Code::INVALID_ARG)：非法参数。
- Status(Error::Code::NOT_FOUND)：无法找到指定操作。

5.4.3.2.3 RemoveOriginalInput 接口

接口功能

从推理请求中删除指定名称的原始输入张量对象。

接口格式

```
Status RemoveOriginalInput(const std::string &name);
```

接口参数

参数	是否必选	说明	取值要求
name	必选	输入张量名称。	合法的数据名称。

使用样例

```
std::string removeName = "INPUT_IDS";
request->RemoveOriginalInput(removeName);
```

返回值

- Status(Error::Code::OK)：操作成功。
- Status(Error::Code::ERROR)：操作失败。
- Status(Error::Code::INVALID_ARG)：非法参数。
- Status(Error::Code::NOT_FOUND)：无法找到指定操作。

5.4.3.2.4 MaxOutputLen 接口

接口功能

获取推理请求返回token的最大值。

接口格式

```
uint32_t MaxOutputLen() const;
```

接口参数

无。

使用样例

```
uint32_t maxOutLen = request->MaxOutputLen();
```


返回值

推理请求返回token的最大值，uint32_t类型。

5.4.3.2.5 SetMaxOutputLen 接口

接口功能

设置推理请求返回token的最大值。

接口格式

```
bool SetMaxOutputLen(uint32_t maxOutputLen);
```

接口参数

参数	是否必选	说明	取值要求
maxOutputLen	必选	推理请求返回token的最大值。	合法的推理请求返回token最大值。

使用样例

```
uint32_t maxOutLen = 512;
request->SetMaxOutputLen(maxOutLen);
```

返回值

- true：设置成功。
- false：设置失败。

5.4.3.2.6 GetRequestId

接口功能

获取推理请求唯一标识，请求体对象。

接口格式

```
RequestId &GetRequestId();
```

接口参数

无。

使用样例

```
RequestId reqId = request->GetRequestId();
```

返回值

返回推理请求体对象。

5.4.3.2.7 GetSamplingParams 接口

接口功能

获取后处理参数。

接口格式

```
const SamplingParams &GetSamplingParams() const;
```

接口参数

无。

使用样例

```
SamplingParams params = request->GetSamplingParams();
```

返回值

后处理参数。

5.4.3.2.8 SetSamplingParams 接口

接口功能

设置后处理参数。

接口格式

```
void SetSamplingParams(SamplingParams samplingParams);
```

接口参数

参数	是否必选	说明	取值要求
samplingParams	必选	后处理参数结构体。	合法的后处理对象。

使用样例

```
SamplingParams param;
param.temperature = 1.0;
param.topK = 1;
request->SetSamplingParams(param);
```

返回值

无。

5.4.3.2.9 SetSendResponseCallback 接口

接口功能

设置推理响应回调函数。

接口格式

```
void SetSendResponseCallback(const SendResponseCallback &callback);
```

接口参数

参数	是否必选	说明	取值要求
callback	必选	推理请求响应回调函数。	非空指针。

使用样例

```
SendResponseCallback callback = [](std::share_ptr<InferenceResponse> &response) {
    if (response->IsEOS) {
    }
}
request->SetSendResponseCallback(callback);
```

返回值

无。

5.4.3.2.10 HasSampling 接口

接口功能

获取是否存在后处理参数标志。

接口格式

```
bool HasSampling();
```

接口参数

无。

使用样例

```
bool sampling = request->HasSampling();
```

返回值

- true：存在后处理参数。
- false：不存在后处理参数。

5.4.3.2.11 SetInputText 接口

接口功能

设置输入推理文本。

接口格式

```
void SetInputText(std::string &text);
```

接口参数

参数	是否必选	说明	取值要求
text	必选	推理文本。	合法字符串string类型。

使用样例

```
std::string text = "123";
request->SetInputText(text);
```

返回值

无。

5.4.3.2.12 GetInputText 接口

接口功能

获取输入推理文本。

接口格式

```
std::string &GetInputText();
```

接口参数

无。

使用样例

```
std::string text = request->GetInputText();
```

返回值

推理文本字符串。

5.4.3.2.13 IsInputText 接口

接口功能

判断是否是文本推理。

接口格式

```
bool IsInputText();
```

接口参数

无。

使用样例

```
bool isText = request->IsInputText();
```

返回值

- true：属于文本推理。
- false：不属于文本推理。

5.4.3.2.14 GetRequestInner 接口

接口功能

获取InferenceRequestInner对象。

接口格式

```
std::shared_ptr<InferenceRequestInner> GetRequestInner() const;
```

接口参数

无。

使用样例

```
auto reqInner = request->GetRequestInner();
```

返回值

InferenceRequestInner对象。

5.4.3.2.15 Input 类

Input 接口

接口功能

根据输入张量名称，张量内的数据类型，张量维度，张量dim创建推理请求输入张量对象。

接口格式

```
Input(const std::string &name, LLM_ENGINE_DataType datatype, const int64_t *shape, uint64_t dimCount);
Input() = delete;
```


接口参数

表 5-5 接口参数

参数	是否必选	说明	取值要求
name	必选	输入张量名称。	合法字符串string类型，非空。
datatype	必选	张量中保存的数据类型。	LLM_ENGINE_DataType_enum类型，请参见表5-6。
shape	必选	张量的维度。	合法的维度信息，取值建议为{1, size}，size为输入张量的长度。
dimCount	必选	张量的dim。	uint64_类型，小于或等于10000。取值建议为2，与shape长度一致。

表 5-6 LLM_ENGINE_DataType_enum 类型说明

LLM_ENGINE_DataType_enum类型	值	意义
TYPE_INVALID	0	非法类型
TYPE_BOOL	1	bool类型
TYPE_UINT8	2	uint8类型
TYPE_UINT16	3	uint16类型
TYPE_UINT32	4	uint32类型
TYPE_UINT64	5	uint64类型
TYPE_INT8	6	int8类型
TYPE_INT16	7	int16类型
TYPE_INT32	8	int32类型
TYPE_INT64	9	int64类型
TYPE_FP16	10	float16类型
TYPE_FP32	11	float32类型
TYPE_FP64	12	float64类型
TYPE_STRING	13	string类型
TYPE_BF16	14	bf16类型
TYPE_BUTT	15	保留值

使用样例

```
Input* input(name, datatype, shape, dimCount);
```

返回值

返回推理请求输入张量对象。

~Input 接口

接口功能

析构函数。

接口格式

```
~Input() = default;
```

接口参数

无。

使用样例

无。

返回值

无。

Allocate 接口

接口功能

为输入张量对象分配内存空间，用于接收输入数据。

接口格式

```
Status Allocate(size_t size);
```

接口参数

参数	是否必选	说明	取值要求
size	必选	申请的内存空间大小。	合法的内存空间大小。

使用样例

```
const int64_t paramNum = 10;
Error res = input->Allocate(paramNum * sizeof(float));
```


返回值

- Status(Error::Code::OK)：操作成功。
- Status(Error::Code::ERROR)：操作失败。
- Status(Error::Code::INVALID_ARG)：非法参数。
- Status(Error::Code::NOT_FOUND)：无法找到指定操作。

Name 接口

接口功能

返回输入张量名称。

接口格式

```
const std::string &Name() const;
```

接口参数

无。

使用样例

```
std::string name = input->Name();
```

返回值

string类型的输入张量名称。

DType 接口

接口功能

获取推理输入张量保存的数据类型。

接口格式

```
LLM_ENGINE_DataType DType() const;
```

接口参数

无。

使用样例

```
LLM_ENGINE_DataType dataType = input->DType();
```

返回值

LLM_ENGINE_DataType_enum枚举。

Shape 接口

接口功能

获得推理输入张量的维度信息。

接口格式

```
const std::vector<int64_t> &Shape() const;
```

接口参数

无。

使用样例

```
std::vector<int64_t> shape = input->Shape();
```

返回值

输入张量的维度信息。

DimCount 接口

接口功能

获取输入张量dim大小。

接口格式

```
const uint64_t &DimCount() const;
```

接口参数

无。

使用样例

```
uint64_t dim = input->DimCount();
```

返回值

输入张量dim大小。

Buffer 接口

接口功能

获取输入张量指向输入数据的指针。

接口格式

```
void *Buffer();
```


接口参数

无。

使用样例

```
auto *f32Buffer = (float *)input->Buffer();
```

返回值

输入张量指向输入数据的指针。

ByteSize 接口

接口功能

获取输入张量的输入数据的总大小。

接口格式

```
uint64_t ByteSize() const;
```

接口参数

无。

使用样例

```
uint64_t byteSize = input->ByteSize();
```

返回值

输入张量的输入数据总大小。

GetInputInner 接口

接口功能

获得InputInner对象。

接口格式

```
std::shared_ptr<InputInner> GetInputInner() const;
```

接口参数

无。

使用样例

```
auto inputInner = input->GetInputInner();
```

返回值

InputInner对象。

SetRelease 接口

接口功能

设置是否需要释放数据内存的标识。

接口格式

```
Status SetRelease(bool status);
```

接口参数

参数	是否必选	说明	取值要求
status	必选	是否需要释放数据内存的标识。	合法的bool类型。

使用样例

```
bool setFlag = False;
Status status = input->SetRelease(setFlag);
```

返回值

- Status(Error::Code::OK)：操作成功。
- Status(Error::Code::ERROR)：操作失败。
- Status(Error::Code::INVALID_ARG)：非法参数。
- Status(Error::Code::NOT_FOUND)：无法找到指定操作。

5.4.3.3 RequestId

5.4.3.3.1 RequestId 接口

接口功能

- 根据序列字符串requestLabel值创建以string类型值作为唯一序列号的请求体对象。
- 根据序列uint64类型requestIndex值创建以uint64类型值作为唯一序列号的请求体对象。

接口格式

```
explicit RequestId(const std::string &requestLabel);
explicit RequestId(uint64_t requestIndex);
```


接口参数

参数	是否必选	说明	取值要求
requestLabel	必选	推理请求对象的唯一性标识，序列号，这里是string类型的。	合法的字符类型string类型请求序列号。

参数	是否必选	说明	取值要求
requestIndex	必选	推理请求对象的唯一序列号，uint64类型的。	合法的uint64唯一序列号。

使用样例

```
std::string requestLabel = "123";
uint64_t requestIndex = 456;
RequestId requestId1(requestLabel);
RequestId requestId2(requestIndex);
```

返回值

RequestId对象。

5.4.3.3.2 RequestId 操作符重载接口

接口功能

- 重载推理请求体对象的=号运算符，用于给请求体初始化为以uint64_t类型作为序列号类型，序列号值是rhs的推理请求体对象，用于标识一个请求体。
- 重载推理请求体对象的=号运算符，用于给请求体初始化为以string类型作为序列号类型，序列号值是字符rhs的推理请求体对象，用于标识一个请求体。
- 重载推理请求体对象的=号运算符，用于从已有的请求体对象初始化一个新的请求体对象。

接口格式

```
RequestId &operator = (const uint64_t rhs);
RequestId &operator = (const std::string &rhs);
RequestId &operator = (const RequestId &rhs);
```

接口参数

参数	是否必选	说明	取值要求
rhs	必选	请求体序列号，唯一标识请求体。	合法的uint64_t数值。

参数	是否必选	说明	取值要求
rhs	必选	请求体序列号，唯一标识请求体。	合法的字符串。

参数	是否必选	说明	取值要求
rhs	必选	推理请求体对象。	合法的请求体对象。

使用样例

```
std::string requestLabel = "123";
uint64_t requestIndex = 456;
RequestId requestId1 = requestLabel;
RequestId requestId2 = requestIndex;
RequestId requestId3 = requestId1;
```

返回值

RequestId对象。

5.4.3.3.3 RequestId 接口

接口功能

使用已有的requestId初始化RequestId对象。

接口格式

```
RequestId(const SimpleLLMInference::RequestId &other);
```

接口参数

参数	是否必选	说明	取值要求
other	必选	已有的RequestId对象。	合法的请求体对象。

使用样例

```
std::string requestLabel = "123";
RequestId requestId1 = requestLabel;

RequestId requestId2(requestId1);
```

返回值

RequestId对象。

5.4.3.3.4 Type 接口

接口功能

获取请求体对象唯一的序列号标识的数据类型。

接口格式

```
DataType Type() const;
```

接口参数

无。

使用样例

```
std::string requestLabel = "123";
RequestId requestId1 = requestLabel;
DataType type = requestId1->Type();
```

返回值

请求体对象唯一的序列号标识的数据类型。

5.4.3.3.5 StringValue 接口

接口功能

获取请求体字符类型的序列号，多为请求体序列号类似为string类型请求体使用。

接口格式

```
const std::string &StringValue() const;
```

接口参数

无。

使用样例

```
std::string stringVal = requestId->StringValue();
```

返回值

返回请求体对象字符类型序列号，若请求体对象为uint64_t类型，会返回空序列号。

5.4.3.3.6 UnsignedIntValue 接口

接口功能

获取请求体数字类型的序列号，多为请求体序列号类似为uint64_t类型请求体使用。

接口格式

```
uint64_t UnsignedIntValue() const;
```


接口参数

无。

使用样例

```
uint64_t value = requestId->UnsignedIntValue();
```

返回值

返回请求体对象数字类型序列号，若请求体对象为string类型，会返回空默认值0。

5.4.3.4 InferenceResponse

5.4.3.4.1 ~InferenceResponse 接口

接口功能

析构函数。

接口格式

```
virtual ~InferenceResponse() = default;
```

接口参数

无。

使用样例

无。

返回值

无。

5.4.3.4.2 IsEOS 接口

接口功能

判断推理使用的tokennizer是否需要使用eos。

接口格式

```
virtual bool IsEOS() const noexcept = 0;
```

接口参数

无。

使用样例

```
bool eos = response->IsEOS();
```


返回值

tokennizer是否需要使用eos判断结果。

5.4.3.4.3 GetFlags 接口

接口功能

获取推理请求当前状态标志。

接口格式

```
virtual uint32_t GetFlags() const noexcept = 0;
```

接口参数

无。

使用样例

```
uint32_t flag = response->GetFlags();
```

返回值

返回推理请求当前所处状态的标志。

5.4.3.4.4 ImmutableOutput 接口

接口功能

查找张量名称为name的响应张量对象，并通过指针的指针进行函数外引用。

接口格式

```
virtual Status ImmutableOutput(const std::string &name, Output **output) const noexcept = 0;
```

接口参数

参数	是否必选	说明	取值要求
name	必选	输出张量的名称。	合法的输出张量名称。
output	必选	指针的指针，保证可以在函数外引用指定输出张量对象。	合法的指针的指针。

使用样例

```
std::string name = "OUTPUT_IDS";
InferenceResponse::Output *output;
Status status = response->ImmutableOutput(name, output);
```


返回值

- Status(Error::Code::OK)：操作成功。
- Status(Error::Code::ERROR)：操作失败。
- Status(Error::Code::INVALID_ARG)：非法参数。
- Status(Error::Code::NOT_FOUND)：无法找到指定操作。

5.4.3.4.5 GetRequestId 接口

接口功能

获取当前响应请求对象对应的推理请求体对象。

接口格式

```
virtual const RequestId &GetRequestId() const noexcept = 0;
```

接口参数

无。

使用样例

```
RequestId requestId = response->GetRequestId();
```

返回值

当前响应请求对象对应的推理请求体对象。

5.4.3.4.6 Output 类

~Output 接口

接口功能

output的析构函数，用于对象销毁。

接口格式

```
virtual ~Output() = default;
```

接口参数

无。

使用样例

无。

返回值

无。

Name 接口

接口功能

获得输出张量的名称。

接口格式

```
virtual const char *Name() const noexcept = 0;
```

接口参数

无。

使用样例

```
char *name = output->Name();
```

返回值

输出张量的名称。

DType 接口

接口功能

获取输出张量的数据类型。

接口格式

```
virtual LLM_ENGINE_DataType DType() const noexcept = 0;
```

接口参数

无。

使用样例

```
LLM_ENGINE_DataType type = output->DType();
```

返回值

输出张量的数据类型。

Shape 接口

接口功能

获取输出张量的维度值，以数组的形式返回。

接口格式

```
virtual const std::vector<int64_t> &Shape() const noexcept = 0;
```


接口参数

无。

使用样例

```
std::vector<int64_t> shape = output->Shape();
```

返回值

返回输出张量维度数组。

DimCount

接口功能

获取输出张量的dim值。

接口格式

```
virtual uint64_t DimCount() const noexcept = 0;
```

接口参数

无。

使用样例

```
uint64_t dim = output->DimCount();
```

返回值

输出张量的dim值。

Buffer

接口功能

获取输出张量数据地址指针。

接口格式

```
virtual void *Buffer() noexcept = 0;
```

接口参数

无。

使用样例

```
uint64_t dim = output->DimCount();
```

返回值

无。

Buffer 接口

接口功能

获取输出张量数据地址指针。

接口格式

```
virtual const void *Buffer() const noexcept = 0;
```

接口参数

无。

使用样例

```
uint64_t buffer= output->Buffer();
```

返回值

无。

ByteSize 接口

接口功能

获取输出张量对象数据总大小。

接口格式

```
virtual uint64_t ByteSize() const noexcept = 0;
```

接口参数

无。

使用样例

```
uint64_t byteSize= output->ByteSize();
```

返回值

返回输出张量对象数据总大小。

5.4.3.5 Error

5.4.3.5.1 Error 接口

接口功能

- 根据枚举Code创建返回信息对象。
- 根据枚举Code、错误信息msg，创建返回信息对象。

接口格式

```
explicit Error(Code code = Code::OK);
explicit Error(Code code, std::string msg);
```

接口参数

参数	是否必选	说明	取值要求
code	必选	错误码。	• Code::OK • Code::ERROR • Code::INVALID • Code::NOT_FOUND

参数	是否必选	说明	取值要求
code	必选	错误码。	• Code::OK • Code::ERROR • Code::INVALID • Code::NOT_FOUND
msg	必选	错误信息。	合法字符串。

使用样例

```
Error(Error::Code::OK)
Error(Error::Code::ERROR, "ERROR: Failed to get real path of home.");
```

返回值

Error对象。

5.4.3.5.2 ErrorCode 接口

接口功能

获取返回对象内的错误码。

接口格式

```
Code ErrorCode() const;
```

接口参数

无。

使用样例

```
Error(Error::Code::ERROR, "ERROR: Failed to get real path of home.").ErrorCode();
```

返回值

错误码枚举。

5.4.3.5.3 Message 接口

接口功能

获取返回对象内的错误码信息。

接口格式

```
const std::string &Message() const;
```

接口参数

无。

使用样例

```
Error(Error::Code::ERROR, "ERROR: Failed to get real path of home.").Message();
```

返回值

错误码信息，string类型。

5.4.3.5.4 IsOk 接口

接口功能

获取返回对象状态，是否是正常状态。

接口格式

```
bool IsOk() const;
```

接口参数

无。

使用样例

```
auto error = input->Allocate(pData.size() * sizeof(int64_t));
if (!error.IsOk()) {
    EP_LOG_ERROR("Failed to allocate data buffer for id=" << logId);
    return -1;
}
```

返回值

- true：返回正常，代表操作成功。
- false：返回异常，代表操作失败。

5.4.3.5.5 ToString 接口

接口功能

将返回对象转为字符串格式，具体为错误码映射为相应字符后，和错误信息以": "进行拼接。

接口格式

```
std::string ToString() const;
```

接口参数

无。

使用样例

```
auto err = input->Allocate(totalSize * sizeof(int64_t));
err.ToString();
```

返回值

返回对象转化出来的字符串。

Error对象	返回字符格式
Error::Code::OK	OK: + msg
Error::Code::ERROR	Error: +msg
Error::Code::INVALID_ARG	Invalid argument: +msg
Error::Code::NOT_FOUND	Not found: +msg

5.4.3.5.6 CodeToString 接口

接口功能

将错误码转化为对应字符。

接口格式

```
static const char *CodeToString(const Code code);
```

接口参数

参数	是否必选	说明	取值要求
code	必选	错误码枚举。	枚举类Code值。

使用样例

```
std::string str(CodeToString(code_));
```

返回值

Code枚举	返回字符格式
Code::OK	OK
Code::ERROR	Error
Code::INVALID_ARG	Invalid argument
Code::NOT_FOUND	Not found

5.4.3.6 Status

5.4.3.6.1 Status 接口

接口功能

- 通过错误码创建状态对象。
- 通过错误码和string类型的msg创建状态对象。
- 通过Error对象创建状态。

接口格式

```
explicit Status(Error::Code code = Error::Code::OK) noexcept;
explicit Status(Error::Code code, const std::string &msg);
explicit Status(const Error &error);
```

接口参数

参数	是否必选	说明	取值要求
code	必选	错误码。	• Code::OK • Code::ERROR • Code::INVALID • Code::NOT_FOUND

参数	是否必选	说明	取值要求
code	必选	错误码。	• Code::OK • Code::ERROR • Code::INVALID • Code::NOT_FOUND
msg	必选	错误信息.	合法字符串。

参数	是否必选	说明	取值要求
error	必选	错误信息返回对象。	合法错误信息返回对象。

使用样例

```
Status(Error::Code::OK);

Status(Error::Code::ERROR, "Engine init model failed: new modelBackend failed");

Status CommonErrorToStatus(const Error &error)
{
    return Status(error);
}
```

返回值

返回状态对象。

5.4.3.6.2 IsOk 接口

接口功能

判断状态对象状态是否是成功的。

接口格式

```
bool IsOk() const;
```

接口参数

无。

使用样例

```
status = response->ImmutableOutput("OUTPUT_IDS", &outputToken);
if (!status.IsOk()) {
    EP_LOG_ERROR("Failed to get token by " << status.StatusMsg());
    return;
}
```


返回值

- true：返回正常，代表返回状态成功。
- false：返回异常，代表返回状态失败。

5.4.3.6.3 StatusCode 接口

接口功能

获取状态对象的错误码。

接口格式

```
Error::Code StatusCode() const;
```

接口参数

无。

使用样例

```
auto status = inferEngine->ControlRequest(requestId, Operation::STOP);
if (status.StatusCode() == Error::Code::OK) {
    EP_LOG_INFO("stop request id = " << stopReqId << " success.");
    return HttpResult{ HTTP_STATUS_CODE_OK, reqCtx->MsgBody() };
}
```

返回值

错误码枚举。

5.4.3.6.4 StatusMsg 接口

接口功能

获取状态对象的错误信息。

接口格式

```
std::string StatusMsg() const;
```

接口参数

无。

使用样例

```
if (!status.IsOk()) {
    EP_LOG_ERROR("Failed to get eos info by " << status.StatusMsg());
    return;
}
```

返回值

错误信息，string类型。

5.4.3.6.5 operator 操作符重载接口

接口功能

重载状态信息对象==操作符，实现若两个状态对象判断是否相等时，只需比较错误码是否相同即可。

接口格式

```
bool operator==(const Status& other) const;
```

接口参数

参数	是否必选	说明	取值要求
other	必选	与本状态对象比较的另外一个状态对象。	合法的状态对象。

使用样例

无。

返回值

bool类型，两个对象是否相等。

5.4.4 函数

5.4.4.1 SendResponseCallback

函数功能

定义了一个函数对象类型SendResponseCallback，它接受一个参数，类型为std::shared_ptr<InferenceResponse>的智能指针，并且没有返回值。

函数格式

```
using SendResponseCallback = std::function<void(std::shared_ptr<InferenceResponse> &)>;
```

5.4.4.2 ReleaseCallback

函数功能

定义了一个函数对象类型ReleaseCallback，它接受一个参数，类型为const RequestId，并且没有返回值。

函数格式

```
using ReleaseCallback = std::function<void(const RequestId &)>;
```


6 特性介绍

6.1 量化特性介绍

6.1.1 前提条件

已在环境上安装CANN和ATB Models详情请参见《**MindIE安装指南**》。

说明

本次样例参考以下安装路径进行：

安装ATB Models并初始化ATB Models环境变量。模型仓set_env.sh脚本中有初始化“${ATB_SPEED_HOME_PATH}”环境变量的操作，所以source模型仓中set_env.sh脚本时会同时初始化“${ATB_SPEED_HOME_PATH}”环境变量。

ATB Models公共能力支持以下量化方式：

- W8A8
- W8A16
- KV Cache int8
- W8A8SC 稀疏量化

- Anti-Outlier 离群值处理

须知

每个模型具体支持的量化方式不同，请参考${ATB_SPEED_HOME_PATH}/examples/models/路径下模型Readme文件中的特性支持矩阵。

量化特性支持量化回退，即模型中部分权重不做量化，使用原始的浮点权重进行MatMul计算，支持Linear级别的量化回退。量化回退可以提升量化权重的精度。每个模型不同量化方式下的量化回退配置可能都不相同，请参考各模型生成量化权重脚本中的配置。

以LLaMa为例，在${ATB_SPEED_HOME_PATH}/examples/models/llama/generate_quant_weight.sh中定义了不同量化方式下的回退层。在W8A16量化场景下，不设置回退层（默认回退lmhead），其他量化场景下，将layer中的down层全部回退。

```
get_down_proj_disable_name() {
    local num_layer=$1
    local disable_names=""
    for ((i=0; i<$num_layer; i++)); do
        disable_names="$disable_names model.layers.$i.mlp.down_proj"
    done
    echo "$disable_names"
}
```

6.1.2 量化脚本说明

MindIE LLM中提供统一的脚本${ATB_SPEED_HOME_PATH}/examples/convert/model_slim/quantifier.py供用户生成量化权重。

使用说明

由于不同模型量化特性参数配置不同，模型基于公共脚本编写各自的量化脚本。具体使用方式见模型Readme文件（${ATB_SPEED_HOME_PATH}/examples/models/{模型名称}/README.md）。

不同量化方式下的参数配置方法见后续章节。

参数说明

说明

详情可参考《CANN msModelSlim工具指南》的“模型量化 > 简介”章节。

表 6-1 量化脚本参数说明

参数名称	是否为必选	类型	默认值	描述
--model_path	是	string	-	模型权重路径。

参数名称	是否为必选	类型	默认值	描述
--save_directory	是	string	-	量化权重保存路径。
--calib_texts	否	string	None	量化时的校准数据，多条数据间使用空格分割。 **说明** 脚本基于calib_texts进行推理，若用户传入数值过大，会出现由out of memory导致的报错信息。
--calib_file	否	string	${ATB_SPEED_HOME_PATH}/examples/convert/model_slim/teacher_qualification.jsonl	包含校准数据的文件。
--w_bit	否	int	8	权重量化bit。 ● 可配置为4或8。 ● per_group的场景下需配置为4。 ● 稀疏量化场景下，需配置为4。
--a_bit	否	int	8	激活值量化bit。 可选值为8和16。 ● 大模型量化场景下，可配置为8或16。 – per_group的场景下需配置为16。 – is_dynamic参数配置为True，使用per-token动态量化场景下，需配置为8。 ● 大模型稀疏量化场景下，需配置为8。
--disable_names	否	string	None	需排除量化的节点名称，即手动回退的量化层名称。如精度太差，推荐回退量化敏感层，如分类层、输入层、检测head层等。

参数名称	是否为必选	类型	默认值	描述
--device_type	否	string	"cpu"	量化时的硬件类型，仅支持"cpu"或"npu"。
--fraction	否	float	0.01	稀疏量化精度控制。
--act_method	否	int	1	激活值量化方法，仅支持1或2或3。 ● 1代表Label-Free场景的min-max量化方式。 ● 2代表Label-Free场景的histogram量化方式。 ● 3代表Label-Free场景的自动混合量化方式。 ● 开启lowbit稀疏量化功能时不支持选择值3。
--co_sparse	否	bool	False	是否开启稀疏量化功能。大模型稀疏量化场景下，优先使用lowbit稀疏量化功能，开启lowbit稀疏量化后，co_sparse参数自动失效。
--anti_method	否	string	""	离群值抑制算法，默认不开启。 ● "m1": Smooth Quant算法。 ● "m2": Outlier Suppression算法。 ● "m3": AWQ算法。 ● "m4": Smooth Quant优化算法。 ● "m5": CBQ量化算法。
--disable_level	否	string	"L0"	自动回退等级。 ● "L0"：不执行回退 ● "L1"：回退1层 ● "L2"：回退2层 ● "L3"：回退3层 ● "L4"：回退4层 ● "L5"：回退5层
--input_ids_name	否	string	"input_ids"	tokenize后input_ids对应的键名。

参数名称	是否为必选	类型	默认值	描述
--attention_mask_name	否	string	"attention_mask"	tokenize后attention_mask对应的键名。
--do_smooth	否	bool	False	是否开启smooth功能。启用do_smooth功能后，平滑激活值。默认为"False"，不开启smooth功能。
--use_sigma	否	bool	False	是否启动sigma功能。启用use_sigma功能后，可根据正态分布数值特点进行异常值保护。默认为"False"，不开启sigma功能。
--sigma_factor	否	float	3.0	启用sigma功能后sigma_factor的值，用于限制异常值的保护范围。默认为3.0，取值范围为[3.0, 4.0]。
--is_lowbit	否	bool	False	是否开启lowbit量化功能。 默认为"False"，不开启lowbit量化功能。
--mm_tensor	否	bool	True	选择进行per-channel量化或per-tensor量化。 ● True: per-tensor量化。 ● False: per-channel量化。
--w_sym	否	bool	True	权重量化是否为对称量化，默认开启对称量化。
--use_kvcache_quant	否	bool	False	是否使用KV Cache量化功能，默认不开启KV Cache量化功能。
--open_outlier	否	bool	True	是否开启权重异常值划分。 ● True：开启权重异常值划分。 ● False：关闭权重异常值划分。 ● 仅在lowbit设置为True时生效。

参数名称	是否为必选	类型	默认值	描述
--group_size	否	int	64	per_group量化中group的大小。 默认值为64，支持配置为64或128。
--is_dynamic	否	bool	False	是否开启per token量化，当前仅W8A8支持per token量化。 ● True：开启。 ● False：不开启。
--tokenizer_args	否	符合json格式的string	"{}"	对校准数据集做tokenize时可额外配置的参数。 例如：'{"padding_side":"left","pad_token":"!"}'
--use_reduce_quant	否	bool	False	是否使用lccl reduce量化功能，默认不开启。
--tp_size	否	int	1	lccl reduce量化时需要用到的卡数，默认为1。
--is_dynamic	否	bool	False	● True：使用per-token动态量化。 ● False：不使用per-token动态量化。
--tokenizer_args	否	符合json格式的string	"{}"	tokenize扩展参数。
--part_file_size	否	int	None	量化权重保存文件切分大小，单位GB，默认不切分。

说明

calib_texts参数包含校准数据，量化校准时间和传入数据量成正比。当使用NPU进行量化时，输入会被转换成token id搬运至NPU，传入数据量过大可能会导致NPU tensor占用显存过大，而出现由out of memory导致的报错信息。

6.1.3 W8A8

简介

此量化方式对权重和激活值均进行量化，将高位浮点数转为8 bit，减少模型权重的体积。使用int8格式的数据进行计算，可以减少MatMul算子计算量，以提升推理性能。

量化后权重目录结构：

```
├─ config.json
├─ quant_model_weight_w8a8.safetensors
├─ quant_model_description_w8a8.json
├─ tokenizer_config.json
├─ tokenizer.json
└─ tokenizer.model
```

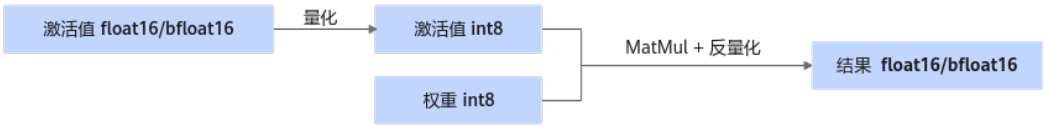
- 量化输出包含：权重文件quant_model_weight_w8a8.safetensors和权重描述文件quant_model_description_w8a8.json。
- 目录中的其余文件为推理时所需的配置文件，不同模型略有差异。

以下展示了量化后权重描述文件quant_model_description_w8a8.json中的部分内容：

```
{
  "model_quant_type": "W8A8",
  "model.embed_tokens.weight": "FLOAT",
  "model.layers.0.self_attn.q_proj.weight": "W8A8",
  "model.layers.0.self_attn.q_proj.input_scale": "W8A8",
  "model.layers.0.self_attn.q_proj.input_offset": "W8A8",
  "model.layers.0.self_attn.q_proj.quant_bias": "W8A8",
  "model.layers.0.self_attn.q_proj.deq_scale": "W8A8",
}
```

量化后的MatMul权重新增input_scale、input_offset、quant_bias和deq_scale。其中input_scale和input_offset用于对激活值进行量化。MatMul使用量化后的激活值和量化权重进行计算。quant_bias和deq_scale用于对MatMul的计算结果进行反量化。

图 6-1 量化权重推理时流程



此量化方式支持量化float16或bfloat16类型的原始权重。

表 6-2 float16 权重量化后 dtype 及 shape 信息（假设原始权重的 shape 为[n, k]）

Tensor信息	weight	input_scale	input_offset	quant_bias	deq_scale
dtype	int8	float16	float16	int32	int64
shape	[n, k]	[1]	[1]	[n]	[n]

表 6-3 bfloat16 权重量化后 dtype 及 shape 信息（假设原始权重的 shape 为[n, k]）

Tensor信息	weight	input_scale	input_offset	quant_bias	deq_scale
dtype	int8	bfloat16	bfloat16	int32	float32
shape	[n, k]	[1]	[1]	[n]	[n]

生成权重

以LLama3.1-8B为例，您可以使用以下指令生成W8A8量化权重。

```
cd ${ATB_SPEED_HOME_PATH}
python examples/models/llama3/convert_quant_weights.py --model_path {浮点权重路径} --save_directory {W8A8量化权重路径} --w_bit 8 --a_bit 8 --disable_level L0 --device_type cpu --anti_method m1 --act_method 1 --calib_file ${llm_path}/examples/convert/model_slim/boolq.jsonl
```

- 以上指令展示了生成LLama3.1-8B W8A8权重的最优参数配置，不同模型的参数配置不同，请参考模型Readme文件。
- W8A8量化权重的config.json中应包含quantize字段，其值为"w8a8"。

执行推理

以LLama3.1-8B为例，您可以使用以下指令执行对话测试，推理内容为"What's deep learning"。

```
cd ${ATB_SPEED_HOME_PATH}
bash examples/models/llama3/run_pa.sh {W8A8量化权重路径} 20
```

6.1.4 W8A16

简介

此量化方式对激活值不做量化，仅将权重量化为8 bit。使用per Channel量化。

说明

仅Atlas A2 推理系列产品支持此量化方式。

量化后权重目录结构：

```
├── config.json
├── quant_model_weight_w8a16.safetensors
├── quant_model_description_w8a16.json
├── tokenizer_config.json
├── tokenizer.json
└── tokenizer.model
```

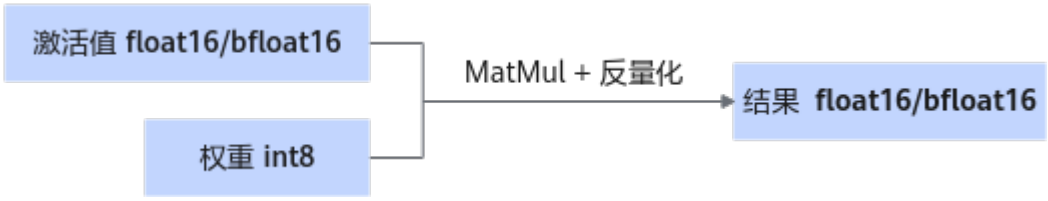
- 量化输出包含：权重文件quant_model_weight_w8a16.safetensors和权重描述文件quant_model_description_w8a16.json。
- 目录中的其余文件为推理时所需的配置文件，不同模型略有差异。

以下展示了量化后权重描述文件quant_model_description_w8a16.json中的部分内容：

```
{
  "model_quant_type": "W8A16",
  "model.embed_tokens.weight": "FLOAT",
  "model.layers.0.self_attn.q_proj.weight": "W8A16",
  "model.layers.0.self_attn.q_proj.weight_scale": "W8A16",
  "model.layers.0.self_attn.q_proj.weight_offset": "W8A16",
}
```

量化后的MatMul权重新增weight_scale和weight_offset，用于对MatMul的计算结果进行反量化。

图 6-2 量化权重推理时流程



此量化方式支持量化float16或bfloat16类型的原始权重。

表 6-4 float16 权重量化后 dtype 及 shape 信息（假设原始权重的 shape 为[n, k]）

Tensor信息	weight	weight_scale	weight_offset	bias
dtype	int8	float32	float32	float16
shape	[n, k]	[n, 1]	[n, 1]	[n]

表 6-5 bfloat16 权重量化后 dtype 及 shape 信息（假设原始权重的 shape 为[n, k]）

Tensor信息	weight	weight_scale	weight_offset	bias
dtype	int8	float32	float32	float32
shape	[n, k]	[n, 1]	[n, 1]	[n]

说明

仅当浮点权重存在bias场景时，量化权重才会有bias。

生成权重

以LLaMa2-70B为例，您可以使用以下指令生成W8A16量化权重。

```
cd ${ATB_SPEED_HOME_PATH}
python -m examples.convert.model_slim.quantifier --model_path {浮点权重路径} --save_directory {W8A16量化权重路径} --calib_file= --w_bit 8 --a_bit 16 --act_method 3 --tokenizer_args
'{"padding_side":"left","pad_token":"<unk>"}'
```

说明

LLaMa1_65b和LLaMa2_70b用的是一个type。

- 以上指令包含生成LLaMa2_70B W8A16权重的最优参数配置，不同模型的参数配置不同，请参考模型Readme文件。
- W8A16量化时无需校准数据集，故calib_file传入空字符串即可。
- W8A16量化权重的config.json中应包含quantize字段，其值为"w8a16"。

执行推理

以LLaMa2-70B为例，您可以使用以下指令执行对话测试，推理内容为"What's deep learning?"。


```
cd ${ATB_SPEED_HOME_PATH}
bash examples/models/llama/run_pa.sh {W8A16量化权重路径}
```

6.1.5 W8A8SC 稀疏量化

简介

大模型稀疏量化工具包括稀疏、量化和压缩三个部分：

- 稀疏：模型稀疏工具通过算法判断模型权重中每个元素对精度结果的重要性，并将模型权重中对最终精度影响小的权重值置零。
- 量化：对权重和激活值都做量化，将高位浮点数转为8bit，可以直接降低权重体积，带来性能收益。
- 压缩：权重压缩工具将模型权重通过压缩算法进一步编码压缩，最大程度地降低权重体积，生成压缩后权重和索引文件。

说明

- 压缩算法和硬件强相关，仅Atlas 300I Duo 推理卡支持稀疏量化。
- bfloat16权重不支持稀疏量化。

稀疏+量化后权重目录结构：

```
├─ config.json
├─ quant_model_weight_w8a8s.safetensors
├─ quant_model_description_w8a8s.json
├─ tokenizer_config.json
├─ tokenizer.json
└─ tokenizer.model
```

- 量化后产物包含：权重文件quant_model_weight_w8a8s.safetensors和权重描述文件quant_model_description_w8a8s.json。
- 目录中的其余文件为推理时所需的配置文件，不同模型略有差异。

以下展示了量化后权重描述文件quant_model_description_w8a8s.json中的部分内容：

```
{
  "model_quant_type": "W8A8S",
  "model.embed_tokens.weight": "FLOAT",
  "model.layers.0.self_attn.q_proj.weight": "W8A8S",
  "model.layers.0.self_attn.q_proj.input_scale": "W8A8S",
  "model.layers.0.self_attn.q_proj.input_offset": "W8A8S",
  "model.layers.0.self_attn.q_proj.quant_bias": "W8A8S",
  "model.layers.0.self_attn.q_proj.deq_scale": "W8A8S",
}
```

量化后的MatMul权重新增input_scale、input_offset、quant_bias和deq_scale。其中input_scale和input_offset用于对激活值进行量化。MatMul使用量化后的激活值和量化权重进行计算。quant_bias和deq_scale用于对MatMul的计算结果进行反量化。

压缩后权重目录结构：

```
├─ config.json
├─ part0-of-4
│  ├─ quant_model_weight_w8a8sc.safetensors
│  └─ quant_model_description_w8a8sc.json
├─ part1-of-4
│  ├─ quant_model_weight_w8a8sc.safetensors
│  └─ quant_model_description_w8a8sc.json
├─ part2-of-4
│  ├─ quant_model_weight_w8a8sc.safetensors
```



```
│   └── quant_model_description_w8a8sc.json
├── part3-of-4
│   ├── quant_model_weight_w8a8sc.safetensors
│   └── quant_model_description_w8a8sc.json
├── tokenizer_config.json
├── tokenizer.json
└── tokenizer.model
```

压缩前会先加载权重，并进行多卡切分，压缩算法须在切分后的权重上执行。

以下展示了量化后权重描述文件part0-of-4/quant_model_description_w8a8sc.json中的部分内容：

```
{
  "model_quant_type": "W8A8SC",
  "model.embed_tokens.weight": "FLOAT",
  "model.layers.0.self_attn.query_key_value.weight": "W8A8SC",
  "model.layers.0.self_attn.query_key_value.index": "W8A8SC",
  "model.layers.0.self_attn.query_key_value.info": "W8A8SC",
  "model.layers.0.self_attn.query_key_value.input_scale": "W8A8S",
  "model.layers.0.self_attn.query_key_value.input_offset": "W8A8S",
  "model.layers.0.self_attn.query_key_value.deq_scale": "W8A8S",
  "model.layers.0.self_attn.query_key_value.quant_bias": "W8A8S",
}
```

压缩后的MatMul权重相比量化新增了index，压缩信息用于复原权重。

图 6-3 量化权重推理时流程

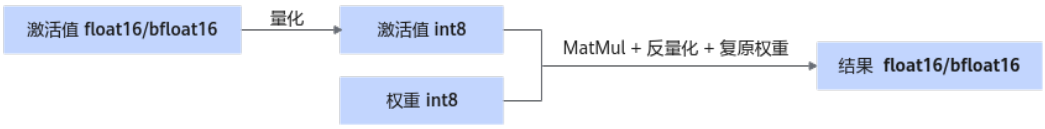


表 6-6 float16 权重量化后 dtype 及 shape 信息（假设原始权重的 shape 为[n, k]）

Tensor 信息	weight	input_scale	input_offset	quant_bias	deq_scale	index
dtype	int8	float16	int8	int32	int64	int8

Tensor信息	weight	input_scale	input_offset	quant_bias	deq_scale	index
shape	[x] x的取值范围在(0, n * k)之间。	[1]	[1]	[n]	[n]	[y] y由以下计算得出。 y = k_index * n_index * 8 k_index = ceil(k1 / tilingK) n_index = ceil(n1 / tilingN) k1 = k / 32 n1 = n / 16 其中，ceil()为向上取整函数，tilingK和tilingN为稀疏量化默认参数。

生成权重

以LLaMa-33B为例：

步骤1 使用以下指令生成W8A8S量化权重。

```
cd ${ATB_SPEED_HOME_PATH}
get_down_proj_disable_name() {
    local num_layer=$1
    local disable_names=""
    for ((i=0; i<$num_layer; i++)); do
        disable_names="$disable_names model.layers.$i.mlp.down_proj"
    done
    echo "$disable_names"
}
disable_names=$(get_down_proj_disable_name 60)
python -m examples.convert.model_slim.quantifier --model_path {浮点权重路径} --save_directory {W8A8S量化权重路径} --calib_file $llm_path/examples/convert/model_slim/boolq.jsonl --disable_names $disable_names --act_method 2 --do_smooth True --use_sigma True --is_lowbit True --co_sparse True --w_bit 4 --tokenizer_args '{"padding_side":"left","pad_token":"<unk>"}'
```

- 以上指令包含生成LLaMa-33B W8A8SC 稀疏量化权重的最优参数配置，不同模型的参数配置不同，请参考模型Readme文件。
- 生成权重后需将浮点权重下的special_tokens_map.json文件复制到W8A8S量化权重路径
- W8A8S量化权重的config.json中应包含quantize字段，其值为"w8a8s"。

步骤2 使用以下指令对量化权重进行压缩，生成W8A8SC量化权重。

```
torchrun --nproc_per_node 2 -m examples.convert.model_slim.sparse_compressor --model_path {W8A8S量化权重路径} --save_directory {W8A8SC量化权重路径}
```


- W8A8SC量化权重的config.json中应包含quantize字段，其值为"w8a8sc"。

----结束

执行推理

以LLaMa-33B为例，您可以使用以下指令执行对话测试，推理内容为"What's deep learning?"。

```
cd ${ATB_SPEED_HOME_PATH}
bash examples/models/llama/run_pa.sh {W8A8SC量化权重路径}
```

6.1.6 KV Cache int8

简介

此量化方式将k cache和v cache量化为8 bit，通过减少KV Cache的显存占用，提升吞吐。

说明

- 仅Atlas A2 推理系列产品支持KV Cache int8量化。
- KV Cache int8仅支持搭配W8A8使用。
- bfloat16权重不支持KV Cache int8量化。

KV Cache int8搭配W8A8量化后权重目录结构：

```
├── config.json
├── quant_model_weight_w8a8.safetensors
├── quant_model_description_w8a8.json
├── tokenizer_config.json
├── tokenizer.json
└── tokenizer.model
```

- 量化输出包含：权重文件quant_model_weight_w8a8.safetensors和权重描述文件quant_model_description_w8a8.json。
- 目录中的其余文件为推理时所需的配置文件，不同模型略有差异。

以下展示了量化后权重描述文件quant_model_description_w8a8.json中的部分内容：

```
{
  "model_quant_type": "W8A8",
  "kv_cache_type": "C8",
  "model.embed_tokens.weight": "FLOAT",
  "model.layers.0.self_attn.q_proj.weight": "FLOAT",
  "model.layers.0.self_attn.k_proj.weight": "FLOAT",
  "model.layers.0.self_attn.k_proj.kv_cache_scale": "W8A8",
  "model.layers.0.self_attn.k_proj.kv_cache_offset": "W8A8",
  "model.layers.0.self_attn.v_proj.weight": "FLOAT",
  "model.layers.0.self_attn.v_proj.kv_cache_scale": "W8A8",
  "model.layers.0.self_attn.v_proj.kv_cache_offset": "W8A8",
}
```

和W8A8量化权重相比，新增kv_cache_type描述字段，新增kv linear激活值量化缩放因子权重文件kv_cache_scale和kv linear激活值量化偏移值权重文件kv_cache_offset。推理时会基于这两个权重，推导出

k_quant_scale，k_dequant_scale，v_quant_scale，v_dequant_scale，k_quant_offset，k_dequant_offset，v_quant_offset和v_dequant_offset。其中quant_scale和quant_offset用于将k和v特征量化为int8类型，dequant_scale和dequant_offset用于将Paged attention的输出反量化为浮点类型。

图 6-4 量化权重推理时流程

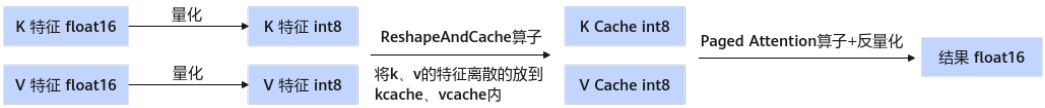


表 6-7 float16 权重量化后 dtype 及 shape 信息（假设原始权重的 shape 为[n, k]）

Tensor信息	kv_cache_scale	kv_cache_offset
dtype	float16	float16
shape	[kv_head_num * kv_head_dim]	[kv_head_num * kv_head_dim]

表 6-8 bfloat16 权重量化后 dtype 及 shape 信息（假设原始权重的 shape 为[n, k]）

Tensor信息	kv_cache_scale	kv_cache_offset
dtype	bfloat16	bfloat16
shape	[kv_head_num * kv_head_dim]	[kv_head_num * kv_head_dim]

生成权重

以LLaMa3.1-70B为例，您可以使用以下指令生成W8A8 + KV Cache int8量化权重。

```
export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
cd ${ATB_SPEED_HOME_PATH}
python examples/models/llama3/convert_quant_weights.py --model_path {模型权重路径}
--save_directory {量化模型保存路径} --w_bit 8 --a_bit 8 --device_type npu
--act_method 3
--anti_method m3 --disable_level L0
--calib_file { boolq.jsonl路径}
--use_kvcache_quant True
```

- 相比于W8A8的量化方式，需要新增use_kvcache_quant参数。
- KV Cache int8量化权重的config.json中应包含quantization_config字段，其中包含"kv_quant_type": "C8"键值对。

执行推理

和W8A8量化权重执行推理的方式相同，请参考**6.1.3 W8A8**。

6.1.7 Anti-Outlier 离群值处理

简介

离群值处理旨在解决在模型量化过程中由于数据分布异常导致的量化精度损失问题。在大模型量化中，离群值可能会导致量化后的模型性能下降，因为量化过程需要将连续的浮点数值转换为离散的整数值，而离群值可能会超出预期的量化范围。通过离群

值抑制的技术手段可以减少或消除离群值对量化模型性能的影响，确保量化后的模型具有良好的精度表现。

Anti-Outlier量化可以配合其他量化方式一起使用。当前支持：W8A8 + Anti-Outlier，W8A16 + Anti-Outlier和W8A8SC + Anti-Outlier。

以下展示了W8A8 + Anti-Outlier量化后权重描述文件quant_model_description_w8a8.json中的部分内容：

```
{
  "model_quant_type": "W8A8",
  "model.embed_tokens.weight": "FLOAT",
  "model.layers.0.input_layernorm.weight": "FLOAT",
  "model.layers.0.input_layernorm.module.weight": "W8A8",
  "model.layers.0.input_layernorm.module.bias": "W8A8"
}
```

新增input_layernorm.module.weight和input_layernorm.module.bias权重，用于对激活值进行离群值处理。

图 6-5 量化权重推理时流程

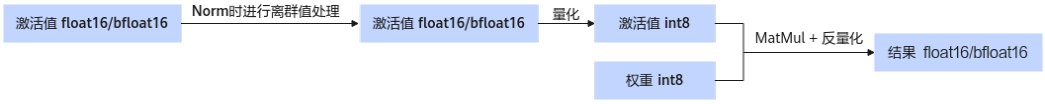


表 6-9 权重量化后 dtype 及 shape 信息（假设原始权重的 shape 为[n]）

Tensor信息	layernorm.module.weight	layernorm.module.bias
dtype	float32	float32
shape	[n]	[n]

生成权重

以LLaMa2-7B为例，您可以使用以下指令生成W8A8量化权重。

```
cd ${ATB_SPEED_HOME_PATH}
python examples/models/llama/convert_quant_weights.py --model_path {浮点权重路径} --save_directory {W8A8量化权重路径} --w_bit 8 --a_bit 8 --disable_level L0 --device_type cpu --anti_method m1 --act_method 1 --calib_file ${llm_path}/examples/convert/model_slim/boolq.jsonl
```

- 相比于W8A8的量化方式，需要新增anti_method 参数。
- 不同模型对anti_method参数的配置不同，请参考模型Readme文件。

执行推理

以LLaMa2-7B为例，您可以使用以下指令执行对话测试，推理内容为"What's deep learning?"。

```
cd ${ATB_SPEED_HOME_PATH}
bash examples/models/llama/run_pa.sh {W8A8量化权重路径}
```


6.2 长序列特性介绍

长序列特性的主要要求是在输入文本超长的场景下，模型回答的效果及性能也可以同时得到保障，其中输入的序列长度大多长于32K，甚至可到达1M的级别。其主要的技术难点是对attention部分以及KV Cache的使用方面的优化。在长序列场景下，由Attention和KV Cache部分造成的显存消耗会快速的成倍增长。因此对这部分显存的优化便是长序列特性的关键技术点。其中涉及到诸如KV Cache量化，kv多头压缩，训短推长等关键算法技术。

- 训长推长：在训练时通过较长的文本对模型的权重进行训练，从而使得模型在推理过程中对长序列输入依然可以保持良好的模型能力。
- 训短推长：模型本身通过诸如Alibi编码或序列压缩算法如NTK，YaRN等技术使得模型具备较强的自扩张能力，从而可以通过短序列训练后在长序列推理场景下获得更好的模型能力。

限制与约束

目前MindIE LLM对最大序列长度的支持主要受限于以下两类因素：

- 硬件的显存规格与模型参数量。该因素决定了在硬件允许的情况下，模型在推理时所能接受的最长输入长度。以64G Atlas A2 推理系列产品硬件为例，使用8卡运行Glm4-9B-Chat模型，在显存允许的条件下能进行最长1M的长序列推理。
- 模型权重和结构等对长序列能力的支持。该因素决定了模型在长序列场景下的生成和对话效果。对于训长推长的模型（例如Glm-4-9B-Chat-1M），MindIE LLM能保证与开源实现相同的长序列推理效果；对于训短推长的模型（例如Qwen1-7B），MindIE LLM中实现了NTK等相应技术，在使能相关特性后也能让模型具备与开源实现同等的长序列推理能力。需要注意的是，若想使原生仅支持短序列的模型来处理长序列输入，MindIE LLM无法保证其长序列推理结果的合理性。

执行推理

请兼顾硬件规格、模型参数及模型支持的最大有效推理长度来确定合适的序列长度。具体规格请参考对应模型的官方介绍文档。长序列推理的使能方式与普通模型推理方式一致，仅需要将长序列文本按照正常推理流程传入模型即可。详细的执行模型推理的流程请参考**4 模型推理使用流程**获取更多信息和支持。运行支持长序列的模型的方式，与运行其他模型无异。

以LLaMa3模型为例，您可以使用以下指令执行对话测试，推理内容为"What's deep learning"。

```
cd ${ATB_SPEED_HOME_PATH}
bash examples/models/llama3/run_pa.sh {模型权重路径} {输出长度}
```

6.3 多机特性介绍

针对于模型参数特别大的场景（如175B以上参数量模型）以及长序列场景，用户可选择使用多台服务器组合进行推理以达到较优效果，多机通信目前有两种硬件连接方式，PCIE建联以及RDMA建联。使用多机推理时，需要将包含设备ip，进程号信息，服务器ip信息的json文件地址传递给底层通信算子。

多机通信有多种并行策略可以结合，目前主流的并行策略包括Tensor Parallel（TP，张量并行），Pipeline Parallel（PP，流水并行），Sequence Parallel（SP，序列并行）

以及MoE（Mixture of Experts）类模型特有的Expert Parallel（EP，专家并行）。通过不同的并行策略的结合，可以最大化整体吞吐量，获得最优性能表现，目前模型默认使用纯TP并行方式执行推理，部分开源模型已经完成多机多卡推理能力构建，其中包括LLaMa系列模型(包括基于LLaMa实现的模型)，DeepSeek-MoE以及Mixtral-MoE。

限制与约束

- 推荐支持模型同时部署在2台Atlas A2 推理系列产品上。
- 最多支持模型同时部署在4台Atlas A2 推理系列产品上。
- Atlas 300I Duo 推理卡不支持多机模型部署。

模型适配

模型需预先基于ATB加速库组图方式完成流水图构建，即模型可以通过MindIE LLM调用ATB加速库组图的方式实现单机推理，在此基础上，需要将一个json文件地址传递给通信算子用于设备的ip，进程关系解析，本文件需借助hccn_tool手动生成，生成方式如下：

说明

hccn tool工具的安装及使用详见**HCCN Tool 接口参考**。

- **获取卡号对应ip信息：**

  在裸机上执行以下命令：

  ```
  for i in {0..7}
  do
     hccn_tool -i $i -ip -g
  done
  ```

  获得输出如图所示：

  ```
  root@huawei89:~#
  for i in {0..7}
  do
  hccn_tool -i $i -ip -g
  done
  ipaddr:10.10.100.10
  netmask:255.255.255.0
  ipaddr:10.10.100.11
  netmask:255.255.255.0
  ipaddr:10.10.100.12
  netmask:255.255.255.0
  ipaddr:10.10.100.13
  netmask:255.255.255.0
  ipaddr:10.10.100.14
  netmask:255.255.255.0
  ipaddr:10.10.100.15
  netmask:255.255.255.0
  ipaddr:10.10.100.16
  netmask:255.255.255.0
  ipaddr:10.10.100.17
  netmask:255.255.255.0
  root@huawei89:~#
  ```

- **检查两台机器连通性：**

  使用如下命令检查两台机器双卡之间连通性：

  ```
  hccn_tool -i 0 -ping -g address 1.1.1.1
  ```

  其中1.1.1.1为对端机器每一张npu卡的ip地址，如果连接成功应当有输出如下：

说明

多机参数面互联：即服务器NPU卡对应的端口处在同一个VLAN，可以通过ROCE互通。

```
root@huawei:~# hccn_tool -i 0 -ping -g address 10.10.100.10
device 0 PING 10.10.100.10
recv seq=0,time=0.067000ms
recv seq=1,time-0.049000ms
recv seq=2,time=0.047000ms
3 packets transmitted, 3 received, 0.00% packet loss
root@huawei: #
```

如果存在丢包，则会输出如下内容，如存在此种情况，请优先对硬件连通性进行检查校验。

```
recv time out seq=O
recv seq=1,time=0.097000ms
recv seq=2,time=0.066000ms
3 packets transmitted, 2 received, 33.33% packet loss
```

- **生成RankTableFile**

  在确认了两机连通性无误后，可通过查询到的ip信息生成RankTableFile，步骤如下：

  新建一个ranktablefile.json文件，将样例的内容复制粘贴到json文件里，得到RankTableFile接下来需手动将样例中的加粗内容替换为查询到的机器实际ip等配置信息。

```
{
   "server_count":"2",
   "server_list":
   [
     {
         "device":[
                {
                "device_id":"0",
                "device_ip":"10.10.100.10",
                "rank_id":"0"
                },
                {
                "device_id":"1",
                "device_ip":"10.10.100.11",
                "rank_id":"1"
                },
                {
                "device_id":"2",
                "device_ip":"10.10.100.12",
                "rank_id":"2"
                },
                {
                "device_id":"3",
                "device_ip":"10.10.100.13",
                "rank_id":"3"
                }],
          "server_id":"90.90.152.89"
      },
      {
        "device":[
        {
               "device_id":"0",
               "device_ip":"10.10.100.22",
               "rank_id":"4"
               },
               {
               "device_id":"1",
               "device_ip":"10.10.100.23",
               "rank_id":"5"
               },
               {
               "device_id":"2",
```



```
                "device_ip":"10.10.100.24",
                "rank_id":"6"
                },
                {
                "device_id":"3",
                "device_ip":"10.10.100.25",
                "rank_id":"7"
                }
                  ],
        "server_id":"90.90.152.91"
    }
  ],
  "status":"completed",
  "version":"1.0"
  }
```

执行推理

若需要实现多机推理，在基础单机推理的启动脚本基础上，需要做出以下适配修改，首先应声明以下额外的环境变量，以双机16卡为例：

```
export BACKEND=hccl
export WORLD_SIZE=16
export RANKTABLEFILE=""（引号内填入RankTableFile的绝对路径）
```

说明

多机推理时，每一台机器上的推理脚本都需要进行单独的适配修改，在执行推理时，每一台机器上的推理脚本都需要被同步执行。

将torchrun进行进程拉起的部分替换为如下使用python的方式进行拉起：

```
cd ${ATB_SPEED_HOME_PATH}
RANK_ID_START=0  #主节点该值为0，副节点此值应为（WORLD_SIZE / 2）
WORLD_SIZE_LOCAL=8
for((RANK_ID=$RANK_ID_START;RANK_ID<$((WORLD_SIZE_LOCAL+RANK_ID_START));RANK_ID++));
do
export RANK=$RANK_ID
export LOCAL_RANK=$RANK_ID  #主节点local_rank与rank相同，副节点local_rank = rank - word_size_local
python3 ./examples/run_pa.py  --model_path /data/atb_testdata/weights/llama2-70b/ \
--max_output_length 35  --max_batch_size 1 --input_text "What's Deep Learning?" &
done
wait
```

6.4 MoE 特性介绍

Mixture of Experts（MoE）在传统transformer结构的基础上进行了两个创新。第一个部分是用Sparse MoE layer来替换transformer结构中Feed Forward Network（FFN）。每一个FFN可扮演一个专家的角色，但针对每一个token的推理，仅需激活其中部分专家即可。这部分激活专家的筛选就涉及到了MoE的第二个关键机制：路由（routing）机制，这个Router决定了token在每一层会进入到哪一个专家。基于这两个机制的结合，MoE模型得益于其广阔的专家知识可以保证很高的模型效果，但相较于同等参数量的传统模型，他只需要激活其中部分专家，便又能同时保证其优秀的推理性能。

MoE结构的典型代表模型有Mixtral 8*7B，Mixtral 8*22B，DeepSeek-16B-MoE，DeepSeek-V2，Grok等。

限制与约束：

表 6-10 能力支持特征矩阵

已支持模型	数据格式	量化	并行方式	硬件平台	多机多卡推理
Mixtral 8*7B	FP16	暂不支持	TP	Atlas A2 推理系列产品	不支持
Mixtral 8*22B	FP16	暂不支持	TP	Atlas A2 推理系列产品	不支持
Deepseek-16B-MoE	FP16	暂不支持	TP	Atlas A2 推理系列产品	不支持
DeepSeek-V2	FP16	支持	TP	Atlas A2 推理系列产品	支持

模型配置参数

模型固有参数配置请参考官方权重文件中的config.json文件。

执行推理

MoE类模型执行推理的方式与其他模型一致，在执行推理时您可参考传统LLM的使用方式，无需做额外配置修改。

以DeepSeek-16B-MoE为例，您可以使用以下指令执行对话测试，推理内容为"What's deep learning"。

```
cd ${ATB_SPEED_HOME_PATH}
bash examples/models/deepseek/run_pa_deepseek_moe.sh {模型权重路径}
```

6.5 Multi-LoRA 特性介绍

LoRA（Low-Rank Adaptation）是一种高效的参数微调方法。将大模型的权重矩阵分解为原始权重矩阵和两个低秩矩阵的乘积，即W' = W + BA。由于B和A矩阵参与训练的参数量远小于原始权重，其乘积结果又能合入线性层并向下传递，从而达到大模型轻量级微调的目的。

Multi-LoRA指基于一个基础模型，使用多个不同的LoRA权重进行推理。每个请求带有指定的LoRA ID，推理时动态匹配对应的LoRA权重。部署服务时，LoRA权重和基础模型权重预先加载至显存中。一个推理请求至多使用一个LoRA权重，兼容推理请求不使用LoRA权重的情况。对于大参数量的模型，若模型参数量过大，无法单卡加载时，可进行Tensor Parallel并行。

限制与约束

- 仅Atlas A2 推理系列产品硬件支持此特性。
- LoRA权重个数上限受硬件显存限制，建议数量为小于等于10个。
- 不支持LoRA权重热加载。

- 不支持和量化、PD分离、并行解码、SplitFuse以及Prefix Cache特性同时开启。
- 仅支持线性层携带Lora权重。
- 仅Qwen1.5-14B、Qwen1.5-72B、LLaMa3.1-70B和Qwen2-72B支持Multi-LoRA特性。
- 在lora_adapter.json文件中配置的Lora权重名称长度不能超过256个字符。
- 仅支持vLLM/generate接口。

LoRA权重中需包含"adapter_config.json"和"adapter_model.safetensors"文件。"adapter_config.json"中包含LoRA权重的超参，支持配置以下参数：r（LoRA 微调中的秩大小），rank_pattern，lora_alpha（LoRA低秩矩阵的缩放系数）和alpha_pattern。"adapter_model.safetensors"文件中包含权重，权重以键值对的形式保存，其中LoRA权重的键名在基础模型的键名前后增加"base_model.model"前缀和"lora_A.weight"、"lora_B.weight"后缀。例如：基础模型中的键名为"model.layers.9.self_attn.v_proj.weight"，则LoRA权重中对应的键名应为"base_model.model.model.layers.9.self_attn.v_proj.lora_A.weight"和"base_model.model.model.layers.9.self_attn.v_proj.lora_B.weight"。

执行推理

已在环境上安装CANN和ATB Models，详情请参见《**MindIE安装指南**》。

说明

本次样例参考以下安装路径进行：

安装ATB Models并初始化ATB Models环境变量。模型仓set_env.sh脚本中有初始化“${ATB_SPEED_HOME_PATH}”环境变量的操作，所以source模型仓中set_env.sh脚本时会同时初始化“${ATB_SPEED_HOME_PATH}”环境变量。

以LLaMa3.1-70B为例，下载基础模型和LoRA权重后，在基础模型路径下新增lora_adapter.json文件配置预加载的LoRA权重。其中权重名称为权重的别名，长度不能超过256个字符，在后续请求中用于指定Lora权重进行推理。

```
cd ${基础模型权重}
vi lora_adapter.json
# 在此文件中配置adapter名称及路径，示例：
{"${Lora权重1的名称}": "${Lora权重1路径}", "${Lora权重2的名称}": "${Lora权重2路径}"}
```

您可以使用以下指令执行对话测试，共3个请求组成组成 Batch进行推理，每个推理请求中LoRA权重不同。run_pa脚本参数参考**表4-2**章节。

```
cd ${ATB_SPEED_HOME_PATH}
torchrun --nproc_per_node 8 --master_port 20030 -m examples.run_pa --model_path ${基础模型权重} --max_output_length 20 --max_batch_size 3 --input_dict '[{"prompt": "A robe takes 2 bolts of blue fiber and half that much white fiber.  How many bolts in total does it take?", "adapter": "${Lora权重1的名称}"}, {"prompt": "A robe takes 2 bolts of blue fiber and half that much white fiber.  How many bolts in total does it take?", "adapter": "${Lora权重2的名称}"}, {"prompt": "What is deep learning?", "adapter": "base"}]'
```

6.6 PD 分离特性介绍

Transformer大模型推理PD分离部署特性，主要是指模型推理的prefill阶段和decode阶段分别实例化部署在不同的机器资源上同时进行推理，其结合prefill阶段的计算密集型特性，以及decode阶段的访存密集型特性，通过调节PD节点数量配比来提升decode节点的batch size来充分发挥NPU卡的算力，进而提升集群整体吞吐。此外，在decode平均低时延约束场景，PD分离相比PD混合部署，更加能够发挥性能优势。

MindIE LLM提供PD分离部署所需要的关键能力，包括PD按角色实例化，PD KV Cache高性能传输，计算传输并行，batch调度。

限制与约束

- 仅Atlas A2 推理系列产品硬件支持此特性。
- 不同P、D节点使用的NPU卡数量必须相同。
- NPU网口互联（带宽：200Gbps）。
- Server服务化支持PD分离，包括重计算、集群调度等特性。
- 不支持和Multi-LoRA、并行解码、SplitFuse以及Prefix Cache特性同时使用。

表 6-11 依赖部件说明

组件	用途说明
MindIE MS	MindIE MS主要负责P、D实例的生命周期管理、状态采集、请求调度等。 逻辑上包含2部分：Controller和Coordinator。 其中Controller主要负责P、D实例生命周期管理，Coordinator主要负责P、D请求的调度。
MindIE Service	通过endpoint方式接收Coordinator推理请求。
MindIE LLM BatchScheduler	调度batch能力，单独调度prefill或decode类型的请求并下发batch。
MindIE LLM	提供基础的模型执行能力。
CANN KV库	提供基于RDMA的KV Cache传输能力。

表 6-12 ServerConfig 补充参数

配置项	取值类型	取值范围	配置说明
InferMode	std::string	"dmi"或者"standard"	• dmi为PD分离模式，该模式下服务化和模型启动解耦。待下发P/D身份后才拉起模型。standard为非PD分离模式，服务化和模型启动不解耦。 • 必填，默认值："standard"。

接口说明

请参见 **SetReqType接口**~ **GetSrcBlockTable接口**章节。

执行推理

此特性需要配合MindIE Service使用。请参考《 MindIE Service开发指南 》的“集群服务部署 > **PD分离服务部署**”章节。

6.7 SplitFuse 特性介绍

SplitFuse特性的目的是将长prompt request分解成更小的块，并在多个forward step中进行调度，只有最后一块的forward完成后才开始这个prompt request的生成。将短prompt request组合以精确填充step的空隙，每个step的计算量基本相等，达到所有请求平均延迟更稳定的目的。

两个关键行为：

1. 长prompts被分解成更小的块，并在多个迭代中进行调度，只有最后一遍迭代执行输出生成token。
2. 短prompts也可能切分成小块，以确保计算效率发挥最佳。

其优势主要包括：

- **提高响应速度**：减少长prompt处理延迟，提升用户体验。
- **提升效率**：通过合理组合短prompt，保持模型高吞吐量运行。
- **增强一致性**：统一前向传递大小，降低延迟波动，使生成频率更稳定。

限制与约束

- Atlas A2 推理系列产品硬件支持此特性。
- LLaMa3.1-70b浮点模型和Qwen2-72B浮点模型支持对接此特性。
- 该特性不能和PD分离、Multi-LoRA、并行解码、Prefix Cache、多机推理和长序列特性同时使用。
- 该特性暂不支持与量化特性同时使用。
- 该特性暂不支持python组图。
- SplitFuse暂不支持以下后处理模式：
  - 忽略结束符继续推理。
  - 遇到stop词时答案是否包含stop词。
  - 回答是否跳过特殊token。
  - 推理在特定字符串停止。
  - 推理在特定token_id停止。

开启SplitFuse特性，需要配置的补充参数如下表6-13及表6-14所示：

表 6-13 SplitFuse 补充参数 1：**ModelDeployConfig 中的 ModelConfig 参数**

配置项	取值类型	取值范围	配置说明
plugin_params	std::string	"{\"plugin_type\":\"splitfuse\"}"或""	不添加该字段或将该字段设置为""均表示默认不生效任何插件功能。 "{\"plugin_type\":\"splitfuse\"}"表示执行splitfuse。 **约束：若enableSplit开启或templateType为"Mix"，则此处必须开启为splitfuse（特性不开启时非必填项）。**

表 6-14 SplitFuse 补充参数 2：**ScheduleConfig 的参数**

配置项	取值类型	取值范围	配置说明
templateType	std::string	"Standard"或"Mix"	"Mix"为prefill和decode可一起组batch。 "Standard"为默认值（特性不开启时为必填项），表示prefill和decode各自分别组batch。
policyType	uint32_t	• 0 • 4 • 5 • 6 • 7	• 0: 先入先出策略（FCFS）。 • 4：短任务优先（SJF）。 • 5：长任务优先（LJF）。 • 6：多级反馈队列（Skip-Join MLFQ）。 • 7：短任务优先多级反馈队列（SJF-MLFQ）。 建议：一般使用FCFS策略；对于追求极限吞吐的场景建议使用SJF，但是可能会导致长输入等待时间大幅增长；对于希望在吞吐和长输入等待取平衡的使用场景，建议使用Skip-Join MLFQ或SJF-MLFQ策略。
enableSplit	bool	• true • false	• true：会对prompt进行动态切分。 • false：不会对prompt进行动态切分。 **约束：开启时templateType必须为Mix（特性不开启时非必填项）。**

配置项	取值类型	取值范围	配置说明
splitType	bool	• true • false	• true：优化切分长度，仅对超出切分长度2倍的prompt进行切分。 • false：按照配置的切分长度切分。 默认值：false。
splitStartType	bool	• true • false	• true：每次组batch时重置切分状态，重新判断当前是否满足splitStartBatchSize。 • false：首次满足splitStartBatchSize条件后，不再重置切分状态。 默认值：false。
splitChunkTokens	uint32_t	>=1	当前参与组batch的总token个数。 默认值：512，建议不低于512。
splitStartBatchSize	uint32_t	[0,maxBatchSize]	当batch数达到该值时开启切分。 默认值：16。

执行推理

步骤1 配置服务化参数。该特性需配合MindIE Service使用，按照**表6-13**及**表6-14**在服务化的config.json文件中添加相应参数，config.json文件路径的详细说明请参考《MindIE安装指南》中“配置MindIE > 配置MindIE Server > **单机推理**”章节中的软件包文件清单。

步骤2 启动服务。启动服务化即为开启splitfuse特性。启动服务方法具体请参考《MindIE Service开发指南》的“快速开始 > **启动服务**”章节。

----结束

6.8 并行解码特性介绍

在LLM的推理场景中，传统的Auto-Regressive Decoding之所以慢，本质上是step-by-step导致了并发性不够。推理阶段属于内存带宽受限，计算资源过剩，因此一个很自然的想法，是采用处理器中常用的“Speculative Execution”优化技术，通过额外的计算资源完成推测执行，提升并发性。

目前支持两种并行解码算法，差异主要在于候选token生成的方式不同。

1. memory_decoding：利用trie-tree（前缀树）缓存模型历史的输入输出，从中获取候选token。

2. lookahead：基于jacobi迭代并辅以Prompt以及输出结果生成候选token。

限制与约束

- Atlas A2 推理系列产品和Atlas 300I Duo 推理卡硬件支持此特性。
- LLaMa系列、Qwen1和Qwen1.5系列模型支持对接此特性。
- 并行解码支持的量化特性：W8A8量化与稀疏量化，其他量化特性暂不支持。
- 该特性不能和PD分离、Multi-LoRA、SplitFuse、Prefix Cache、长序列以及多机推理特性同时使用。
- 并行解码场景暂不支持python组图。
- 并行解码场景暂不支持流式推理。
- 并行解码惩罚类后处理仅支持重复惩罚。
- memory_decoding算法暂不支持open ai接口和多轮对话场景。
- lookahead和memory_decoding算法不可同时使能。

开启并行解码特性，需要配置的参数如表6-15~表6-19所示：

表 6-15 memory_decoding 补充参数 1：**ModelDeployConfig 中的 ModelConfig 参数**

配置项	取值类型	取值范围	配置说明
plugin_params	std::string	plugin_type：memory_decoding decoding_length：[1, 16] dynamic_algo：true或false	• plugin_type配置memory_decoding，表示当前选择memory_decoding并行解码。 • decoding_length为memory_decoding算法中的参数，表示候选token的最大长度，默认值16。 • dynamic_algo为可选参数，配为true时表示开启动态自适应候选长度功能，默认值False。 • 配置示例： {\"plugin_type\":\"memory_decoding\",\"decoding_length\": 16,\"dynamic_algo\": true} 或 {\"plugin_type\":\"memory_decoding\",\"decoding_length\": 16}

表 6-16 memory_decoding 补充参数 2：**ModelDeployConfig 的参数**

配置项	取值类型	取值范围	配置说明
speculationGamma	uint32_t	与plugin参数配置有关	memory_decoding时，该值配置应大于等于decoding_length。 建议值：等于decoding_length。

表 6-17 memory_decoding 补充参数 3：**ScheduleConfig 的参数**

配置项	取值类型	取值范围	配置说明
maxIterTimes	uint32_t	与plugin参数配置有关	如果dynamic_algo为true，该值需大于等于期望输出的长度+speculationGamma的值。 例：期望最大输出长度为512，则该值需要配置>=512+speculationGamma。

表 6-18 lookahead 补充参数 1：**ModelDeployConfig 中的 ModelConfig 参数**

配置项	取值类型	取值范围	配置说明
plugin_params	std::string	plugin_type：la level ： [2, 16] window ： [1, 16] guess_set_size ： [1, 16]	plugin_type配置la，表示当前选择lookahead并行解码。 level/window/guess_set_size为lookahead算法中的N/W/G参数，默认值为4/5/5，且每个参数可配置的上限不超过16。 配置示例： "{\"plugin_type\":\"la\",\"level\": 4,\"window\": 5,\"guess_set_size\": 5}"

表 6-19 lookahead 补充参数 2：ModelDeployConfig **的参数**

配置项	取值类型	取值范围	配置说明
speculationGamma	uint32_t	与plugin参数配置有关	lookahead中，配置值应大于等于(N-1)*(W+G) 建议值：等于(N-1)*(W+G)。

执行推理

步骤1 配置服务化参数。该特性需配合MindIE Service使用，按照**表6-15~表6-19**在服务化的config.json文件中添加相应参数，config.json文件路径的详细说明请参考《 MindIE安装指南 》中“配置MindIE > 配置MindIE Server > **单机推理**”章节中的软件包文件清单。

步骤2 启动服务。启动服务方法具体请参考《 MindIE Service开发指南 》的“快速开始 > **启动服务**”章节。

----结束

6.9 Prefix Cache 特性介绍

当前大语言模型推理系统普遍采用KV Cache缓存机制，但该机制存在以下两个问题：

1. 随着LLM支持的序列长度不断增长，KV Cache所需要的显存资源也急剧增加。
2. KV Cache只对当前session有效，如果跨session存在重复token序列的情况下无法实现复用。

Prefix Cache通过RadixTree保留session结束后的KV Cache，新的session请求在RadixTree中查找是否存在相同的Token序列，即可复用之前计算好的KV Cache，从而实现跨session的KV Cache复用。

其优势主要包括：

- **更短的prefill时间**：由于跨session的重复token序列对应的KV Cache可以复用，那么就可以减少一部分前缀token的KV Cache计算时间，从而减少prefill的时间。
- **更高效的显存使用**：当正在处理的sessions相互之间存在公共前缀时，公共前缀部分的KV Cache可以共用，不必重复占用多份显存。

限制与约束

- Atlas A2 推理系列产品和Atlas 300I Duo 推理卡硬件支持此特性。
- LLaMa系列、Qwen1.5和Qwen2系列模型支持对接此特性。
- 当跨session公共前缀token数大于等于Page Attention中的block size，才会进行公共前缀token的KV Cache复用。
- 该特性不能和PD分离、Multi-LoRA、SplitFuse、并行解码以及KV Cache量化特性同时使用。
- 暂不支持与LoRA特性配合。

开启Prefix Cache特性需要配置的补充参数如表6-20及表6-21所示：

表 6-20 Prefix Cache 补充参数 1：**ModelDeployConfig 中的 ModelConfig 参数**

配置项	取值类型	取值范围	配置说明
plugin_params	std::string	"{\"plugin_type\":\"prefix_cache\"}"或""	不添加该字段或将该字段设置为""均表示默认不生效任何插件功能。 "{\"plugin_type\":\"prefix_cache\"}"表示执行Prefix Cache。 **须知** • 若enablePrefixCache开启，则此处必须开启为prefix_cache。 • 若enablePrefixCache不开启，为非必填项。

表 6-21 Prefix Cache 补充参数 2：**ScheduleConfig 的参数**

配置项	取值类型	取值范围	配置说明
enablePrefixCache	bool	• true • false	• true：会开启Prefix Cache特性。 • false：不会开启Prefix Cache特性。

执行推理

步骤1 配置服务化参数。该特性需配合MindIE Service使用，按照**表6-20**及**表6-21**在服务化的config.json文件中添加相应参数，config.json文件路径的详细说明请参考：《 MindIE安装指南 》中“配置MindIE > 配置MindIE Server > **单机推理**”章节中的软件包文件清单。

步骤2 启动服务。具体请参考《 MindIE Service开发指南 》的“快速开始 > **启动服务**”章节。

----结束

7 服务化调度推理使用流程

7.1 Triton 基于 LLM Manager 接口开发指南

7.1.1 快速介绍

Triton 简介

Triton Inference Server是高性能开源推理服务框架，专为简化和加速机器学习模型在生产环境中的部署而设计。支持多种框架，如TensorFlow、PyTorch和ONNX等，且支持多种硬件平台上执行推理任务。Triton提供了灵活的模型管理、自动批处理、动态调度以及多模型并行推理等功能，使得开发者可以轻松扩展和优化模型的推理服务，从而更好地满足实时和批处理推理的需求。

基于MindIE LLM向Triton提供的统一接口LLM Manager进行了适配开发，主要为基于开源服务化框架进行深度业务适配与客户提供兼容方案，使其能够在昇腾硬件上运行Triton，协助客户降低迁移成本。

Triton 适配昇腾整体方案介绍

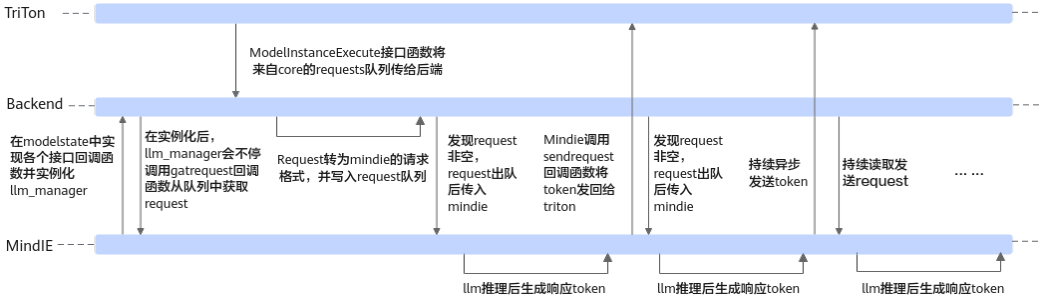
采用非侵入性适配，将Triton_Backend提供的C++后端接口逻辑与MindIE LLM推理运行逻辑对齐并打通，确保Triton能够正确的调用MindIE的推理功能并返回响应结果。

适配过程主要分为两部分：

- 面向Tritonserver中core层的北向（上层）接口设计，Triton提供了定制化接口函数，主要负责对Backend、Model、ModelInstacne进行实际初始化与销毁，同时接收来自core层的请求并设置后续的执行动作。
- 面向MindIE LLM的南向（下层）接口设计，MindIE LLM提供给Backend唯一核心接口类LLM Manager，具有模型推理和请求调度的能力。在适配过程中，需要实现推理请求的读取、解析和转换，并发送给LLM调度执行，最后将底层推理出的响应发送回Tritonserver侧。

Triton推理服务框架搭载MindIE在适配后的整体运行逻辑如图7-1所示。

图 7-1 运行逻辑



支持版本特性与模型

表 7-1 支持的版本特性及模型

支持的Triton版本	浮点	量化	MoE
v24.02	同MindIE LLM	同MindIE LLM	同MindIE LLM

7.1.2 适配说明

7.1.2.1 北向接口实现方式

实现方式

北向实现方式如图7-2所示，分别设计了四个核心接口A、B、C、D。接口A、B和C负责对Tritonserver传递给Backend的后端、模型和模型实例进行初始化并创建ModelState和ModelInstanceState这两个状态类，如表7-2所示。接口D将Triton侧请求发送给ModelInstanceState，在其内部转化为MindIE侧的请求和推理任务。最后发送给ModelState并添加进存储队列中，以供MindIE接口循环调用。

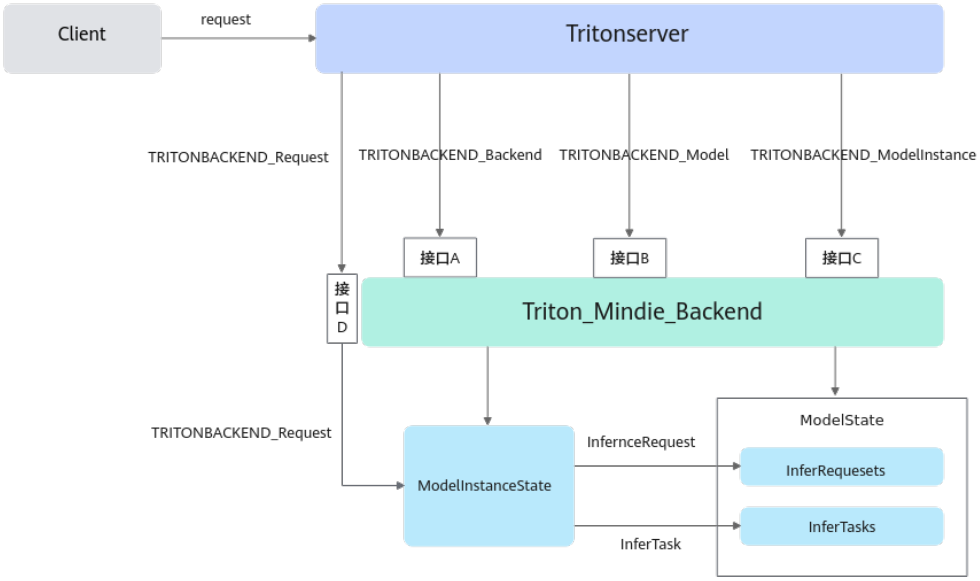
除了上述四个核心接口，还设计了两个接口B~和C~用于清除Model和ModelInstance中的状态类。

表 7-2 两个状态类

状态类	说明
ModelState	ModelState继承自BackendModel，承接了TRITONBACKEND_Model中的全部模型配置信息（如模型名、模型版本、模型仓路径和最大批处理量等），最后依附在TRITONBACKEND_Model上。ModelState中还包含了两个关键存储队列，一个是请求队列，另一个是推理任务队列，以供MindIE LLM接口循环调用。

状态类	说明
ModelInstanceState	ModelInstanceState继承自BackendModelInstance，承接了TRITONBACKEND_ModelInstacne中的全部模型实例配置信息（如实例ID、CPU/GPU类型和硬件ID），最后依附在TRITONBACKEND_ModelInstacne上。ModelInstanceState中定义了Enqueue成员函数，用来接收Triton侧请求并转化为MindIE侧请求InferRequest，同时生成推理任务inferTask，最后一同写入ModelState内队列中。

图 7-2 北向实现方式



接口描述

表 7-3 接口 A

参数	描述
接口函数	TRITONSERVER_Error* TRITONBACKEND_Initialize(TRITONBACKEND_Backend* backend)。
接口功能	Triton传入的backend支持的API版本号与mindie_backend进行校验。
输入参数	Tritonserver侧的backend类指针。
输出参数	执行结果状态。

表 7-4 接口 B

参数	描述
接口函数	TRITONSERVER_Error* TRITONBACKEND_ModelInitialize(TRITONBACKEND_Model* model)。
接口功能	创建一个ModelState并初始化，并将其依附在TRITONBACKEND_Model上。承接了TRITONBACKEND_Model中的全部模型配置信息（如模型名、模型版本、模型仓路径、max_batch_size等，并初始化了两个关键存储队列，一个是请求队列，另一个是推理任务队列，以供MindIE LLM接口循环调用。
输入参数	Tritonserver侧的模型类指针。
输出参数	执行结果状态。

表 7-5 接口 C

参数	描述
接口函数	TRITONSERVER_Error* TRITONBACKEND_ModelInstanceInitialize(TRITONBACKEND_ModelInstance* instance)。
接口功能	创建一个ModelInstanceState并初始化，并将其依附在TRITONBACKEND_ModelInstance上。承接了TRITONBACKEND_ModelInstacne中的全部模型实例配置信息（如实例ID、CPU/GPU类型、硬件ID），其中定义了Enqueue成员函数，用来接收Triton侧请求并转化为MindIE侧请求InferRequest，同时生成推理任务inferTask，最后一同写入ModelState内队列中。
输入参数	Tritonserver侧的模型实例类指针。
输出参数	执行结果状态。

表 7-6 接口 D

参数	描述
接口函数	接口函数：TRITONSERVER_Error* TRITONBACKEND_ModelInstanceExecute(TRITONBACKEND_ModelInstance* instance, TRITONBACKEND_Request** requests, const uint32_t request_count)。
接口功能	backend中执行推理的入口函数，首先从instance中提取出ModelInstanceState，接着调用Enqueue成员函数把所有请求放入model_state_中的请求队列requests_和推理任务哈希表inferTasksMap_，后面供LlmManager进行循环读取、推理。

参数	描述
输入参数	Tritonserver发送给backend的模型实例类指针、请求队列和请求数量。
输出参数	执行结果状态。

表 7-7 接口 B~

参数	描述
接口函数	接口函数：TRITONSERVER_Error* TRITONBACKEND_ModelFinalize(TRITONBACKEND_Model* model)。
接口功能	清除依附在TRITONBACKEND_Model上的模型状态ModelState。
输入参数	Tritonserver侧的模型类指针。
输出参数	执行结果状态。

表 7-8 接口 C~

参数	描述
接口函数	接口函数：TRITONSERVER_Error* TRITONBACKEND_ModelInstanceFinalize(TRITONBACKEND_ModelInstance* instance)。
接口功能	清除依附在TRITONBACKEND_ModelInstance上的实例状态ModelInstanceState。
输入参数	Tritonserver侧的模型实例类指针。
输出参数	执行结果状态。

适配流程

步骤1 通过**接口A**对TRITONBACKEND_Backend初始化：对triton::core传来的TritonBackend（强转成了TRITONBACKEND_Backend）进行初始化。主要对backend支持的API版本号等信息进行校验，代码如下：

```
TRITONSERVER_Error* TRITONBACKEND_Initialize(TRITONBACKEND_Backend* backend)
{
    const char* cname;
    RETURN_IF_ERROR(TRITONBACKEND_BackendName(backend, &cname));
    std::string name(cname);
    LOG_MESSAGE(
        TRITONSERVER_LOG_INFO,
        (std::string("TRITONBACKEND_Initialize: ") + name).c_str());
    // 检查当前triton支持的后端API版本是否支持当前的编译版本
    uint32_t api_version_major, api_version_minor;
    RETURN_IF_ERROR(
```



```
        TRITONBACKEND_ApiVersion(&api_version_major, &api_version_minor));
    LOG_MESSAGE(
        TRITONSERVER_LOG_INFO,
        (std::string("Triton TRITONBACKEND API version: ") +
        std::to_string(api_version_major) + "." +
        std::to_string(api_version_minor))
            .c_str());
    LOG_MESSAGE(
        TRITONSERVER_LOG_INFO,
        (std::string("'") + name + "' TRITONBACKEND API version: " +
        std::to_string(TRITONBACKEND_API_VERSION_MAJOR) + "." +
        std::to_string(TRITONBACKEND_API_VERSION_MINOR)).c_str());
    if ((api_version_major != TRITONBACKEND_API_VERSION_MAJOR) ||
        (api_version_minor < TRITONBACKEND_API_VERSION_MINOR)) {
        return TRITONSERVER_ErrorNew(
            TRITONSERVER_ERROR_UNSUPPORTED,
            (std::string("Triton TRITONBACKEND API version: ") +
            std::to_string(api_version_major) + "." +
            std::to_string(api_version_minor) + " does not support '" + name +
            "' TRITONBACKEND API version: " +
            std::to_string(TRITONBACKEND_API_VERSION_MAJOR) + "." +
            std::to_string(TRITONBACKEND_API_VERSION_MINOR)).c_str());
    }
    return nullptr;  // success
}
```

步骤2 通过**接口B**对TRITONBACKEND_Model初始化：对triton::core传来的TritonModel（强转成了TRITONBACKEND_Model）进行初始化。主要创建一个ModelState并初始化，将其依附在TRITONBACKEND_Model上，核心代码如下：（其中ModelState包含了两个关键存储队列，一个是请求队列，另一个是推理任务队列，以供MindIE LLM接口循环调用。）

```
TRITONSERVER_Error* TRITONBACKEND_ModelInitialize(TRITONBACKEND_Model* model)
{
    // 创建一个ModelState类
    ModelState* model_state;
    RETURN_IF_ERROR(ModelState::Create(model, &model_state));
    // 将ModelState类依附在TRITONBACKEND_Model上
    RETURN_IF_ERROR(TRITONBACKEND_ModelSetState(model, reinterpret_cast<void*>(model_state)));
    return nullptr; // success
}
```

步骤3 通过**接口C**对TRITONBACKEND_ModelInstance初始化：对triton::core传来的TritonModelInstance（强转成了TRITONBACKEND_ModelInstance）进行初始化。主要创建一个ModelInstanceState并初始化，并将其依附在TRITONBACKEND_ModelInstance上，代码如下：（ModelInstanceState中定义了Enqueue成员函数，用来接收triton侧请求并转化为MindIE侧请求InferRequest，同时生成推理任务InferTask，最后一同写入ModelState内的存储队列中。）

```
TRITONSERVER_Error* TRITONBACKEND_ModelInstanceInitialize(TRITONBACKEND_ModelInstance*
instance)
{
    // 获取该模型实例属于的模型.
    TRITONBACKEND_Model* model;
    RETURN_IF_ERROR(TRITONBACKEND_ModelInstanceModel(instance, &model));
    // 获取依附在该模型实例上的ModelState.
    void* vmodelstate;
    RETURN_IF_ERROR(TRITONBACKEND_ModelState(model, &vmodelstate));
    ModelState* model_state = reinterpret_cast<ModelState*>(vmodelstate);
    // 创建一个ModelInstanceState并依附在模型实例上
    ModelInstanceState* instance_state;
    RETURN_IF_ERROR(ModelInstanceState::Create(model_state, instance, &instance_state));
    RETURN_IF_ERROR(TRITONBACKEND_ModelInstanceSetState(instance,
reinterpret_cast<void*>(instance_state)));
    return nullptr; // success
}
```


步骤4 通过**接口D**设置Backend接收triton侧请求的动作：首先从ModelInstance中提取出ModelInstanceState，接着调用Enqueue成员函数把所有请求放入ModelState中的请求队列requests_和推理任务哈希表inferTasksMap_，供LlmManager进行循环读取和推理，代码如下：

```
TRITONSERVER_Error* TRITONBACKEND_ModelInstanceExecute(
    TRITONBACKEND_ModelInstance* instance, TRITONBACKEND_Request** requests,
    const uint32_t request_count)
{
    // 获取模型实例状态
    ModelInstanceState* instance_state;
    RETURN_IF_ERROR(TRITONBACKEND_ModelInstanceState(instance,
reinterpret_cast<void**>(&instance_state)));
    LOG_MESSAGE(
      TRITONSERVER_LOG_INFO,
      (std::string("model instance ") + instance_state->Name() +
       ", executing " + std::to_string(request_count) + " requests")
          .c_str());
    // 将triton::core传来的请求经过解析、转换后存储进ModelState中的请求队列中
    instance_state->Enqueue(requests, request_count);
    return nullptr; // success
}

void ModelInstanceState::Enqueue(TRITONBACKEND_Request **requests, const uint32_t request_count)
{
    LOG_MESSAGE(
        TRITONSERVER_LOG_VERBOSE, (std::string("Process Requests: Executing
ModelInstanceState::Enqueue")).c_str());
    // 创建InferTask哈希表
    std::unordered_map<MindIE_LLM::InferRequestId, std::shared_ptr<triton::backend::mindie::InferTask>>
newInferTaskMap;
    std::vector<std::shared_ptr<MindIE_LLM::InferRequest>> newRequests;
    // 依次将每个来自triton::core的请求转化为mindie支持的请求类型，并记录下请求Id，以InferTask形式一同存
入哈希表中
    for (uint32_t i = 0; i < request_count; i++) {
        TRITONBACKEND_Request *bRequest = requests[i];
        auto inferTask = std::make_shared<InferTask>(bRequest);
        auto req = inferTask->GetMieRequest();
        auto requestId = req->GetRequestId();
        newInferTaskMap[requestId] = inferTask;
        newRequests.push_back(req);
    }
    // 将哈希表中所有的InferTask任务类一同存储进ModelState队列中，以供MindIE读取
    std::unique_lock lock(model_state_->GetMutex());
    model_state_->GetInferTasksMap().insert(newInferTaskMap.begin(), newInferTaskMap.end());
    for (uint32_t i = 0; i < newRequests.size(); i++) {
        model_state_->GetRequestsQueue().push(newRequests.at(i));
    }
}
```

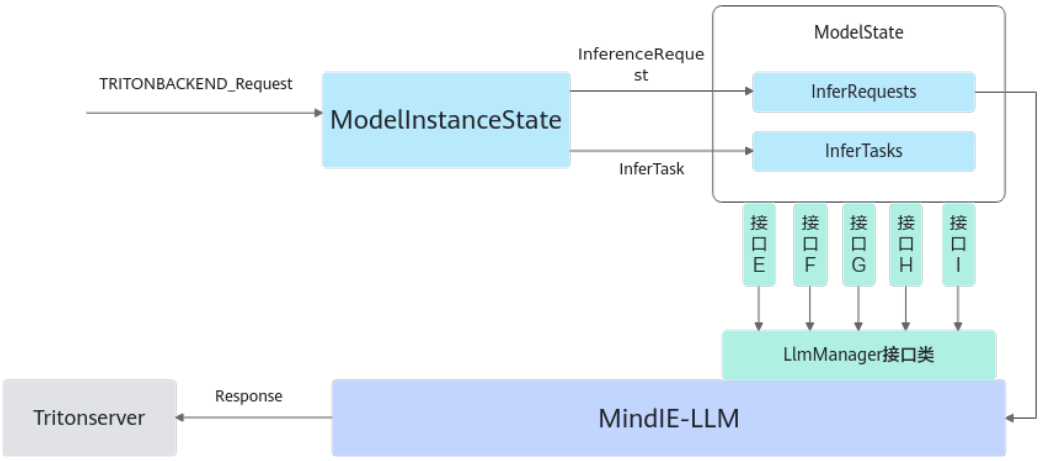
----结束

7.1.2.2 南向接口实现方式

实现方式

南向实现方式如图7-3所示，分别设计了五个核心接口E、F、G、H、I，实际上是五个回调函数，分别负责从ModelState存储队列中读取请求、将推理响应回传给Tritonserver、发送完成推理的请求ID队列、发送LlmManager推理状态和发送异常请求的状态。在ModelState构造函数内部定义并且传参给LlmManager完成初始化。LlmManager作为MindIE LLM与Backend对接的唯一核心接口类，提供模型推理和请求调度的能力，支持Continuous Batching动态调度。首先通过上层Backend传递的这五个回调函数和配置文件完成构造和初始化，接着创建线程来循环读取ModelState中接收到的请求队列，并传递给MindIE LLM进行处理。最后利用回调函数把响应Response直接传递给Tritonserver，并且在完成全部推理后释放请求任务。

图 7-3 南向实现方式



接口描述

表 7-9 接口 E

参数	描述
接口函数	std::function<std::vector<std::shared_ptr<mindie_llm::InferRequest>>()>。
接口功能	GetRequestsCallback回调函数，负责从ModelState存储队列InferRequests中读取出所有请求，将由LlmManager调用。
输入参数	N/A。
输出参数	推理请求队列。

表 7-10 接口 F

参数	描述
接口函数	std::function<void(mindie_llm::InferRequestId, const mindie_llm::TensorMap&, bool, const std::string&)>。
接口功能	SendResponsesCallback回调函数，负责将推理响应回传给Tritonserver，定义了MindIE LLM将推理出的单个Token响应回传给Tritonserver的行为，首先由Reqid对应的响应工厂responseFactory创建一个triton response，并使用convertResponse函数将tensorMap中所有推理张量依次附加至该response中，最后使用TRITONBACKEND_ResponseSend函数将该响应回传给Triton侧。值得一提的是，如果参数isEnd为true，代表该请求的推理全部完成，需要在回调函数中释放该请求。
输入参数	请求Id，包含推理响应信息的张量，end标志位，响应自身的状态。

参数	描述
输出参数	N/A。

表 7-11 接口 G

参数	描述
接口函数	std::function<std::vector<std::pair<mindie_llm::InferRequestId, mindie_llm::Operation>>()>。
接口功能	ControlSignalCallback回调函数，发送停止推理的请求id队列。
输入参数	N/A。
输出参数	完成推理的请求id队列。

表 7-12 接口 H

参数	描述
接口函数	std::function<void(const std::string&)>。
接口功能	LlmManagerStatsCallback回调函数，发送LlmManager推理状态，包括slaves_status、remain_blocks、remain_prefill_slots等。
输入参数	LlmManager推理状态status字符串。
输出参数	N/A。

表 7-13 接口 I

参数	描述
接口函数	std::function<void(mindie_llm::InferRequestId, mindie_llm::Status, mindie_llm::StatusResponseType)>。
接口功能	SendStatuslResponseCallback回调函数，发送异常请求的状态。
输入参数	请求Id，请求状态，响应状态。
输出参数	N/A。

表 7-14 LlmManager 接口类

参数	描述
接口类构造函数	LlmManager::LlmManager(const std::string& llmConfigPath, GetRequestsCallback getRequest, SendResponsesCallback sendResponse, ControlSignalCallback controlCallback, LlmManagerStatsCallback statusCallback, SendStatuslResponseCallback statusResponseCallback)。
接口类功能	LlmManager作为MindIE LLM暴露给backend的唯一核心接口类，提供模型推理和请求调度的能力，支持Continuous Batching动态调度。首先通过上层Backend传递的这五个回调函数和配置文件完成初始化，在构造函数中创建了一个线程循环读取ModelState中接收到的请求队列，并传递给MindIE LLM进行处理。最后利用回调函数把响应response直接传递给Tritonserver，并且在完成全部推理后摧毁请求任务。
输入参数	模型配置文件，五个ModelState中定义的回调函数。
输出参数	N/A。

适配流程

ModelState初始化，在构造函数中主要做以下三点：

1. 初始化BackendModel并将配置文件engineConfigPath透传给MindIE LLM，其中包含了模型推理所需的配置信息和调度策略等。
2. 定义五个接口回调函数，是Triton和MindIE LLM之间收发请求和响应的关键。定义了从ModelState存储队列中读取请求的动作、将推理响应回传给Tritonserver的动作、发送完成推理的请求ID队列的动作、发送LlmManager推理状态的动作及发送异常请求的状态的动作。
3. 使用engineConfigPath和五个Callback函数给每个模型实例构造出LlmManager对象，该对象在初始化后创建了线程循环读取ModelState中接收到的请求队列，并传递给LLM进行推理和调度，代码如下：

```
ModelState::ModelState(TRITONBACKEND_Model *triton_model) : BackendModel(triton_model),
shape_initialized_(false)
{
    // 验证模型的配置参数都被backend支持
    THROW_IF_BACKEND_MODEL_ERROR(ValidateModelConfig());
    // 获取mindie配置文件路径
    std::string configPath = MindIEUtils::GetEngineConfigPath(this);
    // 获取llm-manager初始化时必须的模型实例个数和对应的npu设备号
    auto initConfig = GetInitConfig();
    uint32_t modelInstanceNumber = initConfig.first;
    std::vector<std::set<size_t>> npuDeviceIds = initConfig.second;
    LOG_MESSAGE(TRITONSERVER_LOG_ERROR,
                (std::string("modelInstanceNumber: ") + std::to_string(modelInstanceNumber)).c_str());
    // mindie-llm从modelstate存储队列中获取request的回调函数
    auto getRequestsFunc = CreateGetRequestsCallback();
    // mindie-llm推理出的响应发送回tritonserver的回调函数
    auto sendResponsesFunc = CreateSendResponsesCallback();

    // mindie-llm接收triton侧发来的停止推理请求Id的回调函数
    auto controlSignalFunc = CreateControlSignalCallback();

    // mindie-llm发送推理状态的回调函数
    auto llmManagerStatsFunc = CreateLlmManagerStatsCallback();
```



```
    // mindie-llm发送推理过程中异常请求的状态
    auto sendStatusResponseCallback = CreateSendStatusResponseCallback();

    // 给每个模型实例构造并初始化一个llm_manager接口类，负责mindie中请求的调度推理
    mindie_llm::LlmManager* llm_manager = nullptr;
    for (uint32_t modelInstanceId = 0; modelInstanceId < modelInstanceNumber; ++modelInstanceId) {
        llm_manager = new mindie_llm::LlmManager(configPath,
            getRequestsFunc, sendResponsesFunc, controlSignalFunc, llmManagerStatsFunc,
sendStatusResponseCallback);
        llmManagers_.emplace_back(llm_manager);
        auto status = llm_manager->Init(modelInstanceId, npuDeviceIds[modelInstanceId]);
        if (!status.IsOk()) {
            LOG_MESSAGE(TRITONSERVER_LOG_ERROR,
                "Failed to init llm_manager");
            break;
        }
    }
}
```

7.1.2.3 环境安装与启动服务

前提条件

- 已参见《MindIE安装指南》中“**安装驱动和固件**”章节完成驱动和固件的安装。
- 已参见《MindIE安装指南》中“**安装开发环境**”章节完成CANN、Python 3.10.2、PyTorch 2.1.0框架和Torch_NPU 2.1.0插件的安装。
- 已参见《MindIE安装指南》中“**物理机部署MindIE**”章节完成MindIE的安装。

安装步骤

步骤1 安装模型仓：

HDK、CANN、PTA、ATB Models请参考模型仓**readme**。

步骤2 安装MindIE LLM：

获取MindIE的整包进行安装：

软件包名称：

- Ascend-mindie_1.0.RC3_linux-x86_64.run
- Ascend-mindie_1.0.RC3_linux-aarch64.run

安装MindIE整包代码：

```
# 安装MindIE整包
chmod +x Ascend-mindie*.run ./Ascend-mindie-{version}_linux-{arch}.run --install --install-path=$
{working_dir}
source ${working_dir}/mindie/set_env.sh
```

步骤3 安装Triton Inference Server：

须知

gcc版本需支持c++17。

- 下载server代码并重命名为triton_server。
  ```
  git clone https://github.com/triton-inference-server/server.git triton_server
  ```


- 切换分支到r24.02。
```
cd triton_server
git checkout r24.02
```
- 运行启动脚本。
  a. 复制以下代码并修改对应路径(15和16行)，得到一个.sh文件，本文命名为build_npu.sh。
  b. 将其放在triton_server的根目录下。
  c. 执行 bash build_npu.sh。
```
#!/bin/bash
build_type="Release"
while getopts "d" opt; do
  case ${opt} in
    d)
      build_type="Debug"
      ;;
    \?)
      echo "Invalid option: -$OPTARG" >&2
      exit 1
      ;;
  esac
done

mkdir -p /home/xxx/triton_server/build
cd /home/xxx/triton_server/build

cmake -DCMAKE_BUILD_TYPE=$build_type -DCMAKE_INSTALL_PREFIX:PATH=/opt/tritonserver \
        -DTRITON_THIRD_PARTY_REPO_TAG=r24.02 \
        -DTRITON_COMMON_REPO_TAG=r24.02  \
        -DTRITON_CORE_REPO_TAG=r24.02  \
        -DTRITON_BACKEND_REPO_TAG=r24.02  \
        -DTRITON_VERSION=1.0.0 \
        -DTRITON_ENABLE_METRICS_GPU=OFF \
        -DTRITON_ENABLE_GPU=OFF \
        -DTRITON_CORE_HEADERS_ONLY=OFF \
        ..

make install
```
- 编译过程中如果出现报错有包没有安装，请按照提示安装即可。
```
apt-get install rapidjson-dev # rapidjson
apt-get install libboost-dev # boost
apt-get install libre2-dev # Re2
apt-get install libb64-dev # b64
apt-get install libnuma-dev # numa
apt-get install patchelf # patchelf
```
- 编译过程中大概率会出现grpc无法下载，导致编译失败的情况。解决如下：（以24.02为例）
  a. 手动git clone grpc代码。
```
# 注：git clone前可以将curl的postBuffer设大，笔者配置为20G，按需分配，运行如下指令：
# git config --global http.postBuffer 20971520000

cd /home/xxx/triton_server/build/_deps/repo-third-party-build/grpc-repo/src
git clone --branch v1.48.0 https://github.com/grpc/grpc.git // 使用哪个版本需要看triton_server/build/_deps/repo-third-party-src/CMakeLists.txt中grpc的git tag
cd grpc
git submodule update --init --recursive // 下载git子仓库
```
  b. 下载git子仓库时可能会有部分子仓库无法下载，需要手动进行。
     - 进入到/triton_server/build/_deps/repo-third-party-build/grpc-repo/src/grpc/third_part。
     - git clone 上一步没有clone成功的子仓，需要将boringssl的名字换成boringssl-with-bazel。

- 执行 git submodule init && git submodule update，如果都已经下载下来了，屏幕上会打印子仓的路径及节点号，比如 Submodule path 'third_party/googletest': checked out 'c9ccac7cb7345901884aabf5d1a786c。

```
cd /home/xxx/triton_server/build/_deps/repo-third-party-build/grpc-repo/src/grpc/
third_part
git clone xxx // 上一步没有git clone下来的子仓
cd ..
git submodule init
git submodule update
```

c. 修改对应的CMakelist.txt，让其指向本地文件。

```
cd /home/xxx/triton_server/build/_deps/repo-third-party-src/
vim CMakeLists.txt
```

d. 屏蔽对应的git源，同理，其他git源出现上述问题也同样处理。

```
ExternalProject_Add(grpc-repo
  PREFIX grpc-repo
  # GIT_REPOSITORY "https://gitee.com/Gongen/grpc.git"
  # GIT_TAG "v1.48.0"
  SOURCE_DIR "${CMAKE_CURRENT_BINARY_DIR}/grpc-repo/src/grpc"
  EXCLUDE_FROM_ALL ON
  CONFIGURE_COMMAND ""
  BUILD_COMMAND ""
  INSTALL_COMMAND ""
  TEST_COMMAND ""
  PATCH_COMMAND python3 ${CMAKE_CURRENT_SOURCE_DIR}/tools/install_src.py --src
<SOURCE_DIR> ${INSTALL_SRC_DEST_ARG} --dest-basename=grpc_1.48.0
)
```

e. 编译过程中如果出现boost版本过低的报错提示，运行如下指令安装boost-1.78即可。

```
wget https://boostorg.jfrog.io/artifactory/main/release/1.78.0/source/boost_1_78_0.tar.gz
tar xvf boost_1_78_0.tar.gz
cd boost_1_78_0
./bootstrap.sh --with-python=/usr/local/bin/python3
./b2 install
```

步骤4 安装Triton Client：

```
pip install nvidia-pyindex
pip install tritonclient[all]
```

步骤5 安装MindIE Backend：

- 方法一： 使用已发布的libtriton_mindie.so文件。

  在 /opt/tritonserver/backends 目录下创建mindie目录，将libtriton_mindie.so放在该目录下。

- 方法二：使用编译出来的libtriton_mindie.so文件。

  a. 参考**样例代码**中的目录结构搭建Triton_MindIE-LLM_Backend代码仓。

  b. 设置环境变量。

```
export MINDIE_LLM_HOME_PATH=${mindie_dir}/latest/mindie-llm  #mindie-llm安装路径
export TRITON_HOME_PATH=/opt/tritonserver  #tritonserver安装路径
```

  c. 编译。

```
cd ${working_dir}/Triton_MindIE-LLM_Backend
bash build.sh
```

----结束

启动服务

步骤1 启动Triton Server：

```
// 设置环境变量
cd ${working_dir}/Triton_MindIE-LLM_Backend/example
```



```
source set_env.sh

// 启动triton服务
bash launch.sh
```

当出现如下打印时，表示启动成功。

```
I0807 03:28:54.018305 1434372 grpc_server.cc:2519] Started GRPCInferenceService at 0.0.0.0:8111
I0807 03:28:54.018602 1434372 http_server.cc:4637] Started HTTPService at 0.0.0.0:8110
I0807 03:28:54.060484 1434372 http_server.cc:320] Started Metrics Service at 0.0.0.0:8112
```

步骤2 用Triton Client发送测试请求：

- 另启动一个窗口，进入和Triton server一样的容器，source与上文同样的环境变量。
- 如果运行单用例，修改client_stream.py中的模型名和权重路径，然后执行如下命令。
  ```
  python client_stream.py
  ```
- 如果跑多用例数据集，执行如下命令。
  ```
  python submit.py --name llama3_8b --model_path weights/llama3_8b --trace_dataset GSM8K.jsonl
  ```

须知

必须配置的参数：

- --name 模型名
- --model_path 模型权重路径
- --trace_dataset 数据集名

----结束

7.1.3 样例代码

代码目录结构如下：

```
Triton_MindIE-LLM_Backend
|____build.sh
|____CMakeLists.txt
|____src
|    |____mindie.cc
|    |____model_state.cc
|    |____model_state.h
|    |____model_instance_state.cc
|    |____model_instance_state.h
|    |____mindie_utils.cc
|    |____mindie_utils.h
|    |____infer_task.cc
|    |____infer_task.h
|____example
|    |____tritonModels
|    |    |____llama3_8b
|    |    |    |____1
|    |    |    |____config.pbtxt
|    |    |____config.json
|    |____set_env.sh
|    |____launch.sh
|    |____submit.py
|    |____client_stream.py
|    |____infer_client.py
|    |____logger_util.py
|    |____GSM8K.jsonl
```


src源文件和example文件夹的含义和作用分别如表7-15和表7-16所示。

表 7-15 src 源文件的含义和作用

源文件	含义及作用
mindie.cc	包含了Triton提供给Backend的核心接口函数，分别是TRITONBACKEND_Backend初始化、TRITONBACKEND_Model初始化和释放、TRITONBACKEND_ModelInstance初始化和释放以及模型实例接收并执行请求推理的接口设置。
model_state.cc	模型状态类的源文件，在其构造函数中使用engineConfigPath和五个callback函数给每个模型实例构造出一个LlmManager对象，完成初始化后创建线程从ModelState存储队列中读取出所有请求。
model_instance_state.cc	模型实例状态类的源文件，核心是其中定义的Enqueue成员函数，将所有来自triton::core侧的待推理请求解析并转换成MindIE侧请求类型，最后存储进ModelState队列中以待MindIE读取。
mindie_utils.cc	定义了几个MindIE工具函数，一个是将响应从MindIE侧转化为Triton侧类型的转换函数，其他几个是从文件中提取出配置和参数信息的解析函数。
infer_task.cc	推理任务类的源文件，核心是构造函数中将Triton侧请求转化为MindIE侧类型，并给每个推理请求创建了一个响应工厂，用来在发送响应回调函数中创建Triton侧响应。

表 7-16 example 文件的含义和作用

源文件	描述
tritonModels	Triton管理的模型仓，里面存放了LLaMa3_8b模型的Triton侧配置文件config.pbtxt和MindIE侧配置文件config.json。
set_env.sh	环境变量配置文件，根据CANN、ATB Models、MindIE安装路径按需修改。
launch.sh	启动Tritonserver服务端，其中包含MindIE LLM和Triton日志等级配置。
submit.py	利用tritonClinet向服务端依次发送GSM8K.jsonl数据集中的请求。
client_stream.py	利用tritonClinet向服务端发送单个请求。
GSM8K.jsonl	推理数据集，这里以GSM8K为例，需用户自行提供。

样例代码：

- src/mindie.cc

```
// Copyright (c) 2019-2020, NVIDIA CORPORATION. All rights reserved.
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
```

```
// are met:
//  * Redistributions of source code must retain the above copyright
//    notice, this list of conditions and the following disclaimer.
//  * Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
//  * Neither the name of NVIDIA CORPORATION nor the names of its
//    contributors may be used to endorse or promote products derived
//    from this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ``AS IS'' AND ANY
// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
// PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
// CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
// PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
// OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
// OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#include <stdint.h>
#include <string>
#include "model_state.h"
#include "model_instance_state.h"
#include "triton/backend/backend_common.h"
namespace triton { namespace backend { namespace mindie {
//
// ModelState
//
//////////////
extern "C" {
TRITONSERVER_Error* TRITONBACKEND_Initialize(TRITONBACKEND_Backend* backend)
{
    const char* cname;
    RETURN_IF_ERROR(TRITONBACKEND_BackendName(backend, &cname));
    std::string name(cname);
    LOG_MESSAGE(
        TRITONSERVER_LOG_INFO,
        (std::string("TRITONBACKEND_Initialize: ") + name).c_str());

    uint32_t api_version_major, api_version_minor;
    RETURN_IF_ERROR(
        TRITONBACKEND_ApiVersion(&api_version_major, &api_version_minor));
    LOG_MESSAGE(
        TRITONSERVER_LOG_INFO,
        (std::string("Triton TRITONBACKEND API version: ") +
        std::to_string(api_version_major) + "." +
        std::to_string(api_version_minor))
            .c_str());
    LOG_MESSAGE(
        TRITONSERVER_LOG_INFO,
        (std::string("'") + name + "' TRITONBACKEND API version: " +
        std::to_string(TRITONBACKEND_API_VERSION_MAJOR) + "." +
        std::to_string(TRITONBACKEND_API_VERSION_MINOR)).c_str());
    if ((api_version_major != TRITONBACKEND_API_VERSION_MAJOR) ||
        (api_version_minor < TRITONBACKEND_API_VERSION_MINOR)) {
        return TRITONSERVER_ErrorNew(
            TRITONSERVER_ERROR_UNSUPPORTED,
            (std::string("Triton TRITONBACKEND API version: ") +
            std::to_string(api_version_major) + "." +
            std::to_string(api_version_minor) + " does not support '" + name +
            "' TRITONBACKEND API version: " +
            std::to_string(TRITONBACKEND_API_VERSION_MAJOR) + "." +
            std::to_string(TRITONBACKEND_API_VERSION_MINOR)).c_str());
    }
    return nullptr;  // success
}
TRITONSERVER_Error* TRITONBACKEND_ModelInitialize(TRITONBACKEND_Model* model)
```



```
{
    // Create a ModelState object and associate it with the
    // TRITONBACKEND_Model. If anything goes wrong with initialization
    // of the model state then an error is returned and Triton will fail
    // to load the model.
    ModelState* model_state;
    RETURN_IF_ERROR(ModelState::Create(model, &model_state));
    RETURN_IF_ERROR(TRITONBACKEND_ModelSetState(model,
reinterpret_cast<void*>(model_state)));
    return nullptr; // success
}
TRITONSERVER_Error* TRITONBACKEND_ModelFinalize(TRITONBACKEND_Model* model)
{
    void* vstate;
    RETURN_IF_ERROR(TRITONBACKEND_ModelState(model, &vstate));
    ModelState* model_state = reinterpret_cast<ModelState*>(vstate);
    delete model_state;
    return nullptr; // success
}
TRITONSERVER_Error* TRITONBACKEND_ModelInstanceInitialize(TRITONBACKEND_ModelInstance*
instance)
{
    // Get the model state associated with this instance's model.
    TRITONBACKEND_Model* model;
    RETURN_IF_ERROR(TRITONBACKEND_ModelInstanceModel(instance, &model));
    void* vmodelstate;
    RETURN_IF_ERROR(TRITONBACKEND_ModelState(model, &vmodelstate));
    ModelState* model_state = reinterpret_cast<ModelState*>(vmodelstate);
    // Create a ModelInstanceState object and associate it with the
    // TRITONBACKEND_ModelInstance.
    ModelInstanceState* instance_state;
    RETURN_IF_ERROR(ModelInstanceState::Create(model_state, instance, &instance_state));
    RETURN_IF_ERROR(TRITONBACKEND_ModelInstanceSetState(instance,
reinterpret_cast<void*>(instance_state)));
    return nullptr; // success
}
TRITONSERVER_Error* TRITONBACKEND_ModelInstanceFinalize(TRITONBACKEND_ModelInstance*
instance)
{
    void* vstate;
    RETURN_IF_ERROR(TRITONBACKEND_ModelInstanceState(instance, &vstate));
    ModelInstanceState* instance_state = reinterpret_cast<ModelInstanceState*>(vstate);
    delete instance_state;
    return nullptr; // success
}
TRITONSERVER_Error* TRITONBACKEND_ModelInstanceExecute(
    TRITONBACKEND_ModelInstance* instance, TRITONBACKEND_Request** requests,
    const uint32_t request_count)
{
    ModelInstanceState* instance_state;
    RETURN_IF_ERROR(TRITONBACKEND_ModelInstanceState(instance,
reinterpret_cast<void**>(&instance_state)));
    LOG_MESSAGE(
      TRITONSERVER_LOG_INFO,
      (std::string("model instance ") + instance_state->Name() +
       ", executing " + std::to_string(request_count) + " requests")
          .c_str());
    instance_state->Enqueue(requests, request_count);
    return nullptr; // success
}
}  // extern "C"
}}} // namespace triton::backend::mindie
```

- src/model_state.h

```
// Copyright 2021, NVIDIA CORPORATION & AFFILIATES. All rights reserved.
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
//  * Redistributions of source code must retain the above copyright
```

```
//    notice, this list of conditions and the following disclr.
//  * Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
//  * Neither the name of NVIDIA CORPORATION nor the names of its
//    contributors may be used to endorse or promote products derived
//    from this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ``AS IS'' AND ANY
// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
// PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
// CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
// PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
// OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
// OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#ifndef MINDIE_STATE_H
#define MINDIE_STATE_H
//
// ModelState
//
// State associated with a model that is using this backend. An object
// of this class is created and associated with each
// TRITONBACKEND_Model. ModelState is derived from BackendModel class
// provided in the backend utilities that provides many common
// functions.
//
#include <string>
#include <thread>
#include <queue>
#include <unordered_map>
#include <set>
#include <mutex>
#include <atomic>
#include <functional>
#include <memory>
#include <shared_mutex>
#include "infer_task.h"
#include "triton/backend/backend_model.h"
#include "triton/backend/backend_common.h"
#include "triton/core/tritonbackend.h"
#include "llm_manager/llm_manager.h"

namespace triton::backend::mindie {
class ModelState : public BackendModel {
public:
    static TRITONSERVER_Error *Create(TRITONBACKEND_Model *triton_model, ModelState **state);
    virtual ~ModelState();
    // Name of the input and output tensor
    const std::string &InputTensorName() const
    {
        return input_name_;
    }
    const std::string &OutputTensorName() const
    {
        return output_name_;
    }
    // Datatype of the input and output tensor
    TRITONSERVER_DataType TensorDataType() const
    {
        return datatype_;
    }
    // Shape of the input and output tensor as given in the model
    // configuration file. This shape will not include the batch
    // dimension (if the model has one).
    const std::vector<int64_t> &TensorNonBatchShape() const
```

```
    {
        return nb_shape_;
    }
    // Validate that this model is supported by this backend.
    TRITONSERVER_Error *ValidateModelConfig();
    std::queue<std::shared_ptr<mindie_llm::InferRequest>> &GetRequestsQueue();
    std::unordered_map<mindie_llm::InferRequestId, std::shared_ptr<InferTask>> &GetInferTasksMap();
    std::shared_mutex &GetMutex();
    std::vector<mindie_llm::LlmManager*> GetLlmManagers();
private:
    ModelState(TRITONBACKEND_Model *triton_model);
    mindie_llm::GetRequestsCallback CreateGetRequestsCallback();
    mindie_llm::SendResponsesCallback CreateSendResponsesCallback();
    mindie_llm::ControlSignalCallback CreateControlSignalCallback();
    mindie_llm::LlmManagerStatsCallback CreateLlmManagerStatsCallback();
    mindie_llm::SendStatusResponseCallback CreateSendStatusResponseCallback();
    std::pair<uint32_t, std::vector<std::set<size_t>>> GetInitConfig();
    std::string input_name_;
    std::string output_name_;
    TRITONSERVER_DataType datatype_;
    bool shape_initialized_;
    std::vector<int64_t> nb_shape_;
    std::vector<int64_t> shape_;
    std::queue<std::shared_ptr<mindie_llm::InferRequest>> requests_;
    std::unordered_map<mindie_llm::InferRequestId, std::shared_ptr<InferTask>> inferTasksMap_;
    std::shared_mutex mutex_;
    std::vector<mindie_llm::LlmManager*> llmManagers_;
};
} // namespace triton::backend::mindie
#endif
```

- src/model_state.cc

```
// Copyright 2021, NVIDIA CORPORATION & AFFILIATES. All rights reserved.
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
//  * Redistributions of source code must retain the above copyright
//    notice, this list of conditions and the following disclaimer.
//  * Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
//  * Neither the name of NVIDIA CORPORATION nor the names of its
//    contributors may be used to endorse or promote products derived
//    from this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ``AS IS'' AND ANY
// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
// PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
// CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
// PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
// OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
// OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

#include <unordered_set>
#include "mindie_utils.h"
#include "model_state.h"

namespace triton::backend::mindie {
ModelState::ModelState(TRITONBACKEND_Model *triton_model) : BackendModel(triton_model, true),
shape_initialized_(false)
{
    // Validate that the model's configuration matches what is supported
    // by this backend.
    THROW_IF_BACKEND_MODEL_ERROR(ValidateModelConfig());
    std::string configPath = MindIEUtils::GetEngineConfigPath(this);
```

```
    LOG_MESSAGE(TRITONSERVER_LOG_ERROR,
                (std::string("configPath: ") + configPath).c_str());
    auto initConfig = GetInitConfig();
    uint32_t modelInstanceNumber = initConfig.first;
    std::vector<std::set<size_t>> npuDeviceIds = initConfig.second;
    LOG_MESSAGE(TRITONSERVER_LOG_ERROR,
                (std::string("modelInstanceNumber: ") + std::to_string(modelInstanceNumber)).c_str());
    auto getRequestsFunc = CreateGetRequestsCallback();
    auto sendResponsesFunc = CreateSendResponsesCallback();
    auto controlSignalFunc = CreateControlSignalCallback();
    auto llmManagerStatsFunc = CreateLlmManagerStatsCallback();
    auto sendStatusResponseCallback = CreateSendStatusResponseCallback();
    mindie_llm::LlmManager* llm_manager = nullptr;
    for (uint32_t modelInstanceId = 0; modelInstanceId < modelInstanceNumber; ++modelInstanceId) {
        llm_manager = new mindie_llm::LlmManager(configPath,
            getRequestsFunc, sendResponsesFunc, controlSignalFunc, llmManagerStatsFunc,
sendStatusResponseCallback);
        llmManagers_.emplace_back(llm_manager);
        auto status = llm_manager->Init(modelInstanceId, npuDeviceIds[modelInstanceId]);
        if (!status.IsOk()) {
            LOG_MESSAGE(TRITONSERVER_LOG_ERROR,
                "Failed to init llm_manager");
            break;
        }
    }
}
mindie_llm::GetRequestsCallback ModelState::CreateGetRequestsCallback()
{
    return [this]() -> std::vector<std::shared_ptr<mindie_llm::InferRequest>> {
        std::vector<std::shared_ptr<mindie_llm::InferRequest>> newRequests;
        std::unique_lock lock(this->mutex_);
        if (this->requests_.empty()) {
            return newRequests;
        }
        while (!this->requests_.empty()) {
            auto req = this->requests_.front();
            auto requestId = req->GetRequestId();
            newRequests.push_back(this->requests_.front());
            this->requests_.pop();
        }
        return newRequests;
    };
}
mindie_llm::SendResponsesCallback ModelState::CreateSendResponsesCallback()
{
    return [this](mindie_llm::InferRequestId reqId,
                  const mindie_llm::TensorMap &tensorMap,
                  bool isEnd,
                  const std::string &flag) {
            auto item = this->inferTasksMap_.find(reqId);
            if (item == this->inferTasksMap_.end()) {
                LOG_MESSAGE(TRITONSERVER_LOG_ERROR,
                    (std::string("can't find infertask with reqId: ") + reqId.StringValue()).c_str());
                return;
            }
            std::shared_ptr<InferTask> inferTask = item->second;
            TRITONBACKEND_ResponseFactory *responseFactory = inferTask->GetResponseFactory();
            if (responseFactory == nullptr) {
                LOG_MESSAGE(
                    TRITONSERVER_LOG_ERROR, std::string("MindIE inferTask->GetResponseFactory() is
nullptr").c_str());
                return;
            }
            TRITONBACKEND_Response *bResponse;
            auto tritonErr = TRITONBACKEND_ResponseNewFromFactory(&bResponse, responseFactory);
            if (tritonErr != nullptr) {
                LOG_MESSAGE(TRITONSERVER_LOG_ERROR,
                    (std::string("ResponseNewFromFactory failed, reqId: ") + reqId.StringValue()).c_str());
                return;
```



```
            }
            tritonErr = MindIEUtils::ConvertResponse(bResponse, tensorMap);
            if (tritonErr != nullptr) {
                LOG_IF_ERROR(tritonErr, std::string("generate triton response failed, reqId: ") +
reqId.StringValue());
                return;
            }
            tritonErr = TRITONBACKEND_ResponseSend(bResponse,
                isEnd ? TRITONSERVER_RESPONSE_COMPLETE_FINAL : 0,
                tritonErr);
            if (isEnd) {
                TRITONBACKEND_RequestRelease(inferTask->GetBackendRequest(),
TRITONSERVER_REQUEST_RELEASE_ALL);
                std::unique_lock lock(this->GetMutex());
                this->inferTasksMap_.erase(item);
            }
            if (tritonErr != nullptr) {
                LOG_IF_ERROR(tritonErr, std::string("send triton response failed, reqId: ") +
reqId.StringValue());
                return;
            }
        };
}
mindie_llm::ControlSignalCallback ModelState::CreateControlSignalCallback()
{
    return [this]() -> std::vector<std::pair<mindie_llm::InferRequestId, mindie_llm::Operation>> {
        std::vector<std::pair<mindie_llm::InferRequestId, mindie_llm::Operation>> stopIds;
        return stopIds;
    };
}
mindie_llm::LlmManagerStatsCallback ModelState::CreateLlmManagerStatsCallback()
{
    return [this](const std::string &strData) {
        return;
    };
}
mindie_llm::SendStatusResponseCallback ModelState::CreateSendStatusResponseCallback()
{
    return [this](mindie_llm::InferRequestId requestId,
                  mindie_llm::Status status,
                  mindie_llm::StatusResponseType responsetype) {
        if (!(status == mindie_llm::Status())) {
            LOG_MESSAGE(TRITONSERVER_LOG_ERROR,
                        (std::string("Failed to stop request ") + requestId.StringValue() +
                         ", Error msg: " + status.StatusMsg() + ", responsetype: ").c_str());
        }
    };
}
ModelState::~ModelState()
{
    if (!llmManagers_.empty()) {
        for (auto& llm_manager:llmManagers_) {
            llm_manager->Shutdown();
            delete llm_manager;
            llm_manager = nullptr;
        }
        llmManagers_.clear();
    }
}
TRITONSERVER_Error *ModelState::Create(TRITONBACKEND_Model *triton_model, ModelState **state)
{
    LOG_MESSAGE(TRITONSERVER_LOG_VERBOSE, (std::string("Create MindIE ModelState:")).c_str());
    try {
        *state = new ModelState(triton_model);
    } catch (const BackendModelException &ex) {
        RETURN_ERROR_IF_TRUE(ex.err_ == nullptr,
            TRITONSERVER_ERROR_INTERNAL,
            std::string("unexpected nullptr in BackendModelException"));
        RETURN_IF_ERROR(ex.err_);
```



```
    }
    return nullptr; // success
}
std::pair<uint32_t, std::vector<std::set<size_t>>> ModelState::GetInitConfig()
{
    triton::common::TritonJson::Value parameters;
    TRITONSERVER_Error* err = this->ModelConfig().MemberAsObject("parameters", &parameters);
    if (err != nullptr) {
        TRITONSERVER_ErrorDelete(err);
        throw std::runtime_error("Model config doesn't have a parameters section");
    }
    std::string modelInstanceNumber_s;
    LOG_IF_ERROR(
        GetParameterValue(parameters, "model_instance_number", &modelInstanceNumber_s),
            "failed to get modelInstanceNumber config.");
    std::string npuDeviceIds_s;
    LOG_IF_ERROR(
        GetParameterValue(parameters, "npu_device_ids", &npuDeviceIds_s),
            "failed to get npuDeviceIds config.");
    // 0 : number of modelInstances
    uint32_t modelInstanceNumber = static_cast<uint32_t>(modelInstanceNumber_s[0] - '0');
    std::vector<std::set<size_t>> npuDeviceIds;
    std::set<size_t> npuDeviceIds_instance;
    for (const auto& id: npuDeviceIds_s) {
        if (id == ';') {
            npuDeviceIds.emplace_back(npuDeviceIds_instance);
            npuDeviceIds_instance.clear();
            continue;
        }
        size_t npuId = static_cast<size_t>(id - '0');
        npuDeviceIds_instance.emplace(npuId);
    }
    auto initConfig = std::pair(modelInstanceNumber, npuDeviceIds);
    return initConfig;
}
TRITONSERVER_Error *ModelState::ValidateModelConfig()
{
    return nullptr;  // success
}
std::queue<std::shared_ptr<mindie_llm::InferRequest>> &ModelState::GetRequestsQueue()
{
    return requests_;
}
std::unordered_map<mindie_llm::InferRequestId, std::shared_ptr<InferTask>>
&ModelState::GetInferTasksMap()
{
    return inferTasksMap_;
}
std::shared_mutex &ModelState::GetMutex()
{
    return mutex_;
}
std::vector<mindie_llm::LlmManager*> ModelState::GetLlmManagers()
{
    return llmManagers_;
}
} // namespace triton::backend::mindie
```

- src/model_instance_state.h

```
// Copyright 2021, NVIDIA CORPORATION & AFFILIATES. All rights reserved.
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
//  * Redistributions of source code must retain the above copyright
//    notice, this list of conditions and the following disclaimer.
//  * Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
//  * Neither the name of NVIDIA CORPORATION nor the names of its
```

```
//    contributors may be used to endorse or promote products derived
//    from this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ``AS IS'' AND ANY
// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
// PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
// CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
// PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
// OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
// OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#ifndef MINDIE_INSTANCE_H
#define MINDIE_INSTANCE_H
//
// ModelInstanceState
//
// State associated with a model instance. An object of this class is
// created and associated with each
// TRITONBACKEND_ModelInstance. ModelInstanceState is derived from
// BackendModelInstance class provided in the backend utilities that
// provides many common functions.
//
#include "triton/backend/backend_common.h"
#include "triton/backend/backend_model_instance.h"
#include "triton/backend/backend_model.h"
#include "triton/core/tritonbackend.h"
#include <string>
#include "model_state.h"
#include "infer_task.h"
#include "mindie_utils.h"
namespace triton::backend::mindie {
class ModelInstanceState : public BackendModelInstance {
public:
    static TRITONSERVER_Error *Create(
        ModelState *model_state, TRITONBACKEND_ModelInstance *triton_model_instance,
ModelInstanceState **state);
    virtual ~ModelInstanceState() = default;
    // Get the state of the model that corresponds to this instance.
    ModelState *StateForModel() const
    {
        return model_state_;
    }
    void Enqueue(TRITONBACKEND_Request **requests, const uint32_t request_count);
private:
    ModelInstanceState(ModelState *model_state, TRITONBACKEND_ModelInstance
*triton_model_instance);
    ModelState *model_state_;
};
}  // namespace triton::backend::mindie
#endif
```

- src/model_instance_state.cc

```
// Copyright (c) 2022, NVIDIA CORPORATION & AFFILIATES. All rights reserved.
// Copyright 2019-2022, NVIDIA CORPORATION & AFFILIATES. All rights reserved.
// Copyright 2021, NVIDIA CORPORATION & AFFILIATES. All rights reserved.
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
//  * Redistributions of source code must retain the above copyright
//    notice, this list of conditions and the following disclaimer.
//  * Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
//  * Neither the name of NVIDIA CORPORATION nor the names of its
//    contributors may be used to endorse or promote products derived
//    from this software without specific prior written permission.
```

```
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ``AS IS'' AND ANY
// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
// PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
// CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
// PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
// OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
// OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#include "model_instance_state.h"
#include <cmath>

namespace triton::backend::mindie {
TRITONSERVER_Error *ModelInstanceState::Create(
    ModelState *model_state, TRITONBACKEND_ModelInstance *triton_model_instance,
ModelInstanceState **state)
{
    try {
        *state = new ModelInstanceState(model_state, triton_model_instance);
    } catch (const BackendModelInstanceException &ex) {
        RETURN_ERROR_IF_TRUE(ex.err_ == nullptr,
            TRITONSERVER_ERROR_INTERNAL,
            std::string("unexpected nullptr in BackendModelInstanceException"));
        RETURN_IF_ERROR(ex.err_);
    }
    return nullptr;
}
ModelInstanceState::ModelInstanceState(ModelState *model_state, TRITONBACKEND_ModelInstance
*triton_model_instance)
    : BackendModelInstance(model_state, triton_model_instance), model_state_(model_state)
{
    LOG_MESSAGE(TRITONSERVER_LOG_VERBOSE, (std::string("Create MindIE
ModelInstanceState:")).c_str());
}
void ModelInstanceState::Enqueue(TRITONBACKEND_Request **requests, const uint32_t
request_count)
{
    LOG_MESSAGE(
        TRITONSERVER_LOG_VERBOSE, (std::string("Process Requests: Executing
ModelInstanceState::Enqueue")).c_str());

    std::unordered_map<mindie_llm::InferRequestId, std::shared_ptr<triton::backend::mindie::InferTask>>
newInferTaskMap;
    std::vector<std::shared_ptr<mindie_llm::InferRequest>> newRequests;
    for (uint32_t i = 0; i < request_count; i++) {
        TRITONBACKEND_Request *bRequest = requests[i];
        auto inferTask = std::make_shared<InferTask>(bRequest);
        auto req = inferTask->GetMieRequest();
        auto requestId = req->GetRequestId();
        newInferTaskMap[requestId] = inferTask;
        newRequests.push_back(req);
    }
    std::unique_lock lock(model_state_->GetMutex());
    model_state_->GetInferTasksMap().insert(newInferTaskMap.begin(), newInferTaskMap.end());
    for (uint32_t i = 0; i < newRequests.size(); i++) {
        model_state_->GetRequestsQueue().push(newRequests.at(i));
    }
}
} // namespace triton::backend::mindie
```

- src/mindie_utils.h

```
// Copyright 2021, NVIDIA CORPORATION & AFFILIATES. All rights reserved.
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
//  * Redistributions of source code must retain the above copyright
```

```
//    notice, this list of conditions and the following disclaimer.
//  * Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
//  * Neither the name of NVIDIA CORPORATION nor the names of its
//    contributors may be used to endorse or promote products derived
//    from this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ``AS IS'' AND ANY
// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
// PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
// CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
// PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
// OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
// OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#ifndef MINDIE_UTILS_H
#define MINDIE_UTILS_H
//
// ModelInstanceState
//
// State associated with a model instance. An object of this class is
// created and associated with each
// TRITONBACKEND_ModelInstance. ModelInstanceState is derived from
// BackendModelInstance class provided in the backend utilities that
// provides many common functions.
//
#include "triton/backend/backend_common.h"
#include "triton/backend/backend_model_instance.h"
#include "triton/backend/backend_model.h"
#include "triton/core/tritonbackend.h"
#include "llm_manager/infer_request.h"
#include "llm_manager/llm_manager.h"
#include "infer_task.h"

namespace triton::backend::mindie
{
class ModelState;
class MindIEUtils {
public:
    static TRITONSERVER_Error* ConvertResponse(TRITONBACKEND_Response* bResponse,
      const mindie_llm::TensorMap& result);
    static std::string GetEngineConfigPath(ModelState* model_state);
};
}
#endif
```

- src/mindie_utils.cc

```
// Copyright 2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
//  * Redistributions of source code must retain the above copyright
//    notice, this list of conditions and the following disclaimer.
//  * Redistributions in binary form must reproduce the above copyright
//    notice, this list of conditions and the following disclaimer in the
//    documentation and/or other materials provided with the distribution.
//  * Neither the name of NVIDIA CORPORATION nor the names of its
//    contributors may be used to endorse or promote products derived
//    from this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ``AS IS'' AND ANY
// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
// PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
// CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
```

```
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
// PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
// OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
// OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#include "mindie_utils.h"
#include <iostream>
#include <cmath>
#include <securec.h>
#include "model_state.h"

namespace triton::backend::mindie
{
TRITONSERVER_Error* MindIEUtils::ConvertResponse(TRITONBACKEND_Response* bResponse,
    const mindie_llm::TensorMap& result)
{
    for (const auto& entry : result) {
        const std::string& name = entry.first;
        if (name != "OUTPUT_IDS" && name != "IBIS_EOS_ATTR") {
            continue;
        }
        const auto& tensor = entry.second;
        TRITONBACKEND_Output* output = nullptr;
        RETURN_IF_ERROR(TRITONBACKEND_ResponseOutput(bResponse, &output, name.c_str(),
                                                     (TRITONSERVER_DataType)tensor->GetDataType(),
                                                     tensor->GetShape().data(),
                                                     tensor->GetShape().size()));

        uint64_t byteSize = tensor->GetSize();
        void* buffer = nullptr;
        TRITONSERVER_MemoryType memory_type = (TRITONSERVER_MemoryType)tensor-
>GetMemType();
        int64_t memory_type_id = 0;

        RETURN_IF_ERROR(TRITONBACKEND_OutputBuffer(output, &buffer, byteSize,
                                                   &memory_type, &memory_type_id));

        if (memory_type != TRITONSERVER_MEMORY_CPU && memory_type !=
TRITONSERVER_MEMORY_CPU_PINNED) {
            return TRITONSERVER_ErrorNew(TRITONSERVER_ERROR_INTERNAL, "Triton failed to allocate
output buffer on CPU");
        }
        errno_t memRet = memcpy_s(buffer, byteSize, tensor->GetData(), byteSize);
        if (memRet != EOK) {
            return TRITONSERVER_ErrorNew(TRITONSERVER_ERROR_INTERNAL,
                                         "Failed to copy tensor data to Triton output buffer");
        }
    }
    return nullptr;
}
std::string MindIEUtils::GetEngineConfigPath(ModelState* model_state)
{
    triton::common::TritonJson::Value parameters;
    TRITONSERVER_Error* err = model_state->ModelConfig().MemberAsObject("parameters",
&parameters);
    if (err != nullptr) {
        TRITONSERVER_ErrorDelete(err);
        throw std::runtime_error("Model config doesn't have a parameters section");
    }
    std::string engineConfigPath;
    LOG_IF_ERROR(
        GetParameterValue(parameters, "engine_config_path", &engineConfigPath), "failed to get
log_path config.");
    return engineConfigPath;
}
} // namespace triton::backend::mindie
```

- src/infer_task.h

```
#ifndef MINDIE_INFERTASK_H
#define MINDIE_INFERTASK_H

#include "triton/backend/backend_common.h"
#include "triton/core/tritonbackend.h"
#include "llm_manager/infer_request.h"
#include "llm_manager/infer_tensor.h"
#include <unordered_set>

namespace triton::backend::mindie {
class InferTask {
public:
    explicit InferTask(TRITONBACKEND_Request* request);
    ~InferTask();
    TRITONBACKEND_ResponseFactory* GetResponseFactory();
    std::shared_ptr<mindie_llm::InferRequest> GetMieRequest() const;
    TRITONBACKEND_Request* GetBackendRequest() const;
private:
    template <typename T>
    void SetSamplingValue(std::shared_ptr<mindie_llm::InferRequest>& leRequest,
                          TRITONBACKEND_Request* request, const char* postprocessParam,
                          mindie_llm::InferDataType type, T default_sampling_value)
    {
        const int64_t paramNum = 1;
        auto samplingTensor = std::make_shared<mindie_llm::InferTensor>(postprocessParam, type,
                                                                        std::vector<int64_t>{1, paramNum});
        bool ret = samplingTensor->Allocate(paramNum * sizeof(T));
        samplingTensor->SetRelease(false);
        if (!ret) {
            LOG_MESSAGE(TRITONSERVER_LOG_ERROR, "failed to allocate data for sampling param");
        }
        T *samplingBuffer = static_cast<T*>(samplingTensor->GetData());
        TRITONBACKEND_Input* input = nullptr;
        const void* buffer;
        size_t bufferByteSize;
        TRITONSERVER_MemoryType memoryType;
        int64_t memoryType_id;
        if (TRITONBACKEND_RequestInput(request, postprocessParam, &input) != nullptr) {
            *samplingBuffer = static_cast<T>(default_sampling_value);
        } else {
            TRITONBACKEND_InputBuffer(input, 0, &buffer, &bufferByteSize, &memoryType,
&memoryType_id);
            *samplingBuffer = static_cast<T>(*reinterpret_cast<const T*>(buffer));
        }
        leRequest->AddTensor(postprocessParam, samplingTensor);
    }

    std::shared_ptr<mindie_llm::InferRequest> ConvertRequest(TRITONBACKEND_Request* request);
    void AddMieInput(TRITONBACKEND_Request* bRequest, std::shared_ptr<mindie_llm::InferRequest>
&leRequest,
        const char* name);

    void SetSampling(std::shared_ptr<mindie_llm::InferRequest>& leRequest,
TRITONBACKEND_Request* request);
    std::shared_ptr<mindie_llm::InferRequest> mieRequest_;
    TRITONBACKEND_ResponseFactory* responseFactory_;
    TRITONBACKEND_Request* backendRequest_;
};
}
#endif
```

- src/infer_task.cc

```
#include "infer_task.h"
#include <securec.h>
namespace triton::backend::mindie {
InferTask::InferTask(TRITONBACKEND_Request* request)
{
    mieRequest_ = ConvertRequest(request);
    TRITONSERVER_Error* err = nullptr;
    err = TRITONBACKEND_ResponseFactoryNew(&responseFactory_, request);
```



```
    LOG_MESSAGE(
        TRITONSERVER_LOG_VERBOSE,
        (std::string("MindIE TRITONBACKEND_ResponseFactoryNew: ") +
         ((err == nullptr) ? "Success" : TRITONSERVER_ErrorMessage(err)))
            .c_str());
    backendRequest_ = request;
}
InferTask::~InferTask()
{
    if (responseFactory_ != nullptr) {
        TRITONBACKEND_ResponseFactoryDelete(responseFactory_);
    }
}
TRITONBACKEND_ResponseFactory* InferTask::GetResponseFactory()
{
    return responseFactory_;
}
std::shared_ptr<mindie_llm::InferRequest> InferTask::GetMieRequest() const
{
    return mieRequest_;
}
void InferTask::AddMieInput(TRITONBACKEND_Request* bRequest,
    std::shared_ptr<mindie_llm::InferRequest> &leRequest, const char* name)
{
    TRITONBACKEND_Input* input = nullptr;
    if (TRITONBACKEND_RequestInput(bRequest, name, &input) != nullptr) {
        return;
    }
    const int64_t* shape;
    TRITONSERVER_DataType datatype;
    uint32_t dims_count;
    size_t buffer_byte_size;
    TRITONSERVER_MemoryType data_memory_type = TRITONSERVER_MEMORY_CPU;
    int64_t data_memory_id;
    const void* buffer = nullptr;
    TRITONBACKEND_InputProperties(
        input, &name, &datatype, &shape, &dims_count, nullptr, nullptr);
    TRITONBACKEND_InputBuffer(
        input, 0 /* idx */, &buffer,
        &buffer_byte_size, &data_memory_type, &data_memory_id);
    auto tensor = std::make_shared<mindie_llm::InferTensor>(name,
static_cast<mindie_llm::InferDataType>(datatype),
                                                std::vector<int64_t>(shape, shape + dims_count));
    bool ret = tensor->Allocate(buffer_byte_size);
    tensor->SetRelease(false);
    if (!ret) {
        LOG_MESSAGE(TRITONSERVER_LOG_ERROR, "failed to allocate data for INPUT_IDS params");
    }
    errno_t memRet = memcpy_s(tensor->GetData(), buffer_byte_size, buffer, buffer_byte_size);
    if (memRet != EOK) {
        LOG_MESSAGE(TRITONSERVER_LOG_ERROR, "Triton failed to allocate output buffer on CPU");
        return;
    }

    leRequest->AddTensor(name, tensor);
}
void InferTask::SetSampling(std::shared_ptr<mindie_llm::InferRequest>& leRequest,
TRITONBACKEND_Request* request)
{
    SetSamplingValue<float>(leRequest, request, "TEMPERATURE",
mindie_llm::InferDataType::TYPE_FP32, 1.0f);
    SetSamplingValue<int32_t>(leRequest, request, "TOP_K", mindie_llm::InferDataType::TYPE_INT32,
int32_t(0));
    SetSamplingValue<float>(leRequest, request, "TOP_P", mindie_llm::InferDataType::TYPE_FP32, 1.0f);
    SetSamplingValue<float>(leRequest, request, "TYPICAL_P", mindie_llm::InferDataType::TYPE_FP32,
1.0f);
    SetSamplingValue<bool>(leRequest, request, "DO_SAMPLE",
mindie_llm::InferDataType::TYPE_BOOL, false);
    SetSamplingValue<uint64_t>(leRequest, request, "SEED", mindie_llm::InferDataType::TYPE_UINT64,
```



```
uint64_t(0));
    SetSamplingValue<float>(leRequest, request, "REPETITION_PENALTY",
mindie_llm::InferDataType::TYPE_FP32, 1.0f);
    SetSamplingValue<float>(leRequest, request, "FREQUENCY_PENALTY",
mindie_llm::InferDataType::TYPE_FP32, 0.0f);
    SetSamplingValue<float>(leRequest, request, "PRESENCE_PENALTY",
mindie_llm::InferDataType::TYPE_FP32, 0.0f);
    SetSamplingValue<bool>(leRequest, request, "WATERMARK",
mindie_llm::InferDataType::TYPE_BOOL, false);
}

std::shared_ptr<mindie_llm::InferRequest> InferTask::ConvertRequest(
    TRITONBACKEND_Request* request)
{
    const char* request_id;
    if (TRITONBACKEND_RequestId(request, &request_id) != nullptr) {
        return nullptr;
    }
    auto leRequest =
std::make_shared<mindie_llm::InferRequest>(mindie_llm::InferRequestId(request_id));
    AddMieInput(request, leRequest, "INPUT_IDS");

    SetSampling(leRequest, request);
    // mindie_llm::TensorPtr tp;
    // leRequest->GetTensorByName("REPETITION_PENALTY", tp);
    // float* buffer = static_cast<float*>(tp->GetData());
    // std::cout << "************ REPETITION_PENALTY: " << *buffer
    //     << "**************" << std::endl;
    return leRequest;
}
TRITONBACKEND_Request* InferTask::GetBackendRequest() const
{
    return backendRequest_;
}
} // namespace triton::backend::mindie
```

- example/tritonModels/llama3_8b/config.pbtxt

```
name: "llama3_8b"
max_batch_size : 200
backend: "mindie"
model_transaction_policy {
  decoupled: True
}

dynamic_batching {
   max_queue_delay_microseconds: 1000
default_queue_policy : {
            timeout_action : 1
                default_timeout_microseconds : 50000000
        }
}
# path of config.json to mindie
parameters {
  key: "engine_config_path",
  value: {string_value: "tritonModels/config.json"}
}

# number of model instances, currently just support a single model instance
parameters {
  key: "model_instance_number"
  value: { string_value: "1" }
}
# npu card number on which each model instance is running, separate them sequentially with a
semicolon
parameters {
  key: "npu_device_ids"
  value: { string_value: "01;"}
}

input [
```



```
 {
  name: "INPUT_IDS"
  data_type: TYPE_INT64
  dims: [ -1 ]
 },
 {
  name: "TEMPERATURE"
  data_type: TYPE_FP32
  dims: [ -1 ]
  optional: true
 },
 {
  name: "TOP_K"
  data_type: TYPE_INT32
  dims: [ -1 ]
  optional: true
 },
 {
  name: "TOP_P"
  data_type: TYPE_FP32
  dims: [ -1 ]
  optional: true
 },
 {
  name: "TYPICAL_P"
  data_type: TYPE_FP32
  dims: [ -1 ]
  optional: true
 },
 {
  name: "DO_SAMPLE"
  data_type: TYPE_BOOL
  dims: [ -1 ]
  optional: true
 },
 {
  name: "SEED"
  data_type: TYPE_UINT64
  dims: [ -1 ]
  optional: true
 },
 {
  name: "REPETITION_PENALTY"
  data_type: TYPE_FP32
  dims: [ -1 ]
  optional: true
 },
 {
  name: "FREQUENCY_PENALTY"
  data_type: TYPE_FP32
  dims: [ -1 ]
  optional: true
 },
 {
  name: "PRESENCE_PENALTY"
  data_type: TYPE_FP32
  dims: [ -1 ]
  optional: true
 },
 {
  name: "WATERMARK"
  data_type: TYPE_BOOL
  dims: [ -1 ]
  optional: true
 }
]

output [
 {
```



```
    name: "OUTPUT_IDS"
    data_type: TYPE_INT64
    dims: [ -1 ]
  },
  {
    name: "IBIS_EOS_ATTR"
    data_type: TYPE_INT64
    dims: [ -1 ]
  }
]
instance_group [
  {
    count: 1
    kind: KIND_CPU
  }
]
```

- example/tritonModels/config.json

```
{
    "Version" : "1.0.0",
    "LogConfig" :
    {
        "logLevel" : "Info",
        "logFileSize" : 20,
        "logFileNum" : 20,
        "logPath" : "logs/mindservice.log"
    },
    "ServerConfig" :
    {
        "ipAddress" : "127.0.0.1",
        "managementIpAddress" : "127.0.0.2",
        "port" : 1025,
        "managementPort" : 1026,
        "allowAllZeroIpListening" : false,
        "maxLinkNum" : 1000,
        "httpsEnabled" : false,
        "fullTextEnabled" : false,
        "tlsCaPath" : "security/ca/",
        "tlsCaFile" : ["ca.pem"],
        "tlsCert" : "security/certs/server.pem",
        "tlsPk" : "security/keys/server.key.pem",
        "tlsPkPwd" : "security/pass/key_pwd.txt",
        "tlsCrl" : "security/certs/server_crl.pem",
        "managementTlsCaFile" : ["management_ca.pem"],
        "managementTlsCert" : "security/certs/management/server.pem",
        "managementTlsPk" : "security/keys/management/server.key.pem",
        "managementTlsPkPwd" : "security/pass/management/key_pwd.txt",
        "managementTlsCrl" : "security/certs/management/server_crl.pem",
        "kmcKsfMaster" : "tools/pmt/master/ksfa",
        "kmcKsfStandby" : "tools/pmt/standby/ksfb",
        "inferMode" : "standard",
        "interCommTLSEnabled" : true,
        "interCommPort" : 1121,
        "interCommTlsCaFile" : "security/grpc/ca/ca.pem",
        "interCommTlsCert" : "security/grpc/certs/server.pem",
        "interCommPk" : "security/grpc/keys/server.key.pem",
        "interCommPkPwd" : "security/grpc/pass/key_pwd.txt",
        "interCommTlsCrl" : "security/grpc/certs/server_crl.pem",
        "openAiSupport" : "vllm"
    },
    "BackendConfig" : {
        "backendName" : "mindieservice_llm_engine",
        "modelInstanceNumber" : 1,
        "npuDeviceIds" : [[0]],
        "tokenizerProcessNumber" : 8,
        "multiNodesInferEnabled" : false,
        "multiNodesInferPort" : 1120,
        "interNodeTLSEnabled" : true,
        "interNodeTlsCaFile" : "security/grpc/ca/ca.pem",
        "interNodeTlsCert" : "security/grpc/certs/server.pem",
```



```
    "interNodeTlsPk" : "security/grpc/keys/server.key.pem",
    "interNodeTlsPkPwd" : "security/grpc/pass/mindie_server_key_pwd.txt",
    "interNodeTlsCrl" : "security/grpc/certs/server_crl.pem",
    "interNodeKmcKsfMaster" : "tools/pmt/master/ksfa",
    "interNodeKmcKsfStandby" : "tools/pmt/standby/ksfb",
    "ModelDeployConfig" :
    {
        "maxSeqLen" : 2560,
        "maxInputTokenLen" : 2048,
        "truncation" : false,
        "ModelConfig" : [
            {
                "modelInstanceType" : "Standard",
                "modelName" : "llama3_8b",
                "modelWeightPath" : "/path/to/weights/LLaMA3-8B",
                "worldSize" : 2,
                "cpuMemSize" : 5,
                "npuMemSize" : -1,
                "backendType" : "atb"
            }
        ]
    },
    "ScheduleConfig" :
    {
        "templateType" : "Standard",
        "templateName" : "Standard_LLM",
        "cacheBlockSize" : 128,
        "maxPrefillBatchSize" : 50,
        "maxPrefillTokens" : 8192,
        "prefillTimeMsPerReq" : 150,
        "prefillPolicyType" : 0,
        "decodeTimeMsPerReq" : 50,
        "decodePolicyType" : 0,
        "maxBatchSize" : 200,
        "maxIterTimes" : 512,
        "maxPreemptCount" : 0,
        "supportSelectBatch" : false,
        "maxQueueDelayMicroseconds" : 5000
    }
  }
}
```

- example/set_env.sh

```
# Set the environment variables for ascend-toolkit, nnal, atb_models, and mindie, and modify them manually based on their installation paths
# source ascend-toolkit/set_env.sh
# source nnal/atb/set_env.sh
# source atb_models/set_env.sh
# source mindie/set_env.sh
export LCCL_DETERMINISTIC=1
export HCCL_DETERMINISTIC=true
export ATB_MATMUL_SHUFFLE_K_ENABLE=0
export ATB_LLM_LCOC_ENABLE=1
export ATB_STREAM_SYNC_EVERY_KERNEL_ENABLE=0
export ATB_STREAM_SYNC_EVERY_RUNNER_ENABLE=0
export ATB_STREAM_SYNC_EVERY_OPERATION_ENABLE=0
export ATB_OPERATION_EXECUTE_ASYNC=0
export TASK_QUEUE_ENABLE=0
export ATB_OPSRUNNER_KERNEL_CACHE_LOCAL_COUNT=8
export ATB_OPSRUNNER_KERNEL_CACHE_GLOABL_COUNT=16
export ATB_CONTEXT_WORKSPACE_RING=1
export HCCL_BUFFSIZE=120
export ATB_LAYER_INTERNAL_TENSOR_REUSE=1
export MASTER_ADDR=127.0.0.1
export MASTER_PORT=12345
```

- example/launch.sh

```
export MINDIE_LLM_PYTHON_LOG_TO_FILE=0
export MINDIE_LLM_PYTHON_LOG_TO_STDOUT=0
export MINDIE_LLM_PYTHON_LOG_PATH=./mindiepython.log
```



```
export MINDIE_LLM_PYTHON_LOG_LEVEL=INFO
export MINDIE_LLM_LOG_TO_STDOUT=0
export MINDIE_LLM_LOG_TO_FILE=0
export MINDIE_LLM_LOG_LEVEL=INFO
export IBIS_PYTHON_LOG=0
rm -f ./tritonserver.log
# The path to the Triton model repository following --model-repository needs to be modified manually.
/opt/tritonserver/bin/tritonserver --model-repository=tritonModels \
--backend-config=mindie,shm-default-byte-size=134217728 --http-port 8110 --grpc-port 8111 --metrics-port 8112 --log-verbose=1 --log-info=1 --log-warning=0 --log-error=0 --log-file ./tritonserver.log
```

- example/submit.py

```
import sys
import datetime
import queue
import random
import threading
from infer_client import ModelClient, InferRequest
import time
import json
import numpy as np
import multiprocessing as mp
from concurrent.futures import ThreadPoolExecutor, wait, ALL_COMPLETED
import argparse
import pandas as pd
from prettytable import PrettyTable
from transformers import AutoTokenizer
from logger_util import get_logger
logger = get_logger()
SAMPLING_Param = {'TEMPERATURE':1.0, # float32
                  'TOP_K':0,         # int32
                  'TOP_P':1.0,       # float32
                  'TYPICAL_P':1.0,   # float32
                  'DO_SAMPLE':1,     # bool
                  'SEED':0,   # uint64
                  'REPETITION_PENALTY':1.0, # float32
                  'FREQUENCY_PENALTY':0,    # float32
                  'PRESENCE_PENALTY':0,     # float32
                  'WATERMARK':0             # bool
                 }
RESULT_FILE_NAME = ''
MODEL_NAME = ''
MODEL_PATH = ''
TRACE_DATASET = ''
DATASET_LANGUAGE = ''
# total submission time
TOTAL_SUBMIT_TIME = 0
# the number of requests for different request types
REQUEST_TYPE_NUM = 2000
# input token quantity (randomly distributed with a uniform distribution within [INPUT_TOKEN_NUM_START, INPUT_TOKEN_NUM_END])
INPUT_TOKEN_NUM_START = 1
INPUT_TOKEN_NUM_END = 16
# output token quantity (randomly distributed with a uniform distribution within [OUTPUT_TOKEN_NUM_START, OUTPUT_TOKEN_NUM_END])
OUTPUT_TOKEN_NUM_START = 1
OUTPUT_TOKEN_NUM_END = 256
THREAD_NUM = 200
SEED = 1
random.seed(SEED)
class SubmitService:
    def __init__(self, token_config_list, thread_num):
        self.requests = queue.Queue()
        self.exception_queue = queue.Queue()
        self.total_request_num = 0
        self.submitted_request_num = 0
        self.start_time = 0
        self.elapsed = 0
```

```
        self.thread_num = thread_num
        self.lock = threading.Lock()
        request_idx = 0
        for token_config in token_config_list:
            request_num = token_config[0]
            self.total_request_num += request_num
            input_tensor = token_config[1]
            for _ in range(request_num):
                input_lens = token_config[2]
                question = token_config[3]
                self.requests.put(InferRequest(str(request_idx), input_tensor, input_lens, question,
sampling_param=SAMPLING_Param, model_name=MODEL_NAME))
                request_idx += 1
        logger.info("total request num:{}".format(self.total_request_num))
        self.metrics_queue = dict()
    def start(self):
        logger.info("\n############ submit job ###########")
        all_task = []
        init = queue.Queue(self.thread_num)
        with ThreadPoolExecutor(self.thread_num + 1, thread_name_prefix='submit') as p:
            for i in range(THREAD_NUM):
                all_task.append(p.submit(self._submit_requests, i, init))
            all_task.append(p.submit(self._monitor, init))
            wait(all_task, return_when=ALL_COMPLETED)
        if not self.exception_queue.empty():
            raise self.exception_queue.get()
        logger.info("############ submit over ###########")
    def _submit_requests(self, client_id, init):
        try:
            client = ModelClient(client_id)
            init.put(1)
            # wait all grpc client init ready
            while not init.full():
                time.sleep(0.1)
            while not self.requests.empty():
                with self.lock:
                    if not self.requests.empty():
                        request = self.requests.get()
                        self.requests.task_done()
                logger.info("start a new request.client id:{},request_id:{}".format(client_id,
request.request_id))
                success, metrics = client.infer(request)
                if not success:
                    logger.info("request {} submit failed! triton url: {}, model name: {}, inputs: {}".format(
                        request.request_id, client.server_url, request.model_name,
                        request.inputs))
                self._request_finish(request.request_id,metrics)
            client.triton_client.stop_stream()
        except Exception as e:
            self.exception_queue.put(e)
    def _request_finish(self, request_id, metrics):
        self.metrics_queue[request_id]  = metrics.format()
    def format_metrics(self):
        metrics_list = []
        for _, metric in self.metrics_queue.items():
            metrics_list.append(metric)
        json_str = json.dumps(metrics_list, ensure_ascii=False)
        with open(RESULT_FILE_NAME, 'w') as f:
            f.write(json_str)
        request_df = pd.read_json(json_str, orient='records')
        metrics = PrettyTable(['METRICS', 'AVG', 'P75', 'P99', 'MAX', 'N'])
        origin_metric = ['FirstTokenTime', 'DecodeTime', 'MaxDecodeTime', 'GenerateTime',
'InputCharacters',
                         'InputTokens', 'GeneratedTokens', 'GenerateTokenSpeed']
        for metric in origin_metric:
            if metric in ['FirstTokenTime', 'DecodeTime', 'MaxDecodeTime', 'GenerateTime']:
                metrics.add_row([metric, str(round(request_df[metric].mean() * 1000)) + 'ms',
                                 str(round(request_df[metric].quantile(0.75) * 1000)) + 'ms',
                                 str(round(request_df[metric].quantile(0.99) * 1000)) + 'ms',
```



```
                            str(round(request_df[metric].max() * 1000)) + 'ms',
                            request_df[metric].count()])
            elif metric in ['InputCharacters', 'InputTokens', 'GeneratedTokens']:
                metrics.add_row([metric, round(request_df[metric].mean()),
round(request_df[metric].quantile(0.75)),
                            round(request_df[metric].quantile(0.99)), round(request_df[metric].max()),
                            request_df[metric].count()])
            else:
                metrics.add_row([metric, str(round(request_df[metric].mean(), 2)) + '/s',
                            str(round(request_df[metric].quantile(0.75), 2)) + '/s',
                            str(round(request_df[metric].quantile(0.99), 2)) + '/s',
                            str(round(request_df[metric].max(), 2)) + '/s',
                            request_df[metric].count()])
        logger.info(metrics)
        total = PrettyTable(['METRICS', 'VALUE'])
        total.add_row(['Current Time', datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')])
        total.add_row(['Data Source', TRACE_DATASET])
        total.add_row(['NonEmpty', request_df['NotEmpty'].sum()])
        total.add_row(['Returned', request_df['InputCharacters'].count()])
        total.add_row(['Total', self.total_request_num])
        total.add_row(['Concurrency', THREAD_NUM])
        total.add_row(['Time Elapsed', round(self.elapsed)])
        total.add_row(['Throughput', round(self.total_request_num / self.elapsed, 2)])
        total.add_row(['JCT', round(request_df['GenerateTime'].mean(), 2)])
        total.add_row(['GenerateSpeed', round(request_df['GeneratedTokens'].sum() / self.elapsed, 2)])
        total.add_row(
            ['GenerateSpeedPerClient', round(request_df['GeneratedTokens'].sum() / self.elapsed /
THREAD_NUM, 2)])
        logger.info(total)
    def _monitor(self, init):
        # wait all grpc client init ready
        while not init.full():
            time.sleep(0.1)
        self.start_time = time.time()
        last_submitted_count = -1
        while len(self.metrics_queue)<self.total_request_num:
            submitted_count = len(self.metrics_queue)
            if last_submitted_count != submitted_count:
                logger.info("progress: {}/{}, spend time:{}".format(submitted_count,
                                                    self.total_request_num,
                                                    int(time.time() - self.start_time)))
                last_submitted_count = submitted_count
            time.sleep(0.1)
        self.elapsed = time.time() - self.start_time
        self.format_metrics()
def gen_requests_from_config():
    token_config_list = list()
    for _ in range(REQUEST_TYPE_NUM):
        request_num = 1
        input_token_num = random.randint(INPUT_TOKEN_NUM_START, INPUT_TOKEN_NUM_END)
        output_token_num = random.randint(OUTPUT_TOKEN_NUM_START,
OUTPUT_TOKEN_NUM_END)
        token_config_list.append([request_num, input_token_num, output_token_num])
    return token_config_list
def gen_requests_from_trace(trace_file):
    tokenizer = AutoTokenizer.from_pretrained(
    MODEL_PATH, trust_remote_code=True, use_fast=True)
    requests = list()
    df = pd.read_json(trace_file, lines=True)
    for i, row in df.iterrows():
        request_num = 1
        question = row["question"]
        token = tokenizer([question], return_tensors="np")
        token = token["input_ids"].astype(np.int64)
        input_tensor = token.reshape(1, -1)
        input_lens = len(question)
        requests.append([request_num, input_tensor, input_lens, question])
    return requests
def parser_input():
```



```
    parser = argparse.ArgumentParser(description='Personal information')
    parser.add_argument('--name', dest='name', required=True, type=str, help='Name of the model')
    parser.add_argument('--model_path', dest='model_path', required=True, type=str, help='path of the
model')
    parser.add_argument('--trace_dataset', dest='trace_dataset', required=True, type=str,
                        help='Trace dataset with jsonl style file')
    parser.add_argument('--dataset_language', dest='dataset_language', default='english', type=str,
                        help='Language of the trace dataset')
    parser.add_argument('--req_num', dest='req_num', default=2000, type=int, help='Number of the
request')
    parser.add_argument('--max_input_len', dest='max_input_len', default=16, type=int,
                        help='Max length of input sequence')
    parser.add_argument('--max_output_len', dest='max_output_len', default=100, type=int,
                        help='Max length of output sequence')
    args = parser.parse_args()
    globals()['MODEL_NAME'] = args.name
    globals()['MODEL_PATH'] = args.model_path
    globals()['REQUEST_TYPE_NUM'] = args.req_num
    globals()['INPUT_TOKEN_NUM_END'] = args.max_input_len
    globals()['OUTPUT_TOKEN_NUM_END'] = args.max_output_len
    globals()['TRACE_DATASET'] = args.trace_dataset
    globals()['DATASET_LANGUAGE'] = args.dataset_language
def main():
    parser_input()
    dt = datetime.datetime.now().strftime('%Y-%m-%d|%H:%M:%S')
    if TRACE_DATASET != '':
        requests = gen_requests_from_trace(TRACE_DATASET)
        globals()['RESULT_FILE_NAME'] = './trace.json'
    else:
        requests = gen_requests_from_config()
        globals()['RESULT_FILE_NAME'] = './mock-{}-{}-{}-{}-{}.json'.format(MODEL_NAME,
REQUEST_TYPE_NUM,
                                                                            INPUT_TOKEN_NUM_END,
                                                                            OUTPUT_TOKEN_NUM_END, dt)
    submit_service = SubmitService(requests, THREAD_NUM)
    submit_service.start()
if __name__ == "__main__":
    try:
        main()
    except:
        ttype, tvalue, traceback = sys.exc_info()
        print(ttype, tvalue, end="\n")
        idx = 1
        while traceback:
            print("第{}层堆栈信息".format(idx))
            tracebackCode = traceback.tb_frame.f_code
            print("文件名：{}".format(tracebackCode.co_filename))
            print("函数或者模块名：{}".format(tracebackCode.co_name))
            traceback = traceback.tb_next
            idx += 1
        sys.exit(1)
```

- example/client_stream.py

```
# Copyright 2022, NVIDIA CORPORATION & AFFILIATES. All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
#  * Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
#  * Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in the
#    documentation and/or other materials provided with the distribution.
#  * Neither the name of NVIDIA CORPORATION nor the names of its
#    contributors may be used to endorse or promote products derived
#    from this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ``AS IS'' AND ANY
# EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
```

```
# PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
# CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
# EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
# PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
# PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
# OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
# OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
from functools import partial
import numpy as np
import queue
from tritonclient.utils import *
import tritonclient.grpc as grpcclient
from transformers import AutoTokenizer
SAMPLING_Param = {'TEMPERATURE':1.0, # float32
                  'TOP_K':0,         # int32
                  'TOP_P':1.0,       # float32
                  'TYPICAL_P':1.0,   # float32
                  'DO_SAMPLE':1,     # bool
                  'SEED':0,   # uint64
                  'REPETITION_PENALTY':1.0, # float32
                  'FREQUENCY_PENALTY':0,    # float32
                  'PRESENCE_PENALTY':0,     # float32
                  'WATERMARK':0             # bool
                 }
MODEL_PATH = "weights/LLaMA3-8B"
model_name = "llama3_8b"
question = "Please introduce yourself."
tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH, trust_remote_code=True, use_fast=True)
class UserData:
    def __init__(self):
        self._completed_requests = queue.Queue()
def callback(user_data, result, error):
    if error:
        user_data._completed_requests.put(error)
    else:
        user_data._completed_requests.put(result)
def main():
    user_data = UserData()
    with grpcclient.InferenceServerClient(url="localhost:8111", verbose=False) as triton_client:
        triton_client.start_stream(callback=partial(callback, user_data))
        request_id = '1'
        inputs = []
        outputs = []
        input_ids = tokenizer([question], return_tensors="np")
        input_ids = input_ids["input_ids"].astype(np.int64)
        input_ids = input_ids.reshape(1, -1)
        sampling_param = SAMPLING_Param
        temperature = np.array([[sampling_param['TEMPERATURE']]], np.float32)
        top_k = np.array([[sampling_param['TOP_K']]], np.int32)
        top_p = np.array([[sampling_param['TOP_P']]], np.float32)
        typical_p = np.array([[sampling_param['TYPICAL_P']]], np.float32)
        do_sample = np.array([[sampling_param['DO_SAMPLE']]], bool)
        random_seed = np.array([[sampling_param['SEED']]], np.uint64)
        repetition_penalty = np.array([[sampling_param['REPETITION_PENALTY']]], np.float32)
        frequency_penalty = np.array([[sampling_param['FREQUENCY_PENALTY']]], np.float32)
        presence_penalty = np.array([[sampling_param['PRESENCE_PENALTY']]], np.float32)
        watermark = np.array([[sampling_param['WATERMARK']]], bool)
        inputs.append(grpcclient.InferInput('INPUT_IDS', list(input_ids.shape), "INT64"))
        inputs.append(grpcclient.InferInput('TEMPERATURE', list(temperature.shape), "FP32"))
        inputs.append(grpcclient.InferInput('TOP_K', list(top_k.shape), "INT32"))
        inputs.append(grpcclient.InferInput('TOP_P', list(top_p.shape), "FP32"))
        inputs.append(grpcclient.InferInput('TYPICAL_P', list(typical_p.shape), "FP32"))
        inputs.append(grpcclient.InferInput('DO_SAMPLE', list(do_sample.shape), "BOOL"))
        inputs.append(grpcclient.InferInput('SEED', list(random_seed.shape), "UINT64"))
        inputs.append(grpcclient.InferInput('REPETITION_PENALTY', list(repetition_penalty.shape),
"FP32"))
        inputs.append(grpcclient.InferInput('FREQUENCY_PENALTY', list(frequency_penalty.shape),
"FP32"))
```



```
        inputs.append(grpcclient.InferInput('PRESENCE_PENALTY', list(presence_penalty.shape), "FP32"))
        inputs.append(grpcclient.InferInput('WATERMARK', list(watermark.shape), "BOOL"))

        inputs[0].set_data_from_numpy(input_ids)
        inputs[1].set_data_from_numpy(temperature)
        inputs[2].set_data_from_numpy(top_k)
        inputs[3].set_data_from_numpy(top_p)
        inputs[4].set_data_from_numpy(typical_p)
        inputs[5].set_data_from_numpy(do_sample)
        inputs[6].set_data_from_numpy(random_seed)
        inputs[7].set_data_from_numpy(repetition_penalty)
        inputs[8].set_data_from_numpy(frequency_penalty)
        inputs[9].set_data_from_numpy(presence_penalty)
        inputs[10].set_data_from_numpy(watermark)
        outputs.append(grpcclient.InferRequestedOutput("OUTPUT_IDS"))
        outputs.append(grpcclient.InferRequestedOutput('IBIS_EOS_ATTR'))
        triton_client.async_stream_infer(model_name=model_name,
                                         inputs=inputs,
                                         request_id=request_id,
                                         outputs=outputs)
        triton_client.stop_stream()
        output_ids = []
        while not user_data._completed_requests.empty():
            data_item = user_data._completed_requests.get()
            output_id = data_item.as_numpy("OUTPUT_IDS")
            output_ids.extend(output_id)
        output = tokenizer.decode(output_ids)
        print(f"output:\n{output}")
if __name__=='__main__':
    main()
```

- example/infer_client.py

```
import sys
from functools import partial
import queue
from tritonclient.utils import *
import tritonclient.grpc as grpcclient
import time
import numpy as np
from transformers import AutoTokenizer
from logger_util import get_logger
logger = get_logger()
class UserData:
    def __init__(self):
        self.completed_requests = queue.Queue()
def callback(user_data, result, error):
    user_data.completed_requests.put((result, error))
class Metrics:
    def __init__(self):
        self.first_token_time = 0
        self.avg_decode_time = 0
        self.max_decode_time = 0
        self.full_time = 0
        self.prompt_lens = 0
        self.prompt_token_lens = 0
        self.generated_tokens = 0
        self.cost_timestamp_list = []
        self.client_id = -1
    def format(self):
        generate_token_speed = (self.generated_tokens / self.full_time) if self.full_time != 0 else 0
        return {'FirstTokenTime': self.first_token_time,
                'DecodeTime': self.avg_decode_time, 'MaxDecodeTime': self.max_decode_time,
                'GenerateTime': self.full_time, 'InputCharacters': self.prompt_lens,
                'InputTokens': self.prompt_token_lens,
                'GeneratedTokens': self.generated_tokens,
                'GenerateTokenSpeed': generate_token_speed,
                'NotEmpty': 0 if self.generated_tokens == 0 else 1,
                'CostTimestampList': self.cost_timestamp_list,
                'ClientID': self.client_id}
class InferRequest:
```



```
    def __init__(self, request_id, input_tensor, input_lens, question, sampling_param,
max_iter_times=4096, model_name='ibis_benchmark'):
        self.submit_time = 0
        self.request_id = request_id
        self.input_lens = input_lens
        self.question = question
        self.input_token_lens = input_tensor.shape[1]

        temperature = np.array([[sampling_param['TEMPERATURE']]], np.float32)
        top_k = np.array([[sampling_param['TOP_K']]], np.int32)
        top_p = np.array([[sampling_param['TOP_P']]], np.float32)
        typical_p = np.array([[sampling_param['TYPICAL_P']]], np.float32)
        do_sample = np.array([[sampling_param['DO_SAMPLE']]], bool)
        random_seed = np.array([[sampling_param['SEED']]], np.uint64)
        repetition_penalty = np.array([[sampling_param['REPETITION_PENALTY']]], np.float32)
        frequency_penalty = np.array([[sampling_param['FREQUENCY_PENALTY']]], np.float32)
        presence_penalty = np.array([[sampling_param['PRESENCE_PENALTY']]], np.float32)
        watermark = np.array([[sampling_param['WATERMARK']]], bool)
        self.inputs = []
        self.inputs.append(grpcclient.InferInput('INPUT_IDS', list(input_tensor.shape), "INT64"))
        self.inputs.append(grpcclient.InferInput('TEMPERATURE', list(temperature.shape), "FP32"))
        self.inputs.append(grpcclient.InferInput('TOP_K', list(top_k.shape), "INT32"))
        self.inputs.append(grpcclient.InferInput('TOP_P', list(top_p.shape), "FP32"))
        self.inputs.append(grpcclient.InferInput('TYPICAL_P', list(typical_p.shape), "FP32"))
        self.inputs.append(grpcclient.InferInput('DO_SAMPLE', list(do_sample.shape), "BOOL"))
        self.inputs.append(grpcclient.InferInput('SEED', list(random_seed.shape), "UINT64"))
        self.inputs.append(grpcclient.InferInput('REPETITION_PENALTY', list(repetition_penalty.shape),
"FP32"))
        self.inputs.append(grpcclient.InferInput('FREQUENCY_PENALTY', list(frequency_penalty.shape),
"FP32"))
        self.inputs.append(grpcclient.InferInput('PRESENCE_PENALTY', list(presence_penalty.shape),
"FP32"))
        self.inputs.append(grpcclient.InferInput('WATERMARK', list(watermark.shape), "BOOL"))

        self.inputs[0].set_data_from_numpy(input_tensor)
        self.inputs[1].set_data_from_numpy(temperature)
        self.inputs[2].set_data_from_numpy(top_k)
        self.inputs[3].set_data_from_numpy(top_p)
        self.inputs[4].set_data_from_numpy(typical_p)
        self.inputs[5].set_data_from_numpy(do_sample)
        self.inputs[6].set_data_from_numpy(random_seed)
        self.inputs[7].set_data_from_numpy(repetition_penalty)
        self.inputs[8].set_data_from_numpy(frequency_penalty)
        self.inputs[9].set_data_from_numpy(presence_penalty)
        self.inputs[10].set_data_from_numpy(watermark)

        self.outputs = []
        self.outputs.append(grpcclient.InferRequestedOutput('OUTPUT_IDS'))
        self.outputs.append(grpcclient.InferRequestedOutput('IBIS_EOS_ATTR'))
        self.max_iter_times = max_iter_times
        self.model_name = model_name
class ModelClient:
    def __init__(self, client_id=0, server_url="localhost:8111", verbose=False):
        self.server_url = server_url
        self.client_id = client_id
        self.triton_client = grpcclient.InferenceServerClient(url=server_url, verbose=verbose)
        self.user_data = UserData()
        self.triton_client.start_stream(callback=partial(callback, self.user_data))
    def infer(self, request: InferRequest):
        metrics = Metrics()
        metrics.client_id = self.client_id
        metrics.prompt_lens = request.input_lens
        metrics.prompt_token_lens = request.input_token_lens
        begin_time = time.time()
        try:
            logger.info("Send a request, request id : {}".format(request.request_id))
            self.triton_client.async_stream_infer(model_name=request.model_name,
                                                  inputs=request.inputs,
                                                  request_id=request.request_id,
```



```
                                        outputs=request.outputs
                                        )
        except Exception as e:
            raise e
        tokens = []
        last_token_time = time.time()
        metrics.cost_timestamp_list.append(last_token_time)
        decode_full_time = 0
        for i in range(request.max_iter_times):
            (output_tensor, error) = self.user_data.completed_requests.get()
            if error is not None:
                logger.error(error)
                return False, metrics
            gen_tokens = output_tensor.as_numpy('OUTPUT_IDS')
            for token in gen_tokens:
                tokens.append(token)
            if metrics.first_token_time == 0:
                metrics.first_token_time = time.time() - last_token_time
            else:
                decode_time = time.time() - last_token_time
                gen_token_num = output_tensor.as_numpy('IBIS_EOS_ATTR')[1]
                single_decode_time = decode_time / gen_token_num
                decode_full_time += decode_time
                if decode_time > metrics.max_decode_time:
                    metrics.max_decode_time = single_decode_time
            last_token_time = time.time()
            metrics.cost_timestamp_list.append(last_token_time)
            inferparam = output_tensor.get_response().parameters['triton_final_response']
            if(inferparam.bool_param):
                logger.info("recieve a eos token, request id : {}".format(request.request_id))
                break
        metrics.full_time = time.time() - begin_time
        metrics.generated_tokens = len(tokens)
        if metrics.generated_tokens > 1:
            metrics.avg_decode_time = decode_full_time / (metrics.generated_tokens - 1)
        logger.info("return from client infer, request id : {}".format(request.request_id))
        return True, metrics
```

- example/logger_util.py

```
import sys
import loguru
import os, time
os.environ['TZ'] = 'UTC'
time.tzset()
def get_logger():
    logger = loguru.logger
    logger.add("script.log", level="INFO")
    return logger
```

- build.sh

```
#!/bin/bash
set -e
export MINDIE_LLM_HOME_PATH=/usr/local/Ascend/mindie/latest/mindie-llm
export TRITON_HOME_PATH=/opt/tritonserver
if [ -z "$MINDIE_LLM_HOME_PATH" ]; then
  echo "env MINDIE_LLM_HOME_PATH is null, please install mindie and source set_env.sh"
  exit 1
fi
COMPILE_OPTIONS=""
if [ $(python3 -c 'import torch; print(torch.compiled_with_cxx11_abi())') == "True" ]; then
  USE_CXX11_ABI=ON
else
  USE_CXX11_ABI=OFF
fi
COMPILE_OPTIONS="${COMPILE_OPTIONS} -DUSE_CXX11_ABI=$USE_CXX11_ABI"
BUILD_TYPE="Release"
TRITON_VERSION="r24.02"
while getopts "dv:p:" opt; do
  case ${opt} in
    d)
```



```
    BUILD_TYPE="Debug"
    ;;
  v)
    TRITON_VERSION=$OPTARG
    ;;
  p)
    export TRITON_HOME_PATH=$OPTARG
    ;;
  \?)
    echo "Invalid option: -$opt" >&2
    exit 1
    ;;
 esac
done
echo "Triton version: ${TRITON_VERSION}"
if [ -z "$TRITON_HOME_PATH" ]; then
 echo "env TRITON_HOME_PATH is null, please set env or use -p to tell us where triton is installed."
 exit 1
fi
echo "Triton install path: ${TRITON_HOME_PATH}"
if [ ! -d "$TRITON_HOME_PATH" ]; then
 echo "$TRITON_HOME_PATH is not a directory! Please check triton install path."
 exit 1
fi
rm -rf build && mkdir build && cd build
COMPILE_OPTIONS="${COMPILE_OPTIONS} -DCMAKE_BUILD_TYPE=$BUILD_TYPE"
COMPILE_OPTIONS="${COMPILE_OPTIONS} -DCMAKE_INSTALL_PREFIX:PATH=`pwd`/install"
COMPILE_OPTIONS="${COMPILE_OPTIONS} -DTRITON_COMMON_REPO_TAG=$TRITON_VERSION"
COMPILE_OPTIONS="${COMPILE_OPTIONS} -DTRITON_BACKEND_REPO_TAG=$TRITON_VERSION"
COMPILE_OPTIONS="${COMPILE_OPTIONS} -DTRITON_CORE_REPO_TAG=$TRITON_VERSION"
COMPILE_OPTIONS="${COMPILE_OPTIONS} -DTRITON_ENABLE_GPU=OFF"
cmake $COMPILE_OPTIONS ..
make install -j$(npoc)
```

- CMakeLists.txt

```
# Copyright 2021, NVIDIA CORPORATION & AFFILIATES. All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
#  * Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
#  * Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in the
#    documentation and/or other materials provided with the distribution.
#  * Neither the name of NVIDIA CORPORATION nor the names of its
#    contributors may be used to endorse or promote products derived
#    from this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ``AS IS'' AND ANY
# EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
# PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
# CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
# EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
# PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
# PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
# OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
# OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
cmake_minimum_required(VERSION 3.17)
project(tutorialmindiebackend LANGUAGES C CXX)
#
# Options
#
# Must include options required for this project as well as any
# projects included in this one by FetchContent.
#
# GPU support is disabled by default because mindie backend
# doesn't use GPUs.
```



```
#
option(TRITON_ENABLE_GPU "Enable GPU support in backend" OFF)
option(TRITON_ENABLE_STATS "Include statistics collections in backend" ON)
set(TRITON_COMMON_REPO_TAG "main" CACHE STRING "Tag for triton-inference-server/common
repo")
set(TRITON_CORE_REPO_TAG "main" CACHE STRING "Tag for triton-inference-server/core repo")
set(TRITON_BACKEND_REPO_TAG "main" CACHE STRING "Tag for triton-inference-server/backend
repo")
if(NOT CMAKE_BUILD_TYPE)
  set(CMAKE_BUILD_TYPE Release)
endif()
add_compile_options(-std=c++17)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_SKIP_RPATH TRUE)
if (USE_CXX11_ABI)
  set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -D_GLIBCXX_USE_CXX11_ABI=1")
else()
  set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -D_GLIBCXX_USE_CXX11_ABI=0")
endif()
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -pipe -fstack-protector-all")
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra")
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wno-deprecated-copy")
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fPIC -Wl,--build-id=none")
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wl,-z,relro,-z,now")
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -pie -fexceptions")
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -ftrapv -s")
# add_definitions(-D_GLIBCXX_USE_C99=1)
#
# Dependencies
#
# FetchContent requires us to include the transitive closure of all
# repos that we depend on so that we can override the tags.
#
include(FetchContent)
FetchContent_Declare(
  repo-common
  GIT_REPOSITORY https://github.com/triton-inference-server/common.git
  GIT_TAG ${TRITON_COMMON_REPO_TAG}
  GIT_SHALLOW ON
)
FetchContent_Declare(
  repo-core
  GIT_REPOSITORY https://github.com/triton-inference-server/core.git
  GIT_TAG ${TRITON_CORE_REPO_TAG}
  GIT_SHALLOW ON
)
FetchContent_Declare(
  repo-backend
  GIT_REPOSITORY https://github.com/triton-inference-server/backend.git
  GIT_TAG ${TRITON_BACKEND_REPO_TAG}
  GIT_SHALLOW ON
)
FetchContent_MakeAvailable(repo-common repo-core repo-backend)
#
# The backend must be built into a shared library. Use an ldscript to
# hide all symbols except for the TRITONBACKEND API.
#
set(ASCEND_DRIVER_DIR /usr/local/Ascend/driver)
LINK_DIRECTORIES(
  $ENV{TRITON_HOME_PATH}/lib
  $ENV{MINDIE_LLM_HOME_PATH}/lib
)
INCLUDE_DIRECTORIES(
  $ENV{MINDIE_LLM_HOME_PATH}/include
  $ENV{MINDIE_LLM_HOME_PATH}/include/llm_manager
  $ENV{MINDIE_LLM_HOME_PATH}/include/llm_manager/utils
  ${ASCEND_DRIVER_DIR}/kernel/libc_sec/include
  $ENV{TRITON_HOME_PATH}/include
```



```
)
file(GLOB_RECURSE SOURCE_FILES "${CMAKE_CURRENT_LIST_DIR}/src/*.cc")
add_library(triton-mindie-backend SHARED ${SOURCE_FILES})
add_library(
  TutorialmindieBackend::triton-mindie-backend ALIAS triton-mindie-backend
)
target_include_directories(
  triton-mindie-backend
  PRIVATE
    ${CMAKE_CURRENT_SOURCE_DIR}/src
)
target_compile_features(triton-mindie-backend PRIVATE cxx_std_11)
target_link_libraries(
  triton-mindie-backend
  PRIVATE
    triton-core-serverapi   # from repo-core
    triton-core-backendapi  # from repo-core
    triton-core-serverstub  # from repo-core
    triton-backend-utils    # from repo-backend
    mindie_llm_manager      # from mindie-llm
)
set_target_properties(
  triton-mindie-backend PROPERTIES
  OUTPUT_NAME triton_mindie
)
# Install
install(
  TARGETS
  triton-mindie-backend
  DESTINATION $ENV{TRITON_HOME_PATH}/backends/mindie
)
```

7.2 vLLM 基于 Text Generator 接口开发指南

7.2.1 快速介绍

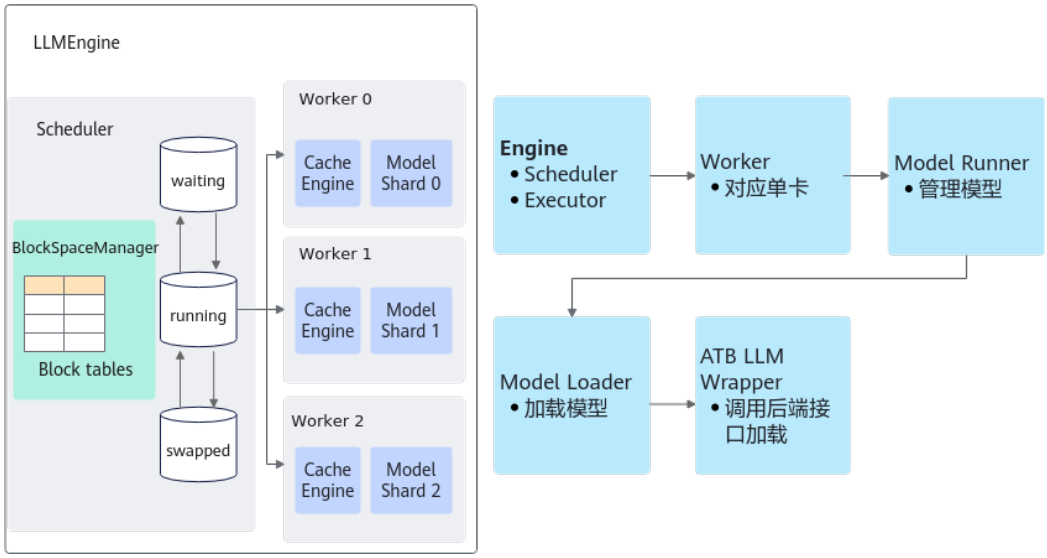
vLLM 简介

vLLM是一个为大语言模型提供高吞吐量和内存高效推理服务的开源引擎，使用PageAttention技术高效实现注意力Key和Value的内存管理，并且支持包括量化模型推理、Multi-LoRA、投机推理和SplitFuse等众多推理特性。作为一个Python编写的开源大语言模型推理框架，vLLM因其易用性与高效性得到了广泛的应用。

vLLM 框架适配昇腾整体方案介绍

vLLM框架在昇腾环境适配的整体方案为上层运行vLLM框架原生的逻辑，包括请求调度、Batch组建、Ray分布式拉起多卡服务等；下层模型推理与后处理通过MindIE LLM提供的GeneratorTorch统一接口接入MindIE模型仓统一进行管理，实现加速库整图模式的模型推理加速。

vLLM框架下层模型推理对接Text Generator接口的基本方式为实例化Text Generator中的GeneratorTorch类，继而通过该类的实例对象的forward_tensor和sample函数分别去使用MindIE LLM的模型推理和后处理功能。



支持版本特性与模型

表 7-17 支持的版本特性及模型

支持的vLLM版本	浮点	量化	SplitFuse	LoRA	多模态
0.3.3	同MindIE LLM	同MindIE LLM	-	同MindIE LLM	-
0.4.2	同MindIE LLM	同MindIE LLM	同MindIE LLM	同MindIE LLM	Qwen-VL

7.2.2 适配说明

7.2.2.1 vLLM 0.3.3 版本昇腾框架适配说明

vllm_npu_0.3.3版本的目录结构如下所示：

```
.
├── cover
│   ├── vllm
│   │   └── __init__.py
│   ├── requirements.txt
│   └── setup.py
├── examples
│   ├── start_server.sh
│   ├── test_offline.py
│   └── test_offline.sh
├── install.sh
└── vllm_npu
    ├── requirements.txt
    ├── setup.py
    ├── tests
    │   ├── models
    │   │   ├── __init__.py
    │   │   └── test_models.py
```

```
    └── sampler
        └── test_sampler.py
└── vllm_npu
    ├── config.py
    ├── core
    │   ├── __init__.py
    │   └── scheduler.py
    ├── engine
    │   ├── __init__.py
    │   ├── llm_engine.py
    │   └── ray_utils.py
    ├── __init__.py
    ├── model_executor
    │   ├── ascend_model_loader.py
    │   ├── __init__.py
    │   ├── layers
    │   │   ├── __init__.py
    │   │   └── sampler.py
    │   ├── models
    │   │   ├── ascend
    │   │   │   ├── __init__.py
    │   │   │   └── mindie_llm_wrapper.py
    │   │   └── __init__.py
    │   └── utils.py
    ├── npu_adaptor.py
    ├── utils.py
    └── worker
        ├── ascend_worker.py
        ├── cache_engine.py
        ├── __init__.py
        └── model_runner.py
```

vllm_npu_0.3.3版本中主要修改了core、engine、model_executor、worker四个模块，与vllm原生框架中的同名模块一一对应进行热替换适配。

表 7-18 各模块修改内容介绍

模块	简介
core	该模块中对于 “num_batched_tokens”的计算与后端有所不同，MindIE后端的实现方法与0.4.2版本一致，因此在此处需要修改该变量计算逻辑。
engine	该模块替换了部分内容，主要是为了确保vLLM框架可以正确获取到后续的worker与model_executor模块。例如：在ray模块启动时新增指定 “num_gpus”参数为 “parallel_config.world_size”，解决ray模块启动问题；并且在DEVICE_TO_WORKER_MODULE_MAP常量字典中新增键值对："npu"、 "vllm_npu.worker.ascend_worker"，确保在启动worker时可以正确使用ascend_worker。

模块	简介
model_executor	该模块负责与MindIE LLM模型的推理与后处理进行对接，包含两个子模块：“layers”和 “models”。其中，“layers” 模块主要负责后处理操作，而“models”模块则专注于模型推理。在“models”模块中，编写了一个名为 “MindIELlmWrapper”的类。这个类用于实例化 MindIE LLM提供的 “GeneratorTorch”统一接口，并从vLLM 原生框架的数据结构中提取出MindIE LLM所需的模型推理参数，进而调用统一接口进行模型推理。此外，该类还实现了在进行预热（warmup）操作时所需的虚拟数据的构造过程。 同时添加了ascend_model_loader.py文件，为后续的“ModelRunner”加载模型提供“get_model”方法。
worker	该模块覆写了“Worker”类与“ModelRunner”类中的部分函数，在ascend_worker.py和model_runner.py中与“MindIELlmWrapper”交互，确保模型的加载、预热、推理和后处理等过程能够正确在后端执行。 此外，修改了原生框架中“CacheEngine”的“_allocate_kv_cache”函数，确保与后端接收的参数形式一致。

推理流程与部分关键代码的说明

在vLLM中从启动引擎到完成推理，有如下步骤：

步骤1 输入模型架构名和权重路径：首先提供模型的基本信息。

步骤2 初始化引擎参数：

“model str”-> “EngineArgs()”->“LLMEngine.from_engine_args()”->“LLMEngine.init()”

步骤3 配置模型：

“ModelConfig()”：如果设置了“VLLM_USE_MODELSCOPE”变量，则从ModelScope hub 下载对应的模型，并设置下载路径为模型权重路径。

步骤4 初始化工作器：“LLMEngine._init_workers()”->“Worker.init_model()”：在这一步骤中，设置NPU环境、分布式配置和随机种子等。“Worker.load_model()”->“ModelRunner.load_model()”->“vllm.model_executor.model_loader.get_model()”

```
def load_model(self):
    # Add an attribute dynamically, not graceful, but useful
    self.model_config.mindie_config = {
        "backend_type": BackendType.ATB,
        "model_id": self.model_config.model,
        "rank": self.rank,
        "local_rank": self.local_rank,
        "world_size": self.parallel_config.world_size,
        "npu_device_id": self.local_rank,
    }
    self.model_runner.load_model()
    del self.model_config.mindie_config
```

步骤5 加载模型：

“ascend_model_loader.get_model()”->“mindie_llm_wrapper.load_weights()”->“mindie_llm.text_generator.adapter.GeneratorTorch(model_config)”，由MindIE LLM实现模型加载的逻辑。

```
def load_weights(self, load_format: Literal["auto", "safetensors", "pt"] = "auto"):
    """Load model weights.
    Args:
        load_format (Literal[auto, safetensors, pt], optional): The format of weights. Defaults to "auto".
    Raises:
        ValueError: Format not supported.
    """
    if load_format not in ["auto", "safetensors", "pt"]:
        raise ValueError("load-format support [safetensors, pt]")
    self.weight_dtype = torch.get_default_dtype()
    torch.set_default_dtype(torch.float32)
    # Replacing the deprecated method with GeneratorTorch
    self.atb_model = GeneratorTorch(self.mindie_config)
    self.sampler = Sampler(self.atb_model)
    self.sampler.logits_as_hidden_states = True
    torch.set_default_dtype(self.weight_dtype)
```

步骤6 创建模型实例：

回到“model_loader.get_model()”函数中根据类创建模型实例。如果存在量化方法，也在此实例化。

步骤7 实际推理：

“LLMEngine.step”->“worker.execute_model”->“model_runner.execute_model”->“mindie_llm_wrapper.foward” -> MindIE后端。其中在 “model_runner”层还需要在“prepare_input_tensor”及其私有方法中对输入进来的数据进行处理。

同样，后处理部分也是由MindIE后端来进行处理与看护。

```
def forward(
    self,
    input_ids: torch.Tensor,
    positions: torch.Tensor,
    kv_caches: List[KVCache],
    input_metadata: InputMetadata,
    lora_requests: List[LoRARequest],
) -> torch.Tensor:
    if kv_caches[0][0] is None:
        kv_caches, block_tables, slots = self.create_dummy_kv_cache(input_metadata, input_ids)
    else:
        if input_metadata.is_prompt:
            block_tables = torch.tensor([0], dtype=torch.int32, device="npu")
        else:
            block_tables = input_metadata.block_tables
        slots = input_metadata.slot_mapping
    if input_metadata.is_prompt:
        input_lengths = input_metadata.prompt_lens.to(torch.int32)
        max_seq_len = int(input_metadata.prompt_lens.max())
        lm_head_indices = (input_metadata.prompt_lens.cumsum(dim=-1) - 1).to(torch.int64)
    else:
        input_lengths = input_metadata.context_lens
        max_seq_len = input_metadata.max_context_len
        lm_head_indices = None
    adapter_ids = [lora_request.lora_name if lora_request else None for lora_request in lora_requests]
    logits = self.atb_model.forward_tensor(
        input_ids,
        positions,
        input_metadata.is_prompt,
        kv_caches,
        block_tables,
        slots,
```



```
            input_lengths,
            max_seq_len,
            lm_head_indices,
            adapter_ids=adapter_ids,  # batch_size len
        )
        return logits
    def sample(
        self,
        hidden_states: torch.Tensor,
        sampling_metadata: SamplingMetadata,
    ) -> Optional[SamplerOutput]:
        # hidden_states is logits
        next_tokens = self.sampler(hidden_states, sampling_metadata)
        return next_tokens
```

----结束

对应修改量

表 7-19 对应修改量

重要组件	代码量	作用
Scheduler	~200行	修改“num_batched_tokens”计算逻辑。
Cache Engine	~30行	调整KV Cache参数结构，兼容底层接口。
Worker	~250行	构建后端统一接口所需的配置参数，并且初始化昇腾硬件环境。
Model Loader	~50行	重新编写get_model函数，对接下层调用。
Model Runner	~300行	传递配置参数，替换get_model，修改profile_run中对KV Cache的初始化方式，修改LoRA对应参数传递方式。
MindIE LLM Wrapper	~150行	负责加载、生成虚拟KV Cache、推理、后处理操作与后端的对接。
Sampler	~300行	从SamplingMetadata重构参数，传递至后端Sampler，替代vLLM自有后处理方式，并将结果恢复至vLLM结构。

说明

除此之外，还有少部分文件需要修改，例如：config.py和utils.py等文件需要修改以解决少量在运行时出现的问题。

7.2.2.2 vLLM 0.4.2 版本昇腾框架适配说明

vllm_npu_0.4.2版本的目录结构如下所示：

```
vllm_npu
|-- __init__.py
|-- attention
|   |-- __init__.py
|   |-- backends.py
|   |-- selector.py
|-- config.py
|-- core
```

```
|   |-- __init__.py
|-- engine
|   |-- __init__.py
|   |-- ascend_engine.py
|   |-- async_ascend_engine.py
|-- executor
|   |-- __init__.py
|   |-- ascend_executor.py
|   |-- ascend_ray_executor.py
|   |-- ray_utils.py
|-- model_executor
|   |-- __init__.py
|   |-- ascend_model_loader.py
|   |-- layers
|   |   |-- __init__.py
|   |   |-- ascend_sampler.py
|   |-- models
|       |-- __init__.py
|       |-- ascend
|           |-- __init__.py
|           |-- mindie_llm_wrapper.py
|-- npu_adaptor.py
|-- usage
|   |-- __init__.py
|   |-- usage_lib.py
|-- utils.py
|-- worker
    |-- __init__.py
    |-- ascend_model_runner.py
    |-- ascend_worker.py
    |-- cache_engine.py
```

vllm_npu_0.4.2版本中重写了attention、engine、executor、model_executor、usage、worker六个模块，与vLLM原生框架中的同名模块一一对应进行热替换适配。

- attention模块：

  重写了vLLM框架中的AttentionBackend类，对昇腾环境下对接MindIE LLM所需要的Attention计算数据以及KV Cache的shape等关键信息进行了定义，并在该模块的初始化文件中对原框架的get_atten_backend函数进行了热替换，从而使得框架在昇腾环境下会运行该模块中的attention后端类。

- engine模块：

  该模块主要重写了vLLM引擎的from_engine_args函数，vLLM 0.4.2版本中引擎会通过该类方法进行实例化，并在其中根据运行环境信息选择对应的不同executor；这里加入了新的判断逻辑分支，当检测到运行在昇腾环境下时，引擎会选择vllm_npu补丁包中定义的AscendExecutor和RayAscendExecutor类分别去进行单卡推理和多卡推理。另外，这里对vllm原生框架的离线同步推理引擎LLMEngine和在线异步推理引擎AsyncLLMEngine的from_engine_args函数分别进行了重写替换，替换操作发生在该模块的初始化文件中。

- executor模块：

  该模块中主要实现了四个executor类，其中AscendExecutor和AscendExecutorAsync用于单卡环境的同步和异步调用模式下的推理，RayAscendExecutor和RayAscendExecutorAsync用于多卡ray分布式环境的同步和异步调用模式下的推理。

  此外，在ray_utils.py中对initialize_ray_cluster函数进行了重写，主要是因为在昇腾的npu环境下ray无法自动识别到npu的数量，因此需要手动显示指定。

- model_executor模块：
  - 该模块为实际对接MindIE LLM模型推理与后处理的位置，其中包括layers模块和models模块，分别对应后处理和模型推理。

在models模块中，编写了MindIELlmWrapper类，在该类中会对MindIE LLM提供的GeneratorTorch统一接口进行实例化操作，并从vLLM原生框架的数据结构中拆解出MindIE LLM所需要的模型推理参数，从而传给统一接口调用模型推理服务；另外，在进行warmup操作时使用的fake data构造操作也在该类中实现。实现代码如下所示：

```
class MindIELlmWrapper(nn.Module):
    def __init__(self, mindie_model_config, linear_method=None, lora_config=None):
        super(MindIELlmWrapper, self).__init__()

        self.mindie_model_config = mindie_model_config
        self.rank = mindie_model_config['rank']
        self.local_rank = mindie_model_config['local_rank']
        self.npu_id = self.local_rank
        self.world_size = mindie_model_config['world_size']
        self.mindie_model = None
        self.sampler = None
    def forward(
            self,
            input_ids: torch.Tensor,
            positions: torch.Tensor,
            kv_caches: List[KVCache],
            attn_metadata: AttentionMetadata,
    ) -> torch.Tensor:
        is_prompt = attn_metadata.num_prefill_tokens > 0

        if kv_caches[0][0] is None:
            kv_caches, block_tables, slots = self.create_dummy_kv_cache(attn_metadata, input_ids)
        else:
            if is_prompt:
                block_tables = torch.tensor([0], dtype=torch.int32, device="npu")
            else:
                block_tables = attn_metadata.decode_metadata.block_tables
            slots = attn_metadata.slot_mapping
        if is_prompt:
            input_lengths = attn_metadata.prefill_metadata.seq_lens_tensor.to(torch.int32)
            max_seq_len = int(attn_metadata.prefill_metadata.seq_lens_tensor.max())
            lm_head_indices = (attn_metadata.prefill_metadata.seq_lens_tensor.cumsum(dim=-1) -
1).to(torch.int64)
        else:
            input_lengths = attn_metadata.decode_metadata.seq_lens_tensor
            max_seq_len = attn_metadata.decode_metadata.max_seq_len
            lm_head_indices = None

        logits = self.mindie_model.forward_tensor(input_ids, positions, is_prompt, kv_caches,
block_tables, slots,
                                  input_lengths, max_seq_len, lm_head_indices)
        return logits
    def compute_logits(self, hidden_states: torch.Tensor,
                       sampling_metadata: SamplingMetadata) -> torch.Tensor:
        return hidden_states
    def sample(
        self,
        logits: torch.Tensor,
        sampling_metadata: SamplingMetadata,
    ) -> Optional[SamplerOutput]:
        # hidden_states is logits
        next_tokens = self.sampler(logits, sampling_metadata)
        return next_tokens
    def load_weights(self,
                     model_name_or_path: str,
                     cache_dir: Optional[str] = None,
                     load_format: str = "auto",
                     revision: Optional[str] = None):
        if load_format not in ['auto', 'safetensors', 'pt']:
            raise ValueError('load-format support [safetensors, pt]')
        self.weight_dtype = torch.get_default_dtype()
        torch.set_default_dtype(torch.float32)
        self.mindie_model = GeneratorTorch(self.mindie_model_config)
```



```
        self.sampler = AscendSampler(self.mindie_model)
        torch.set_default_dtype(self.weight_dtype)
    # when warmup, create dummy kvcache, block_tables, slot_mapping
    def create_dummy_kv_cache(self, attn_metadata, input_ids):
        dummy_block_num = 1
        dummy_block_size = 128
        model_runner = self.mindie_model.model_wrapper.model_runner
        kv_cache = [
            (
                torch.empty(
                    (dummy_block_num, dummy_block_size, model_runner.num_kv_heads,
model_runner.head_size),
                    dtype=self.weight_dtype,
                    device="npu",
                ),
                torch.empty(
                    (dummy_block_num, dummy_block_size, model_runner.num_kv_heads,
model_runner.head_size),
                    dtype=self.weight_dtype,
                    device="npu",
                ),
            )
            for _ in range(model_runner.num_layers)
        ]
        max_s = max(attn_metadata.prefill_metadata.seq_lens_tensor)
        max_need_block = math.ceil(max_s / dummy_block_size)
        batch_size = len(attn_metadata.prefill_metadata.seq_lens_tensor)
        block_tables = torch.zeros(batch_size, max_need_block, dtype=int, device="npu")
        slot = [i for i in range(dummy_block_size)]
        slots = []
        warm_up_len = len(input_ids)
        while warm_up_len > 0:
            if warm_up_len > dummy_block_size:
                slots.extend(slot)
                warm_up_len -= dummy_block_size
            else:
                slots.extend(slot[:warm_up_len])
                warm_up_len = 0
        slots = torch.tensor(slots, dtype=torch.long, device="npu")
        return kv_cache, block_tables, slots
```

- 在layers模块中，编写实现了AscendSampler类，进行vLLM原生框架的数据结构与模型仓底层数据结构之间的对接工作，实现代码如下所示：

```
class AscendSampler(nn.Module):
    def __init__(self, mindie_model):
        super().__init__()
        self.mindie_model = mindie_model
        self.include_gpu_probs_tensor = False
    def forward(
        self,
        logits: torch.Tensor,
        sampling_metadata: SamplingMetadata,
    ) -> Optional[SamplerOutput]:
        _, vocab_size = logits.shape
        mindie_sampling_data, mindie_sampling_param = self.construct_data(sampling_metadata,
vocab_size)
        probs = torch.softmax(logits, dim=-1, dtype=torch.float)
        logprobs = torch.log_softmax(logits, dim=-1, dtype=torch.float)
        next_tokens = self.mindie_model.sample(
            logits,
            sampling_data=mindie_sampling_data,
            sampling_param=mindie_sampling_param,
        )

        sample_results, maybe_sampled_tokens_tensor = recover_data(
            sampling_metadata=sampling_metadata,
            sampled_tokens=next_tokens,
            logprobs=logprobs,
            include_gpu_probs_tensor=self.include_gpu_probs_tensor,
        )
```



```
    if self.include_gpu_probs_tensor:
        if maybe_sampled_tokens_tensor is None:
            raise RuntimeError("maybe_sampled_tokens_tensor is None")
        on_device_tensors = (probs, logprobs, maybe_sampled_tokens_tensor)
    else:
        on_device_tensors = None
    # Get the logprobs query results.
    prompt_logprobs, sample_logprobs = _get_logprobs(
        logprobs, sampling_metadata, sample_results)
    return _build_sampler_output(sample_results,
                                 sampling_metadata,
                                 prompt_logprobs,
                                 sample_logprobs,
                                 on_device_tensors=on_device_tensors)
def construct_data(
    self,
    sampling_metadata: SamplingMetadata,
    vocab_size: int,
) -> Tuple[SamplingData, SamplingParam]:
    all_input_tokens: List[List[int]] = []
    prompt_tokens: List[List[int]] = []
    output_tokens: List[List[int]] = []
    top_ks: List[int] = []
    temperatures: List[float] = []
    top_ps: List[float] = []
    min_ps: List[float] = []
    presence_penalties: List[float] = []
    frequency_penalties: List[float] = []
    repetition_penalties: List[float] = []
    sampling_seeds: List[int] = []
    sample_indices: List[int] = []
    do_samples: List[bool] = []  # To Do
    do_penalties = False
    do_top_p_top_k = False
    do_min_p = False
    greedy_flag = False

    if sampling_metadata.seq_groups is None:
        raise RuntimeError("sampling_metadata.seq_group is None, no data received.")
    for seq_group in sampling_metadata.seq_groups:
        do_samples.append(seq_group.do_sample)
        seq_ids = seq_group.seq_ids
        sampling_params = seq_group.sampling_params
        temperature = sampling_params.temperature
        p = sampling_params.presence_penalty
        f = sampling_params.frequency_penalty
        r = sampling_params.repetition_penalty
        top_p = sampling_params.top_p
        min_p = sampling_params.min_p
        is_greedy = sampling_params.sampling_type == SamplingType.GREEDY
        seed = sampling_params.seed
        if seed is None:
            if is_greedy:
                seed = 0
            else:
                lo, hi = torch.iinfo(torch.long).min, torch.iinfo(torch.long).max
                seed = random.randint(lo, hi)
        if is_greedy:
            greedy_flag = True
        # k should not be greater than the vocab size.
        top_k = min(sampling_params.top_k, vocab_size)
        top_k = vocab_size if top_k == -1 else top_k
        if temperature < _SAMPLING_EPS:
            temperature = 1.0
        if not do_top_p_top_k and (top_p < 1.0 - _SAMPLING_EPS
                                   or top_k != vocab_size):
            do_top_p_top_k = True
        if not do_min_p and min_p > _SAMPLING_EPS:
            do_min_p = True
```



```
        if not do_penalties:
            if abs(p) >= _SAMPLING_EPS:
                do_penalties = True
            elif abs(f) >= _SAMPLING_EPS:
                do_penalties = True
            elif abs(r - 1.0) >= _SAMPLING_EPS:
                do_penalties = True
        is_prompt = seq_group.is_prompt
        if (seq_group.is_prompt
                and sampling_params.prompt_logprobs is not None):
            # For tokens in the prompt that we only need to get
            # their logprobs
            query_len = seq_group.query_len
            if query_len is None:
                raise RuntimeError("query_len is None")
            prefill_len = len(seq_group.prompt_logprob_indices)
            temperatures += [temperature] * prefill_len
            sampling_seeds += [seed] * prefill_len
            top_ps += [top_p] * prefill_len
            top_ks += [top_k] * prefill_len
            min_ps += [min_p] * prefill_len
            presence_penalties += [0] * prefill_len
            frequency_penalties += [0] * prefill_len
            repetition_penalties += [1] * prefill_len
            prompt_tokens.extend([] for _ in range(prefill_len))
            output_tokens.extend([] for _ in range(prefill_len))
            all_input_tokens.extend([] for _ in range(prefill_len))
        if seq_group.do_sample:
            sample_lens = len(seq_group.sample_indices)
            if sample_lens != len(seq_ids):
                raise ValueError("sample_lens != len(seq_ids)")
            for seq_id in seq_ids:
                seq_data = seq_group.seq_data[seq_id]
                prompt_tokens.append(seq_data.prompt_token_ids)
                output_tokens.append(seq_data.output_token_ids)
                all_input_tokens.append(seq_data.prompt_token_ids + seq_data.output_token_ids)
            temperatures += [temperature] * len(seq_ids)
            sampling_seeds += [seed] * len(seq_ids)
            top_ps += [top_p] * len(seq_ids)
            top_ks += [top_k] * len(seq_ids)
            min_ps += [min_p] * len(seq_ids)
            presence_penalties += [p] * len(seq_ids)
            frequency_penalties += [f] * len(seq_ids)
            repetition_penalties += [r] * len(seq_ids)
    repetition_penalties = np.array(repetition_penalties, dtype=np.float32)
    frequency_penalties = np.array(frequency_penalties, dtype=np.float32)
    presence_penalties = np.array(presence_penalties, dtype=np.float32)
    temperatures = np.array(temperatures, dtype=np.float32)
    top_ks = np.array(top_ks, dtype=np.int32)
    top_ps = np.array(top_ps, dtype=np.float32)
    sampling_seeds = np.array(sampling_seeds)
    do_samples = np.array(do_samples)
    max_tokens_len = max([len(tokens) for tokens in all_input_tokens], default=0)
    padded_all_input_tokens = [
        tokens + [vocab_size] * (max_tokens_len - len(tokens))
        for tokens in all_input_tokens
    ]
    padded_all_input_tokens = np.array(padded_all_input_tokens, dtype=np.int32)
    output_max_len = max([len(tokens) for tokens in output_tokens], default=0)
    padded_output_tokens = [
        tokens + [vocab_size] * (output_max_len - len(tokens))
        for tokens in output_tokens
    ]
    padded_output_tokens = np.array(padded_output_tokens, dtype=np.int32)
    all_input_ids_tensor = _to_tensor(
        padded_all_input_tokens,
        torch.int32
    ) if padded_all_input_tokens is not None else None
    output_ids_tensor = _to_tensor(
```



```
            padded_output_tokens,
            torch.int32
        ) if padded_output_tokens is not None else None
        mindie_sampling_data = SamplingData(
            all_input_ids=all_input_ids_tensor,
            output_ids=output_ids_tensor
        )
        if greedy_flag:
            mindie_sampling_param = None
        else:
            mindie_sampling_param = SamplingParam.from_numpy(
                repetition_penalty=repetition_penalties,
                frequency_penalty=frequency_penalties,
                presence_penalty=presence_penalties,
                temperature=temperatures,
                top_k=top_ks,
                top_p=top_ps,
                seed=sampling_seeds,
                do_sample=do_samples,
                to_tensor=_to_tensor,
            )
        return (mindie_sampling_data, mindie_sampling_param)
def recover_data(
    sampling_metadata: SamplingMetadata,
    sampled_tokens: np.ndarray,
    logprobs: torch.Tensor,
    include_gpu_probs_tensor: bool,
) -> Tuple[SampleResultType, Optional[torch.Tensor]]:
    categorized_seq_group_ids: Dict[SamplingType,
                                    List[int]] = {t: []
                                                  for t in SamplingType}
    categorized_sample_indices = sampling_metadata.categorized_sample_indices
    for i, seq_group in enumerate(sampling_metadata.seq_groups):
        sampling_params = seq_group.sampling_params
        sampling_type = sampling_params.sampling_type
        categorized_seq_group_ids[sampling_type].append(i)
    sample_results_dict: Dict[int, Tuple[List[int], List[int]]] = {}
    sample_metadata = {}
    # Create output tensor for sampled token ids.
    if include_gpu_probs_tensor:
        sampled_token_ids_tensor = torch.empty(logprobs.shape[0],
                                               1,
                                               dtype=torch.long,
                                               device=logprobs.device)
    else:
        sampled_token_ids_tensor = None
    for sampling_type in SamplingType:
        sample_indices = categorized_sample_indices[sampling_type][:, 0]
        num_tokens = len(sample_indices)
        if num_tokens == 0:
            continue
        seq_group_id = categorized_seq_group_ids[sampling_type]
        seq_groups = [sampling_metadata.seq_groups[i] for i in seq_group_id]
        sample_metadata[sampling_type] = (seq_group_id, seq_groups)
    for sampling_type in SamplingType:
        if sampling_type not in sample_metadata:
            continue
        (seq_group_id, seq_groups) = sample_metadata[sampling_type]
        if sampling_type in (SamplingType.GREEDY, SamplingType.RANDOM,
SamplingType.RANDOM_SEED):
            sample_results = normal_wrap(seq_groups, sampled_tokens)
        elif sampling_type == SamplingType.BEAM:
            sample_results = beam_wrap(seq_groups, sampled_tokens)
        sample_results_dict.update(zip(seq_group_id, sample_results))
    sample_results = [
        sample_results_dict.get(i, ([], []))
        for i in range(len(sampling_metadata.seq_groups))
    ]
    return sample_results, sampled_token_ids_tensor
```



```
def normal_wrap(
    selected_seq_groups: List[SequenceGroupToSample],
    samples: np.ndarray,
):
    samples = samples.tolist()
    sample_idx = 0
    results: SampleResultType = []
    for seq_group in selected_seq_groups:
        if not seq_group.do_sample:
            results.append(([], []))
            continue
        seq_ids = seq_group.seq_ids
        num_parent_seqs = len(seq_ids)
        parent_ids = list(range(num_parent_seqs))
        next_token_ids = [samples[sample_idx]]
        results.append((next_token_ids, parent_ids))
        sample_idx += num_parent_seqs
    return results
```

另外，该模块中重写了vLLM框架的get_model和get_architecture_class_name函数，从而将MindIELlmWrapper类引入到vLLM框架中。

- usage模块：

  对vLLM框架中该模块里的UsageMessage类的_report_usage_once成员函数进行了重写，修改了其中的torch.cuda.get_device_properties函数的使用方式，该函数目前在昇腾环境上的使用方式和GPU环境上有所差异。

- worker模块：

  实现了AscendWorker类，以供executor模块中的executor类进行调用；实现了AscendModelRunner类，在AscendWorker中进行调用。

  替换原生框架中CacheEngine的_allocate_kv_cache函数，主要是对生成kv_cache的数据格式进行了修改，从Torch.tensor修改为Tuple[torch.Tensor, torch.Tensor]。

  AscendModelRunner类继承自原生框架中ModelRunner类，主要是为了对原生的load_model，execute_model和profile_run函数进行重写：vLLM新版本中执行模型调用时分为了先调用模型生成hidden_states，再使用一个process处理hidden_statesd得到logits，再进行最后的sample操作得到结果；而在MindIE模型仓中前两步操作是通过模型调用一步完成的，因此在这里进行了修改；profile_run函数的修改主要是为了构造warmup时使用的fake data。

除了上述的六个模块的适配外，还有一些主模块外的Python文件里的函数需要热替换，包括config.py中的DeviceConfig类中引入了NPU作为device_type，在utils.py文件中引入了is_ascend()函数用于检测当前运行环境是否为昇腾环境。最后，在npu_adaptor.py，对vLLM原框架中导入的一些昇腾环境下不具备的包（例如预编译的cuda算子、triton等）进行了屏蔽操作。

- 多模态模型Qwen-VL支持：

  为了适配多模态模型Qwen-VL的推理，需要对vLLM框架中离线推理接口类LLM的generate函数进行修改，以及对AscendModelRunner中构造warmup假数据的部分进行修改，并且自己定义了Qwen-VL的tokenizer需要的多模态输入数据格式，同时定义了一个新的MindIETokenizer类来对接MindIE LLM的前处理，具体的适配细节如下：

  首先需要在vllm_npu包里新建entrypoints和transformers_utils两个模块和一个sequence.py文件，entrypoints模块下新建__init__.py和llm.py两个文件，transformers_utils模块下新建__init__.py和mindie_tokenizer.py两个文件。

  a. 在vllm_npu/sequence.py文件中添加如下新类用于定义Qwen-VL的多模态数据。


```
from typing import Dict, List
class MultiModalData:
    """Multi modal input for MindIE LLM.
    Args:
        data_list: List of input data in the format of dict. For example:
        [
            {"image": url_of_image1},
            {"image": url_of_image1},
            {"text": input_prompt1},
            {"text": input_prompt1}
        ]
    """
    def __init__(self, data_list: List[Dict[str, str]]):
        self.data_list = data_list
```

b. 在vllm_npu/transformers_utils/mindie_tokenizer.py文件中引入了新类MindIETokenizer来对接MindIE LLM的前处理。

```
from typing import List, Optional
from transformers import PreTrainedTokenizer
from vllm.transformers_utils.tokenizer_group.base_tokenizer_group import (
    BaseTokenizerGroup)
from vllm.transformers_utils.tokenizer import get_lora_tokenizer, get_lora_tokenizer_async
from vllm.lora.request import LoRARequest
from atb_llm.runner.tokenizer_wrapper import TokenizerWrapper
class MindIETokenizer(BaseTokenizerGroup):
    """A group of tokenizers that can be used for LoRA adapters."""
    def __init__(self, model_path: str):
        self.tokenizer_wrapper = TokenizerWrapper(model_path)
        self.tokenizer = self.tokenizer_wrapper.tokenizer
        self.lora_tokenizers = None
        self.enable_lora = False
    def ping(self) -> bool:
        """Check if the tokenizer group is alive."""
        return True
    def get_max_input_len(self,
                          lora_request: Optional[LoRARequest] = None
                          ) -> Optional[int]:
        """Get the maximum input length for the LoRA request."""
        return 0
    def encode(self,
               prompt: str,
               request_id: Optional[str] = None,
               lora_request: Optional[LoRARequest] = None) -> List[int]:
        tokenizer = self.get_lora_tokenizer(lora_request)
        return tokenizer.encode(prompt)
    async def encode_async(
            self,
            prompt: str,
            request_id: Optional[str] = None,
            lora_request: Optional[LoRARequest] = None) -> List[int]:
        tokenizer = await self.get_lora_tokenizer_async(lora_request)
        return tokenizer.encode(prompt)
    def get_lora_tokenizer(
            self,
            lora_request: Optional[LoRARequest] = None
    ) -> "PreTrainedTokenizer":
        if not lora_request or not self.enable_lora:
            return self.tokenizer
        if lora_request.lora_int_id not in self.lora_tokenizers:
            tokenizer = (get_lora_tokenizer(
                lora_request, **self.tokenizer_config) or self.tokenizer)
            self.lora_tokenizers.put(lora_request.lora_int_id, tokenizer)
            return tokenizer
        else:
            return self.lora_tokenizers.get(lora_request.lora_int_id)
    async def get_lora_tokenizer_async(
            self,
            lora_request: Optional[LoRARequest] = None
    ) -> "PreTrainedTokenizer":
        if not lora_request or not self.enable_lora:
```



```
            return self.tokenizer
        if lora_request.lora_int_id not in self.lora_tokenizers:
            tokenizer = (await get_lora_tokenizer_async(
                lora_request, **self.tokenizer_config) or self.tokenizer)
            self.lora_tokenizers.put(lora_request.lora_int_id, tokenizer)
            return tokenizer
        else:
            return self.lora_tokenizers.get(lora_request.lora_int_id)
```

c. 在vllm_npu/entrypoints/llm.py文件里重新定义generate函数，引入多模态数据的输入接口。

```
import torch
from typing import List, Optional, Union
from vllm.lora.request import LoRARequest
from vllm.outputs import RequestOutput
from vllm.sampling_params import SamplingParams
from vllm.transformers_utils.detokenizer import Detokenizer
from vllm_npu.transformers_utils import MindIETokenizer
from vllm_npu.sequence import MultiModalData

def generate(
    self,
    prompts: Optional[Union[str, List[str]]] = None,
    sampling_params: Optional[Union[SamplingParams,
                                    List[SamplingParams]]] = None,
    prompt_token_ids: Optional[List[List[int]]] = None,
    use_tqdm: bool = True,
    lora_request: Optional[LoRARequest] = None,
    multi_modal_data_list: Optional[List[MultiModalData]] = None,
) -> List[RequestOutput]:
    """Generates the completions for the input prompts.
    Args:
        prompts: A list of prompts to generate completions for.
        sampling_params: The sampling parameters for text generation. If
            None, we use the default sampling parameters.
            When it is a single value, it is applied to every prompt.
            When it is a list, the list must have the same length as the
            prompts and it is paired one by one with the prompt.
        prompt_token_ids: A list of token IDs for the prompts. If None, we
            use the tokenizer to convert the prompts to token IDs.
        use_tqdm: Whether to use tqdm to display the progress bar.
        lora_request: LoRA request to use for generation, if any.
        multi_modal_data_list: List of Multi modal data. Each element in this list
        is organized in the form of List[Dict[str, str]]
    Returns:
        A list of `RequestOutput` objects containing the generated
        completions in the same order as the input prompts.
    """
    if prompts is None and prompt_token_ids is None and multi_modal_data_list is None:
        raise ValueError("Either prompts or prompt_token_ids of multi_modal_data must be "
                         "provided.")
    if self.llm_engine.model_config.skip_tokenizer_init \
        and prompts is not None:
        raise ValueError("prompts must be None if skip_tokenizer_init "
                         "is True")
    if isinstance(prompts, str):
        # Convert a single prompt to a list.
        prompts = [prompts]
    if (prompts is not None and prompt_token_ids is not None
            and len(prompts) != len(prompt_token_ids)):
        raise ValueError("The lengths of prompts and prompt_token_ids "
                         "must be the same.")
    if multi_modal_data_list and self.llm_engine.tokenizer is None:
        self.llm_engine.tokenizer = MindIETokenizer(self.llm_engine.model_config.model)
        self.llm_engine.detokenizer = Detokenizer(self.llm_engine.tokenizer)
        self.llm_engine.output_processor.detokenizer = self.llm_engine.detokenizer
    if prompts is not None:
        num_requests = len(prompts)
    elif multi_modal_data_list is not None:
        num_requests = len(multi_modal_data_list)
```



```
    else:
        assert prompt_token_ids is not None
        num_requests = len(prompt_token_ids)
    if sampling_params is None:
        # Use default sampling params.
        sampling_params = SamplingParams()
    elif isinstance(sampling_params,
                    list) and len(sampling_params) != num_requests:
        raise ValueError("The lengths of prompts and sampling_params "
                         "must be the same.")
    # Add requests to the engine.
    for i in range(num_requests):
        prompt = prompts[i] if prompts is not None else None
        token_ids = None if prompt_token_ids is None else prompt_token_ids[
            i]
        if multi_modal_data_list:
            token_ids = self.llm_engine.tokenizer.tokenizer_wrapper.tokenize(
                multi_modal_data_list[i].data_list).tolist()
            # print(token_ids)
        self._add_request(
            prompt,
            sampling_params[i]
            if isinstance(sampling_params, list) else sampling_params,
            token_ids,
            lora_request=lora_request,
            # Get ith image while maintaining the batch dim.
            multi_modal_data=None,
        )
    outputs = self._run_engine(use_tqdm)
    for output in outputs:
        token_ids = output.outputs[0].token_ids
        token_ids_tensor = torch.tensor(token_ids, dtype=torch.int64)
        mindie_generated_text = self.llm_engine.tokenizer.tokenizer.decode(token_ids_tensor, False)
        output.outputs[0].text = mindie_generated_text
    return outputs
```

d. 修改vllm_npu/worker/ascend_model_runner.py文件，将其中从vLLM框架里导入的_prepare_fake_inputs去掉，重新定义该函数如下。

```
def _prepare_fake_inputs(
        seq_len: int, model_config: ModelConfig):
    """Prepare fake inputs for profile run."""
    if getattr(model_config.hf_config, "visual", None) is not None:
        img_start_id = model_config.hf_config.visual["image_start_id"]
        img_end_id = img_start_id + 1
        img_patch_id = img_start_id + 2
        fake_img_token_ids = [24669, 220, 16, 25, 151857, 120, 121] + \
            [img_patch_id] * 254 + [img_end_id, 198]
        img_token_nums = len(fake_img_token_ids)
        if seq_len < img_token_nums:
            raise ValueError(f"The number of max_model_len/max_num_seqs is smaller than the
img_token_nums({img_token_nums}) of Qwen-VL.")
        prompt_tokens = fake_img_token_ids + \
            [0] * (seq_len - img_token_nums)
    else:
        prompt_tokens = [0] * seq_len
    fake_image_input = None
    return SequenceData(prompt_tokens), fake_image_input
```

另外，调用该函数部分的代码更改为。

```
seq_data, fake_multi_modal_input = _prepare_fake_inputs(
                seq_len, self.model_config)
```

e. 在vllm_npu/entrypoints/__init__.py中进行generate函数的替换。

```
from vllm_npu.entrypoints.llm import generate
import vllm.entrypoints.llm as vllm_entry_llm
vllm_entry_llm.LLM.generate = generate
```

在vllm_npu/transformers_utils/__init__.py中进行新类的导入。

```
from .mindie_tokenizer import MindIETokenizer
```


f. 在vllm_npu/__init__.py文件中添加新模块的导入。

```
import vllm_npu.transformers_utils
import vllm_npu.entrypoints
import vllm.sequence as v_sequence

v_sequence.MultiModalData = MultiModalData
```

最后，总结对接MindIE LLM的关键部件修改的代码量，如表7-20所示。

表 7-20 对接 MindIE LLM 的关键部件修改的代码量

重要组件	代码量	作用
MindIELlmWrapper	~120行	对接MindIE-LLM模型调用接口。
AscendSampler.py	~300行	对接MindIE-LLM后处理接口。
AscendModelRunner	~170行	原生框架中关键组件ModelRunner中的关键函数execute_model，load_model和profile_run的适配，在这里将模型导入部分替换为导入MindIELlmWrapper。

7.2.2.3 环境安装与启动服务

前提条件

- 已参见《MindIE安装指南》中“**安装驱动和固件**”章节完成驱动和固件的安装。
- 已参见《MindIE安装指南》中“**安装开发环境**”章节完成CANN、Python 3.10.2、PyTorch 2.1.0框架和Torch_NPU 2.1.0插件的安装。
- 已参见《MindIE安装指南》中“**物理机部署MindIE**”章节完成MindIE的安装。

安装步骤

步骤1 安装vllm_npu与vllm。

- 按照**7.2.2 适配说明**与**7.2.3 样例代码**章节的指导，将代码编写为文件并按照对应目录结构调整，以vLLM 0.3.3版本为例，将代码文件整理成以下目录结构。

```
.
├── cover
│   ├── vllm
│   │   └── __init__.py
│   ├── requirements.txt
│   └── setup.py
├── examples
│   ├── start_server.sh
│   ├── test_offline.py
│   └── test_offline.sh
├── install.sh
└── vllm_npu
    ├── requirements.txt
    ├── setup.py
    ├── tests
```



```
    ├── models
    │   ├── __init__.py
    │   └── test_models.py
    └── sampler
        └── test_sampler.py
└── vllm_npu
    ├── config.py
    ├── core
    │   ├── __init__.py
    │   └── scheduler.py
    ├── engine
    │   ├── __init__.py
    │   ├── llm_engine.py
    │   └── ray_utils.py
    ├── __init__.py
    ├── model_executor
    │   ├── ascend_model_loader.py
    │   ├── __init__.py
    │   ├── layers
    │   │   ├── __init__.py
    │   │   └── sampler.py
    │   ├── models
    │   │   ├── ascend
    │   │   │   ├── __init__.py
    │   │   │   └── mindie_llm_wrapper.py
    │   │   └── __init__.py
    │   └── utils.py
    ├── npu_adaptor.py
    ├── utils.py
    └── worker
        ├── ascend_worker.py
        ├── cache_engine.py
        ├── __init__.py
        └── model_runner.py
```

- 其他版本的项目同样按照对应指南进行创建。创建完成后，进入该目录中，执行安装脚本。

须知

请保持网络状态畅通，避免因网络问题而安装失败。

```
bash install.sh
```

- 脚本执行完毕后，可通过以下命令检查是否安装成功。

```
pip show vllm
pip show vllm_npu
```

- 若出现以下类似显示，则说明安装成功。

```
Name: vllm
Version: 0.3.3
Summary: A high-throughput and memory-efficient inference and serving engine for LLMs
Home-page: https://github.com/vllm-project/vllm
Author: vLLM Team
Author-email:
License: Apache 2.0
Requires: fastapi, ninja, numpy, outlines, prometheus_client, psutil, pydantic, pynvml, ray,
sentencepiece, transformers, uvicorn
Required-by:
Name: vllm-npu
Version: 0.3.3
Summary: A high-throughput and memory-efficient inference and serving engine for LLMs
Home-page: UNKNOWN
Author: Huawei
Author-email:
License: Apache 2.0
Requires: absl-py, accelerate, attrs, cloudpickle, decorator, numpy, pandas, psutil, ray, scipy, tornado,
```



```
transformers
Required-by:
```

步骤2 拉起vLLM服务。

在**步骤1**创建的文件夹下的examples路径中存在离线推理和在线推理的示例demo脚本，分别为test_offline.sh和start_server.sh，使用方法如下：

- 离线推理：首先修改test_offline.sh中的model_path参数指定模型路径，然后修改test_offline.py中的tensor_parallel_size参数指定使用的卡数，以及test_offline.py中的prompts变量添加需要推理的prompt内容，然后运行脚本。
  ```
  bash test_offline.sh
  ```
- 在线推理：首先修改start_server.sh中的model参数指定模型路径，并添加-tp参数，后面跟数字指定使用的卡数，然后运行脚本，即可拉起vLLM服务。
  ```
  bash start_server.sh
  ```

步骤3 客户端使用curl、requests等方式向服务端发送推理请求。

须知

- 如果需要使用https协议发送请求，需要在启动vLLM时配置对应的证书，以支持https协议访问（请参考**vLLM文档**，设置以下参数：--ssl-keyfile，--ssl-certfile，--ssl-ca-certs，--ssl-cert-reqs）。
- 这里的model_path需要与拉起服务指定的model参数保持一致。

```
curl https://localhost:8004/v1/completions \
 -H "Content-Type: application/json" \
 -d '{
   "model": "model_path",
   "max_tokens": 1,
   "temperature": 0,
   "top_p": 0.9,
   "prompt": "The future of AI is"
 }'
```

----结束

7.2.3 样例代码

7.2.3.1 vLLM 0.3.3 版本参考适配代码

1. ./cover

   a. ./cover/vllm/__init__.py

```
"""vLLM: a high-throughput and memory-efficient inference engine for LLMs"""
import vllm_npu
from vllm.engine.arg_utils import AsyncEngineArgs, EngineArgs
from vllm.engine.async_llm_engine import AsyncLLMEngine
from vllm.engine.llm_engine import LLMEngine
from vllm.engine.ray_utils import initialize_cluster
from vllm.entrypoints.llm import LLM
from vllm.outputs import CompletionOutput, RequestOutput
from vllm.sampling_params import SamplingParams
__version__ = "0.3.3"
__all__ = [
    "LLM",
    "SamplingParams",
    "RequestOutput",
    "CompletionOutput",
```

```
    "LLMEngine",
    "EngineArgs",
    "AsyncLLMEngine",
    "AsyncEngineArgs",
    "initialize_cluster",
]
```

b. ./cover/requirements.txt

```
ninja  # For faster builds.
psutil
ray == 2.9.3
sentencepiece  # Required for LLaMA tokenizer.
numpy
transformers >= 4.38.0  # Required for Gemma.
fastapi
uvicorn[standard]
pydantic >= 2.0  # Required for OpenAI server.
prometheus_client >= 0.18.0
pynvml == 11.5.0
outlines >= 0.0.27
```

c. ./cover/setup.py

```
import contextlib
import io
import os
import re
import subprocess
import warnings
from pathlib import Path
from typing import List, Set
from packaging.version import parse, Version
import setuptools
import torch
import torch.utils.cpp_extension as torch_cpp_ext
from torch.utils.cpp_extension import BuildExtension, CUDAExtension, CUDA_HOME,
ROCM_HOME
ROOT_DIR = os.path.dirname(__file__)
# If you are developing the C++ backend of vLLM, consider building vLLM with
# `python setup.py develop` since it will give you incremental builds.
# The downside is that this method is deprecated, see
# https://github.com/pypa/setuptools/issues/917
MAIN_CUDA_VERSION = "12.1"
# Supported NVIDIA GPU architectures.
NVIDIA_SUPPORTED_ARCHS = {"7.0", "7.5", "8.0", "8.6", "8.9", "9.0"}
ROCM_SUPPORTED_ARCHS = {"gfx908", "gfx90a", "gfx942", "gfx1100"}
# SUPPORTED_ARCHS = NVIDIA_SUPPORTED_ARCHS.union(ROCM_SUPPORTED_ARCHS)
def _is_hip() -> bool:
    return torch.version.hip is not None
def _is_neuron() -> bool:
    torch_neuronx_installed = True
    try:
        subprocess.run(["neuron-ls"], capture_output=True, check=True)
    except (FileNotFoundError, PermissionError):
        torch_neuronx_installed = False
    return torch_neuronx_installed
def _is_cuda() -> bool:
    return (torch.version.cuda is not None) and not _is_neuron()
# Compiler flags.
CXX_FLAGS = ["-g", "-O2", "-std=c++17"]
# TODO(woosuk): Should we use -O3?
NVCC_FLAGS = ["-O2", "-std=c++17"]
if _is_hip():
    if ROCM_HOME is None:
        raise RuntimeError(
            "Cannot find ROCM_HOME. ROCm must be available to build the package."
        )
    NVCC_FLAGS += ["-DUSE_ROCM"]
    NVCC_FLAGS += ["-U__HIP_NO_HALF_CONVERSIONS__"]
    NVCC_FLAGS += ["-U__HIP_NO_HALF_OPERATORS__"]
if _is_cuda() and CUDA_HOME is None:
```



```
    raise RuntimeError(
        "Cannot find CUDA_HOME. CUDA must be available to build the package.")
ABI = 1 if torch._C._GLIBCXX_USE_CXX11_ABI else 0
CXX_FLAGS += [f"-D_GLIBCXX_USE_CXX11_ABI={ABI}"]
NVCC_FLAGS += [f"-D_GLIBCXX_USE_CXX11_ABI={ABI}"]
def get_hipcc_rocm_version():
    # Run the hipcc --version command
    result = subprocess.run(['hipcc', '--version'],
                            stdout=subprocess.PIPE,
                            stderr=subprocess.STDOUT,
                            text=True)
    # Check if the command was executed successfully
    if result.returncode != 0:
        print("Error running 'hipcc --version'")
        return None
    # Extract the version using a regular expression
    match = re.search(r'HIP version: (\S+)', result.stdout)
    if match:
        # Return the version string
        return match.group(1)
    else:
        print("Could not find HIP version in the output")
        return None
def glob(pattern: str):
    root = Path(__name__).parent
    return [str(p) for p in root.glob(pattern)]
def get_neuronxcc_version():
    import sysconfig
    site_dir = sysconfig.get_paths()["purelib"]
    version_file = os.path.join(site_dir, "neuronxcc", "version",
                                "__init__.py")
    # Check if the command was executed successfully
    with open(version_file, "rt") as fp:
        content = fp.read()
    # Extract the version using a regular expression
    match = re.search(r"__version__ = '(\S+)'", content)
    if match:
        # Return the version string
        return match.group(1)
    else:
        raise RuntimeError("Could not find HIP version in the output")
def get_nvcc_cuda_version(cuda_dir: str) -> Version:
    """Get the CUDA version from nvcc.
    Adapted from https://github.com/NVIDIA/apex/blob/
8b7a1ff183741dd8f9b87e7bafd04cfde99cea28/setup.py
    """
    nvcc_output = subprocess.check_output([cuda_dir + "/bin/nvcc", "-V"],
                                          universal_newlines=True)
    output = nvcc_output.split()
    release_idx = output.index("release") + 1
    nvcc_cuda_version = parse(output[release_idx].split(",")[0])
    return nvcc_cuda_version
def get_pytorch_rocm_arch() -> Set[str]:
    """Get the cross section of Pytorch,and vllm supported gfx arches
    ROCM can get the supported gfx architectures in one of two ways
    Either through the PYTORCH_ROCM_ARCH env var, or output from
    rocm_agent_enumerator.
    In either case we can generate a list of supported arch's and
    cross reference with VLLM's own ROCM_SUPPORTED_ARCHs.
    """
    env_arch_list = os.environ.get("PYTORCH_ROCM_ARCH", None)
    # If we don't have PYTORCH_ROCM_ARCH specified pull the list from
rocm_agent_enumerator
    if env_arch_list is None:
        command = "rocm_agent_enumerator"
        env_arch_list = subprocess.check_output([command]).decode('utf-8')\
                        .strip().replace("\n", ";")
        arch_source_str = "rocm_agent_enumerator"
    else:
```



```
        arch_source_str = "PYTORCH_ROCM_ARCH env variable"
    # List are separated by ; or space.
    pytorch_rocm_arch = set(env_arch_list.replace(" ", ";").split(";"))
    # Filter out the invalid architectures and print a warning.
    arch_list = pytorch_rocm_arch.intersection(ROCM_SUPPORTED_ARCHS)
    # If none of the specified architectures are valid, raise an error.
    if not arch_list:
        raise RuntimeError(
            f"None of the ROCM architectures in {arch_source_str} "
            f"({env_arch_list}) is supported. "
            f"Supported ROCM architectures are: {ROCM_SUPPORTED_ARCHS}.")
    invalid_arch_list = pytorch_rocm_arch - ROCM_SUPPORTED_ARCHS
    if invalid_arch_list:
        warnings.warn(
            f"Unsupported ROCM architectures ({invalid_arch_list}) are "
            f"excluded from the {arch_source_str} output "
            f"({env_arch_list}). Supported ROCM architectures are: "
            f"{ROCM_SUPPORTED_ARCHS}.",
            stacklevel=2)
    return arch_list
def get_torch_arch_list() -> Set[str]:
    # TORCH_CUDA_ARCH_LIST can have one or more architectures,
    # e.g. "8.0" or "7.5,8.0,8.6+PTX". Here, the "8.6+PTX" option asks the
    # compiler to additionally include PTX code that can be runtime-compiled
    # and executed on the 8.6 or newer architectures. While the PTX code will
    # not give the best performance on the newer architectures, it provides
    # forward compatibility.
    env_arch_list = os.environ.get("TORCH_CUDA_ARCH_LIST", None)
    if env_arch_list is None:
        return set()
    # List are separated by ; or space.
    torch_arch_list = set(env_arch_list.replace(" ", ";").split(";"))
    if not torch_arch_list:
        return set()
    # Filter out the invalid architectures and print a warning.
    valid_archs = NVIDIA_SUPPORTED_ARCHS.union(
        {s + "+PTX"
         for s in NVIDIA_SUPPORTED_ARCHS})
    arch_list = torch_arch_list.intersection(valid_archs)
    # If none of the specified architectures are valid, raise an error.
    if not arch_list:
        raise RuntimeError(
            "None of the CUDA architectures in `TORCH_CUDA_ARCH_LIST` env "
            f"variable ({env_arch_list}) is supported. "
            f"Supported CUDA architectures are: {valid_archs}.")
    invalid_arch_list = torch_arch_list - valid_archs
    if invalid_arch_list:
        warnings.warn(
            f"Unsupported CUDA architectures ({invalid_arch_list}) are "
            "excluded from the `TORCH_CUDA_ARCH_LIST` env variable "
            f"({env_arch_list}). Supported CUDA architectures are: "
            f"{valid_archs}.",
            stacklevel=2)
    return arch_list
if _is_hip():
    rocm_arches = get_pytorch_rocm_arch()
    NVCC_FLAGS += ["--offload-arch=" + arch for arch in rocm_arches]
else:
    # First, check the TORCH_CUDA_ARCH_LIST environment variable.
    compute_capabilities = get_torch_arch_list()
if _is_cuda() and not compute_capabilities:
    # If TORCH_CUDA_ARCH_LIST is not defined or empty, target all available
    # GPUs on the current machine.
    device_count = torch.cuda.device_count()
    for i in range(device_count):
        major, minor = torch.cuda.get_device_capability(i)
        if major < 7:
            raise RuntimeError(
                "GPUs with compute capability below 7.0 are not supported.")
```



```
        compute_capabilities.add(f"{major}.{minor}")
ext_modules = []
if _is_cuda():
    nvcc_cuda_version = get_nvcc_cuda_version(CUDA_HOME)
    if not compute_capabilities:
        # If no GPU is specified nor available, add all supported architectures
        # based on the NVCC CUDA version.
        compute_capabilities = NVIDIA_SUPPORTED_ARCHS.copy()
        if nvcc_cuda_version < Version("11.1"):
            compute_capabilities.remove("8.6")
        if nvcc_cuda_version < Version("11.8"):
            compute_capabilities.remove("8.9")
            compute_capabilities.remove("9.0")
    # Validate the NVCC CUDA version.
    if nvcc_cuda_version < Version("11.0"):
        raise RuntimeError(
            "CUDA 11.0 or higher is required to build the package.")
    if (nvcc_cuda_version < Version("11.1")
            and any(cc.startswith("8.6") for cc in compute_capabilities)):
        raise RuntimeError(
            "CUDA 11.1 or higher is required for compute capability 8.6.")
    if nvcc_cuda_version < Version("11.8"):
        if any(cc.startswith("8.9") for cc in compute_capabilities):
            # CUDA 11.8 is required to generate the code targeting compute capability 8.9.
            # However, GPUs with compute capability 8.9 can also run the code generated by
            # the previous versions of CUDA 11 and targeting compute capability 8.0.
            # Therefore, if CUDA 11.8 is not available, we target compute capability 8.0
            # instead of 8.9.
            warnings.warn(
                "CUDA 11.8 or higher is required for compute capability 8.9. "
                "Targeting compute capability 8.0 instead.",
                stacklevel=2)
            compute_capabilities = set(cc for cc in compute_capabilities
                                       if not cc.startswith("8.9"))
            compute_capabilities.add("8.0+PTX")
        if any(cc.startswith("9.0") for cc in compute_capabilities):
            raise RuntimeError(
                "CUDA 11.8 or higher is required for compute capability 9.0.")
    NVCC_FLAGS_PUNICA = NVCC_FLAGS.copy()
    # Add target compute capabilities to NVCC flags.
    for capability in compute_capabilities:
        num = capability[0] + capability[2]
        NVCC_FLAGS += ["-gencode", f"arch=compute_{num},code=sm_{num}"]
        if capability.endswith("+PTX"):
            NVCC_FLAGS += [
                "-gencode", f"arch=compute_{num},code=compute_{num}"
            ]
        if int(capability[0]) >= 8:
            NVCC_FLAGS_PUNICA += [
                "-gencode", f"arch=compute_{num},code=sm_{num}"
            ]
            if capability.endswith("+PTX"):
                NVCC_FLAGS_PUNICA += [
                    "-gencode", f"arch=compute_{num},code=compute_{num}"
                ]
    # Use NVCC threads to parallelize the build.
    if nvcc_cuda_version >= Version("11.2"):
        nvcc_threads = int(os.getenv("NVCC_THREADS", 8))
        num_threads = min(os.cpu_count(), nvcc_threads)
        NVCC_FLAGS += ["--threads", str(num_threads)]
    if nvcc_cuda_version >= Version("11.8"):
        NVCC_FLAGS += ["-DENABLE_FP8_E5M2"]
    # changes for punica kernels
    NVCC_FLAGS += torch_cpp_ext.COMMON_NVCC_FLAGS
    REMOVE_NVCC_FLAGS = [
        '-D__CUDA_NO_HALF_OPERATORS__',
        '-D__CUDA_NO_HALF_CONVERSIONS__',
        '-D__CUDA_NO_BFLOAT16_CONVERSIONS__',
        '-D__CUDA_NO_HALF2_OPERATORS__',
```



```
    ]
    for flag in REMOVE_NVCC_FLAGS:
        with contextlib.suppress(ValueError):
            torch_cpp_ext.COMMON_NVCC_FLAGS.remove(flag)
    install_punica = bool(int(os.getenv("VLLM_INSTALL_PUNICA_KERNELS", "0")))
    device_count = torch.cuda.device_count()
    for i in range(device_count):
        major, minor = torch.cuda.get_device_capability(i)
        if major < 8:
            install_punica = False
            break
    if install_punica:
        ext_modules.append(
            CUDAExtension(
                name="vllm._punica_C",
                sources=["csrc/punica/punica_ops.cc"] +
                glob("csrc/punica/bgmv/*.cu"),
                extra_compile_args={
                    "cxx": CXX_FLAGS,
                    "nvcc": NVCC_FLAGS_PUNICA,
                },
            ))
elif _is_neuron():
    neuronxcc_version = get_neuronxcc_version()
vllm_extension_sources = [
]
if _is_cuda():
    vllm_extension_sources.append("csrc/quantization/awq/gemm_kernels.cu")
    vllm_extension_sources.append(
        "csrc/quantization/marlin/marlin_cuda_kernel.cu")
    vllm_extension_sources.append("csrc/custom_all_reduce.cu")
    # Add MoE kernels.
    ext_modules.append(
        CUDAExtension(
            name="vllm._moe_C",
            sources=glob("csrc/moe/*.cu") + glob("csrc/moe/*.cpp"),
            extra_compile_args={
                "cxx": CXX_FLAGS,
                "nvcc": NVCC_FLAGS,
            },
        ))
"""
if not _is_neuron():
    vllm_extension = CUDAExtension(
        name="vllm._C",
        sources=vllm_extension_sources,
        extra_compile_args={
            "cxx": CXX_FLAGS,
            "nvcc": NVCC_FLAGS,
        },
        libraries=["cuda"] if _is_cuda() else [],
    )
    ext_modules.append(vllm_extension)
"""
def get_path(*filepath) -> str:
    return os.path.join(ROOT_DIR, *filepath)
def find_version(filepath: str) -> str:
    """Extract version information from the given filepath.
    Adapted from https://github.com/ray-project/ray/blob/
0b190ee1160eeca9796bc091e07eaebf4c85b511/python/setup.py
    """
    with open(filepath) as fp:
        version_match = re.search(r"^__version__ = ['\"]([^'\"]*)['\"]",
                                  fp.read(), re.M)
        if version_match:
            return version_match.group(1)
        raise RuntimeError("Unable to find version string.")
def get_vllm_version() -> str:
    version = find_version(get_path("vllm", "__init__.py"))
```



```
    return version
    if _is_hip():
        # Get the HIP version
        hipcc_version = get_hipcc_rocm_version()
        if hipcc_version != MAIN_CUDA_VERSION:
            rocm_version_str = hipcc_version.replace(".", "")[:3]
            version += f"+rocm{rocm_version_str}"
    elif _is_neuron():
        # Get the Neuron version
        neuron_version = str(neuronxcc_version)
        if neuron_version != MAIN_CUDA_VERSION:
            neuron_version_str = neuron_version.replace(".", "")[:3]
            version += f"+neuron{neuron_version_str}"
    else:
        cuda_version = str(nvcc_cuda_version)
        if cuda_version != MAIN_CUDA_VERSION:
            cuda_version_str = cuda_version.replace(".", "")[:3]
            version += f"+cu{cuda_version_str}"
    return version
def read_readme() -> str:
    """Read the README file if present."""
    p = get_path("README.md")
    if os.path.isfile(p):
        return io.open(get_path("README.md"), "r", encoding="utf-8").read()
    else:
        return ""
def get_requirements() -> List[str]:
    """Get Python package dependencies from requirements.txt."""
    if _is_hip():
        with open(get_path("requirements-rocm.txt")) as f:
            requirements = f.read().strip().split("\n")
    elif _is_neuron():
        with open(get_path("requirements-neuron.txt")) as f:
            requirements = f.read().strip().split("\n")
    else:
        with open(get_path("requirements.txt")) as f:
            requirements = f.read().strip().split("\n")
    return requirements
package_data = {
    "vllm": ["py.typed", "model_executor/layers/fused_moe/configs/*.json"]
}
if os.environ.get("VLLM_USE_PRECOMPILED"):
    ext_modules = []
    package_data["vllm"].append("*.so")
setuptools.setup(
    name="vllm",
    version=get_vllm_version(),
    author="vLLM Team",
    license="Apache 2.0",
    description=("A high-throughput and memory-efficient inference and "
                 "serving engine for LLMs"),
    long_description=read_readme(),
    long_description_content_type="text/markdown",
    url="https://github.com/vllm-project/vllm",
    project_urls={
        "Homepage": "https://github.com/vllm-project/vllm",
        "Documentation": "https://vllm.readthedocs.io/en/latest/",
    },
    classifiers=[
        "Programming Language :: Python :: 3.8",
        "Programming Language :: Python :: 3.9",
        "Programming Language :: Python :: 3.10",
        "Programming Language :: Python :: 3.11",
        "License :: OSI Approved :: Apache Software License",
        "Topic :: Scientific/Engineering :: Artificial Intelligence",
    ],
    packages=setuptools.find_packages(exclude=("benchmarks", "csrc", "docs",
                                               "examples", "tests")),
    python_requires=">=3.8",
```



```
    install_requires=get_requirements(),
    ext_modules=ext_modules,
    cmdclass={"build_ext": BuildExtension} if not _is_neuron() else {},
    package_data=package_data,
)
```

2. ./examples

   a. ./examples/start_server.sh

```
#!/bin/bash
# ATB加速库环境变量
export ATB_LAYER_INTERNAL_TENSOR_REUSE=1
export ATB_WORKSPACE_MEM_ALLOC_GLOBAL=1
export ATB_OPERATION_EXECUTE_ASYNC=1
export TASK_QUEUE_ENABLE=1
export ATB_CONTENT_NCHW_TO_ND=1
export ATB_CONTEXT_WORKSPACE_RING=1
export HCCL_BUFFSIZE=120
export LCCL_DETERMINISTIC=1
export HCCL_DETERMINISTIC=1
export ATB_OPSRUNNER_KERNEL_CACHE_LOCAL_COUNT=8
export ATB_OPSRUNNER_KERNEL_CACHE_GLOABL_COUNT=16
export ATB_LAUNCH_KERNEL_WITH_TILING=0
export VLLM_NO_USAGE_STATS=1 # close vllm usage messages to avoid errors
python -m vllm.entrypoints.openai.api_server --model=facebook/opt-125m --trust-remote-code
--enforce-eager --worker-use-ray
```

   b. ./examples/test_offline.py

```
from vllm import LLM, SamplingParams
import json
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('--model_path', type=str, default="facebook/opt-125m")
# input prompts for test
prompts = [
    "Hello, my name is",
    "The president of the United States is",
    "The capital of France is",
    "The future of AI is",
]
sampling_params = SamplingParams(max_tokens=512, temperature=0)
args = parser.parse_args()
model_path = args.model_path
llm = LLM(model=model_path,
          max_model_len=4096, # max length of prompt
          tensor_parallel_size=8, # number of NPUs to be used
          max_num_seqs=256, # max batch number
          enforce_eager=True, # disable CUDA graph mode
          trust_remote_code=True, # If the model is a custom model not yet available in the
HuggingFace transformers library
)
outputs = llm.generate(prompts, sampling_params)
for i, output in enumerate(outputs):
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"req_num: {i}\nPrompt: {prompt!r}\nGenerated text: {generated_text!r}")
```

   c. ./examples/test_offline.sh

```
#!/bin/bash
# ATB加速库环境变量
export ATB_LAYER_INTERNAL_TENSOR_REUSE=1
export ATB_WORKSPACE_MEM_ALLOC_GLOBAL=1
export ATB_OPERATION_EXECUTE_ASYNC=1
export TASK_QUEUE_ENABLE=1
export ATB_CONTENT_NCHW_TO_ND=1
export ATB_CONTEXT_WORKSPACE_RING=1
export HCCL_BUFFSIZE=120
export LCCL_DETERMINISTIC=1
export HCCL_DETERMINISTIC=true
export ATB_OPSRUNNER_KERNEL_CACHE_LOCAL_COUNT=8
export ATB_OPSRUNNER_KERNEL_CACHE_GLOABL_COUNT=16
```



```
export ATB_LAUNCH_KERNEL_WITH_TILING=0
export VLLM_NO_USAGE_STATS=1 # close vllm usage messages to avoid errors
python3 test_offline.py --model_path facebook/opt-125m
```

3. ./vllm_npu/tests

   a. ./vllm_npu/tests/models/test_models.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
from typing import Union
import pytest
import torch
from mindie_llm.modeling.backend_type import BackendType
from pytest_mock import MockerFixture
from vllm.config import ModelConfig, ParallelConfig
from vllm.model_executor.input_metadata import InputMetadata
from vllm_npu import DeviceConfig
from vllm_npu.model_executor.models.ascend.mindie_llm_wrapper import MindIELlmWrapper
from vllm_npu.worker.ascend_worker import Worker
MODELS = ["Ascend/Mock-Model-42B"]
PARAMS = [
    {"rank": 0, "local_rank": 0, "world_size": 1},
    {"rank": 1, "local_rank": 1, "world_size": 2},
]
@pytest.mark.parametrize("model", MODELS)
@pytest.mark.parametrize("params", PARAMS)
@pytest.mark.parametrize("dtype", [torch.bfloat16])
def test_load_model(
    mocker: MockerFixture, model: str, params: list[dict[str, int]], dtype: Union[str, torch.dtype]
) -> None:
    world_size, local_rank, rank = params.get("world_size"), params.get("local_rank"),
params.get("rank")
    mock_parallel_config = mocker.MagicMock(spec=ParallelConfig)
    mock_parallel_config.world_size = world_size
    mock_model_config = mocker.MagicMock(spec=ModelConfig)
    mock_model_config.model = model
    mock_model_config.dtype = dtype
    mock_model_config.load_format = "auto"
    mock_model_config.max_context_len_to_capture = 0
    worker = Worker(mock_model_config, mock_parallel_config, None, DeviceConfig(),
local_rank, rank, None)
    mock_get_model =
mocker.patch("vllm_npu.model_executor.ascend_model_loader.MindIELlmWrapper")
    worker.load_model()
    # Check if the args are correctly passed to backend
    mock_get_model.assert_called_once_with(
        {
            "backend_type": BackendType.ATB,
            "model_id": model,
            "rank": rank,
            "local_rank": local_rank,
            "world_size": world_size,
            "npu_device_id": local_rank,
        }
    )
@pytest.mark.parametrize("load_format", ["auto", "safetensors", "pt"])
def test_load_weights(mocker: MockerFixture, load_format: str) -> None:
    mocker.patch("torch.get_default_dtype", return_value=torch.float32)
    mocker.patch("torch.set_default_dtype")
    mock_generator_torch =
mocker.patch("vllm_npu.model_executor.models.ascend.mindie_llm_wrapper.GeneratorTorch")
    mock_sampler =
mocker.patch("vllm_npu.model_executor.models.ascend.mindie_llm_wrapper.Sampler")
    mindie_llm_wrapper = MindIELlmWrapper(mocker.ANY)
    # Testing supported formats, raising inside
    mindie_llm_wrapper.load_weights(load_format)
    # Testing GeneratorTorch and Sampler call counts
    mock_generator_torch.assert_called_once()
    mock_sampler.assert_called_once()
    # Testing unsupported format
    try:
```



```
        mindie_llm_wrapper.load_weights("unsupported")
    except ValueError as e:
        if "load-format support [safetensors, pt]" not in str(e):
            raise AssertionError("Error message does not match expected text for unsupported
formats") from e
    else:
        raise AssertionError("Expected a ValueError for unsupported format, but none was raised")
@pytest.mark.parametrize("with_kv_cache", [True, False])
@pytest.mark.parametrize("is_prompt", [True, False])
def test_forward(mocker: MockerFixture, with_kv_cache: bool, is_prompt: bool) -> None:
    mocker.patch("torch.get_default_dtype", return_value=torch.float32)
    mocker.patch("torch.set_default_dtype")
    mocker.patch("vllm_npu.model_executor.models.ascend.mindie_llm_wrapper.GeneratorTorch")
    mocker.patch("vllm_npu.model_executor.models.ascend.mindie_llm_wrapper.Sampler")
    wrapper = MindIELlmWrapper(mocker.ANY)
    wrapper.load_weights()
    mock_forward_tensor = mocker.patch.object(
        wrapper.mindie_model, "forward_tensor", return_value=torch.tensor([[0.1, 0.2, 0.3]])
    )
    input_ids = torch.tensor([[1, 2, 3]])
    positions = torch.tensor([[0, 1, 2]])
    kv_caches = [(torch.tensor([0]), torch.tensor([0]))] if with_kv_cache else [(None, None)]
    lora_requests = [None]
    mock_input_metadata = mocker.MagicMock(spec=InputMetadata)
    mock_input_metadata.is_prompt = is_prompt
    mock_input_metadata.prompt_lens = torch.tensor([3], dtype=torch.int32)
    mock_input_metadata.context_lens = torch.tensor([3], dtype=torch.int32)
    mock_input_metadata.max_context_len = 3
    mock_input_metadata.block_tables = torch.tensor([[0]], dtype=torch.int32, device="npu")
    mock_input_metadata.slot_mapping = torch.tensor([0, 1, 2], dtype=torch.int64)
    wrapper.forward(input_ids, positions, kv_caches, mock_input_metadata, lora_requests)
    if is_prompt:
        expected_lengths = mock_input_metadata.prompt_lens.to(torch.int32)
        expected_max_seq_len = int(mock_input_metadata.prompt_lens.max())
        expected_lm_head_indices = (mock_input_metadata.prompt_lens.cumsum(dim=-1) -
1).to(torch.int64)
    else:
        expected_lengths = mock_input_metadata.context_lens
        expected_max_seq_len = mock_input_metadata.max_context_len
        expected_lm_head_indices = None
    if with_kv_cache:
        expected_kv_caches = kv_caches
        expected_block_tables = mock_input_metadata.block_tables
        expected_slots = mock_input_metadata.slot_mapping
    else:
        _, expected_block_tables, expected_slots =
wrapper.create_dummy_kv_cache(mock_input_metadata, input_ids)
        # Hard to compare this object
        expected_kv_caches = mocker.ANY
    expected_adapter_ids = [None]
    mock_forward_tensor.assert_called_once_with(
        input_ids,
        positions,
        is_prompt,
        expected_kv_caches,
        expected_block_tables,
        expected_slots,
        expected_lengths,
        expected_max_seq_len,
        expected_lm_head_indices,
        adapter_ids = expected_adapter_ids,
    )
```

b. ./vllm_npu/tests/sampler/test_sampler.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
import random
from typing import Union
import numpy as np
import pytest
import torch
```

```
from mindie_llm.text_generator.utils.sampling_metadata import (
    DoSampleMetadata,
    PenaltyMetadata,
    SamplingData,
    SamplingParam,
    SeedMetadata,
    TopKMetadata,
    TopPMetadata,
)
from pytest_mock import MockerFixture
from vllm.sampling_params import _SAMPLING_EPS
from vllm.sequence import SamplingParams, SequenceData, SequenceGroupMetadata
from vllm.worker.model_runner import ModelRunner
from vllm_npu.model_executor.layers.sampler import Sampler
torch.npu.set_device("npu")
def _prepare_test(
    batch_size: int, is_prompt: bool = True, random_seq_length: bool = False, temprature:
Union[int, float] = 0
) -> tuple[torch.Tensor, Sampler]:
    torch.npu.set_device("npu")
    sampler = Sampler(None)
    model_runner = ModelRunner(None, None, None, None, None)
    model_runner.set_block_size(128)
    model_runner.sliding_window = None
    seq_group_metadata_list = []
    prompt_lens = []
    for i in range(batch_size):
        token_ids = list(range(random.randint(1, 128))) if random_seq_length else [1, 2, 3]
        seq_group_metadata_list.append(
            SequenceGroupMetadata(
                request_id=f"test_{i}",
                is_prompt=is_prompt,
                seq_data={0: SequenceData(token_ids)},
                sampling_params=SamplingParams(temperature=temprature),
                block_tables={0: [1]},
            )
        )
        prompt_lens.append(seq_group_metadata_list[-1].seq_data[0].get_len())
    sampling_metadata = model_runner._prepare_sample(seq_group_metadata_list, prompt_lens,
prompt_lens)
    return sampling_metadata, sampler
@pytest.mark.parametrize("vocab_size", [32000])
@pytest.mark.parametrize("dtype", [torch.bfloat16])
def test_forward(mocker: MockerFixture, vocab_size: int, dtype: torch.dtype):
    batch_size = random.randint(1, 256)
    sampling_metadata, sampler = _prepare_test(batch_size)
    fake_logits = torch.full((batch_size, vocab_size), 1e-1, dtype=dtype)
    fake_tokens = np.arange(batch_size)
    fake_sample_results = [([i], [0]) for i in range(batch_size)]
    deconstructor = sampler.get_sample_data_and_param
    reconstructor = sampler.get_sample_results
    mock_process = mocker.patch(
        "vllm_npu.model_executor.layers.sampler._apply_logits_processors",
return_value=fake_logits
    )
    mock_atb_model = mocker.patch.object(sampler, "atb_model")
    mock_deconstructor = mocker.patch.object(
        sampler, "get_sample_data_and_param", side_effect=lambda *args, **kwargs:
deconstructor(*args, **kwargs)
    )
    mock_backend_sample = mocker.patch.object(mock_atb_model, "sample",
return_value=fake_tokens)
    mock_reconstructor = mocker.patch.object(
        sampler, "get_sample_results", side_effect=lambda *args, **kwargs: reconstructor(*args,
**kwargs)
    )
    mock_logprobs = mocker.patch(
        "vllm_npu.model_executor.layers.sampler._get_logprobs", return_value=(mocker.ANY,
mocker.ANY)
```



```
    )
    mocker.patch("vllm_npu.model_executor.layers.sampler._build_sampler_output")
    sampler.forward(fake_logits, sampling_metadata)
    mock_process.assert_called_once_with(fake_logits, sampling_metadata)
    mock_deconstructor.assert_called_once_with(sampling_metadata, vocab_size)
    mock_backend_sample.assert_called_once_with(fake_logits, mocker.ANY, mocker.ANY)
    mock_reconstructor.assert_called_once_with(sampling_metadata, fake_tokens)
    mock_logprobs.assert_called_once_with(mocker.ANY, sampling_metadata,
fake_sample_results)
@pytest.mark.parametrize("temperature", [0, 0.1, 1, 10])
@pytest.mark.parametrize("vocab_size", [32000])
def test_sample_data_and_param(temperature, vocab_size):
    batch_size = random.randint(1, 256)
    sampling_metadata, sampler = _prepare_test(batch_size, temprature=temperature)
    sampling_data, sampling_param = sampler.get_sample_data_and_param(sampling_metadata,
vocab_size)
    # Sampling data
    if not isinstance(sampling_data, SamplingData):
        raise TypeError("The returned object is not an instance of SamplingData.")
    if not hasattr(sampling_data, "all_input_ids"):
        raise AttributeError("SamplingData does not have the 'all_input_ids' attribute.")
    if not hasattr(sampling_data, "output_ids"):
        raise AttributeError("SamplingData does not have the 'output_ids' attribute.")
    if not isinstance(sampling_data.all_input_ids, torch.Tensor):
        raise TypeError("all_input_ids should be a tensor.")
    if not isinstance(sampling_data.output_ids, torch.Tensor):
        raise TypeError("output_ids should be a tensor.")
    if sampling_data.all_input_ids.dtype != torch.int32:
        raise ValueError("The dtype of all_input_ids should be int32.")
    if sampling_data.output_ids.dtype != torch.int32:
        raise ValueError("The dtype of output_ids should be int32.")
    # Sampling param
    if temperature < _SAMPLING_EPS:
        if sampling_param is not None:
            raise ValueError("Sampling data should be None in greedy mode.")
    else:
        if not isinstance(sampling_param, SamplingParam):
            raise TypeError("The returned object is not an instance of SamplingParam.")
        if not hasattr(sampling_param, "penalty_meta"):
            raise AttributeError("SamplingParam does not have the 'penalty_meta' attribute.")
        if not hasattr(sampling_param, "temperature"):
            raise AttributeError("SamplingParam does not have the 'temperature' attribute.")
        if not hasattr(sampling_param, "top_k_meta"):
            raise AttributeError("SamplingParam does not have the 'top_k_meta' attribute.")
        if not hasattr(sampling_param, "top_p_meta"):
            raise AttributeError("SamplingParam does not have the 'top_p_meta' attribute.")
        if not hasattr(sampling_param, "seed_meta"):
            raise AttributeError("SamplingParam does not have the 'seed_meta' attribute.")
        if not hasattr(sampling_param, "do_sample_meta"):
            raise AttributeError("SamplingParam does not have the 'do_sample_meta' attribute.")
        if not isinstance(sampling_param.penalty_meta, PenaltyMetadata):
            raise TypeError("penalty_meta should be an instance of PenaltyMetadata.")
        if temperature != 1:
            if not isinstance(sampling_param.temperature, torch.Tensor):
                raise TypeError("Temperature should be a tensor.")
        else:
            if sampling_param.temperature is not None:
                raise ValueError("Temperature tensors should be None.")
        if not isinstance(sampling_param.top_k_meta, TopKMetadata):
            raise TypeError("top_k_meta should be an instance of TopKMetadata.")
        if not isinstance(sampling_param.top_p_meta, TopPMetadata):
            raise TypeError("top_p_meta should be an instance of TopPMetadata.")
        if not isinstance(sampling_param.seed_meta, SeedMetadata):
            raise TypeError("seed_meta should be an instance of SeedMetadata.")
        if not isinstance(sampling_param.do_sample_meta, DoSampleMetadata):
            raise TypeError("do_sample_meta should be an instance of DoSampleMetadata.")
def test_sample_results_reconstruction():
    batch_size = random.randint(1, 256)
    sampling_metadata, sampler = _prepare_test(batch_size)
```



```
    samples = np.arange(batch_size)
    sample_results = sampler.get_sample_results(sampling_metadata, samples)
    expected_sample_results = [([i], [0]) for i in range(batch_size)]
    if sample_results != expected_sample_results:
        raise ValueError(f"Sample results mismatch: Expected {expected_sample_results}, got
{sample_results}.")
    selected_seq_groups = [([0], SamplingParams()), ([1, 2, 3], SamplingParams()), ([4, 5],
SamplingParams())]
    samples = np.array([101, 102, 103, 104, 105, 106])
    results = sampler.extract_next_tokens_and_parents(selected_seq_groups, samples)
    expected_results = [([101], [0]), ([102], [0, 1, 2]), ([105], [0, 1])]
    if results != expected_results:
        raise ValueError(f"Results mismatch: Expected {expected_results}, got {results}.")
@pytest.mark.parametrize("is_prompt", [True, False])
@pytest.mark.parametrize("vocab_size", [32000])
def test_token_padding(is_prompt, vocab_size):
    batch_size = random.randint(1, 256)
    sampling_metadata, sampler = _prepare_test(batch_size, is_prompt, True)
    sampling_data, _ = sampler.get_sample_data_and_param(sampling_metadata, vocab_size)
    # Note that if padding fails, then sampling_data cannot be constructed.
    # So we just check some basic shape info here.
    if sampling_data.all_input_ids.shape[0] != batch_size:
        raise ValueError(
            f"Expected all_input_ids to have shape[0] of {batch_size}, but got
{sampling_data.all_input_ids.shape[0]}."
        )
    if sampling_data.output_ids.shape[0] != batch_size:
        raise ValueError(
            f"Expected output_ids to have shape[0] of {batch_size}, but got
{sampling_data.output_ids.shape[0]}."
        )
```

4. ./vllm_npu/vllm_npu/core

   a. ./vllm_npu/vllm_npu/core/__init__.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
from vllm_npu.core.scheduler import _schedule
from vllm.core.scheduler import Scheduler
Scheduler._schedule = _schedule
```

   b. ./vllm_npu/vllm_npu/core/sheduler.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
# Part of codes in this file was copied from project [vLLM Team][vllm]
import time
from typing import Dict, List, Deque, Optional
from collections import deque
from vllm.core.block_manager import AllocStatus
from vllm.sequence import SequenceGroup, SequenceStatus
from vllm.core.scheduler import logger, SchedulerOutputs
def _schedule(self) -> SchedulerOutputs:
    # Blocks that need to be swapped or copied before model execution.
    blocks_to_swap_in: Dict[int, int] = {}
    blocks_to_swap_out: Dict[int, int] = {}
    blocks_to_copy: Dict[int, List[int]] = {}
    # Fix the current time.
    now = time.monotonic()
    # Join waiting sequences if possible.
    if not self.swapped:
        ignored_seq_groups: List[SequenceGroup] = []
        scheduled: List[SequenceGroup] = []
        # The total number of sequences on the fly, including the
        # requests in the generation phase.
        num_curr_seqs = sum(seq_group.get_max_num_running_seqs()
                            for seq_group in self.running)
        num_batched_tokens = 0
        curr_loras = set(
            seq_group.lora_int_id
            for seq_group in self.running) if self.lora_enabled else None
        # Optimization: We do not sort the waiting queue since the preempted
        # sequence groups are added to the front and the new sequence groups
        # are added to the back.
```

```
        leftover_waiting_sequences = deque()
        while self.waiting:
            seq_group = self.waiting[0]
            waiting_seqs = seq_group.get_seqs(
                status=SequenceStatus.WAITING)
            assert len(waiting_seqs) == 1, (
                "Waiting sequence group should have only one prompt "
                "sequence.")
            num_prompt_tokens = waiting_seqs[0].get_len()
            if num_prompt_tokens > self.prompt_limit:
                logger.warning(
                    f"Input prompt ({num_prompt_tokens} tokens) is too long"
                    f" and exceeds limit of {self.prompt_limit}")
                for seq in waiting_seqs:
                    seq.status = SequenceStatus.FINISHED_IGNORED
                ignored_seq_groups.append(seq_group)
                self.waiting.popleft()
                continue
            # If the sequence group cannot be allocated, stop.
            can_allocate = self.block_manager.can_allocate(seq_group)
            if can_allocate == AllocStatus.LATER:
                break
            elif can_allocate == AllocStatus.NEVER:
                logger.warning(
                    f"Input prompt ({num_prompt_tokens} tokens) is too long"
                    f" and exceeds the capacity of block_manager")
                for seq in waiting_seqs:
                    seq.status = SequenceStatus.FINISHED_IGNORED
                ignored_seq_groups.append(seq_group)
                self.waiting.popleft()
                continue
            lora_int_id = 0
            if self.lora_enabled:
                lora_int_id = seq_group.lora_int_id
                if lora_int_id > 0 and lora_int_id not in curr_loras and len(
                        curr_loras) >= self.lora_config.max_loras:
                    # We don't have a space for another LoRA, so
                    # we ignore this request for now.
                    leftover_waiting_sequences.appendleft(seq_group)
                    self.waiting.popleft()
                    continue
            # If the number of batched tokens exceeds the limit, stop.
            if (num_batched_tokens + num_prompt_tokens >
                    self.scheduler_config.max_num_batched_tokens):
                break
            # The total number of sequences in the RUNNING state should not
            # exceed the maximum number of sequences.
            num_new_seqs = seq_group.get_max_num_running_seqs()
            if (num_curr_seqs + num_new_seqs >
                    self.scheduler_config.max_num_seqs):
                break
            if lora_int_id > 0:
                curr_loras.add(lora_int_id)
            self.waiting.popleft()
            self._allocate(seq_group)
            self.running.append(seq_group)
            num_batched_tokens += num_prompt_tokens
            num_curr_seqs += num_new_seqs
            scheduled.append(seq_group)
        self.waiting.extendleft(leftover_waiting_sequences)
        if scheduled or ignored_seq_groups:
            scheduler_outputs = SchedulerOutputs(
                scheduled_seq_groups=scheduled,
                prompt_run=True,
                num_batched_tokens=num_batched_tokens,
                blocks_to_swap_in=blocks_to_swap_in,
                blocks_to_swap_out=blocks_to_swap_out,
                blocks_to_copy=blocks_to_copy,
                ignored_seq_groups=ignored_seq_groups,
```



```
        )
        return scheduler_outputs
# NOTE(woosuk): Preemption happens only when there is no available slot
# to keep all the sequence groups in the RUNNING state.
# In this case, the policy is responsible for deciding which sequence
# groups to preempt.
self.running = self.policy.sort_by_priority(now, self.running)
# Reserve new token slots for the running sequence groups.
running: Deque[SequenceGroup] = deque()
preempted: List[SequenceGroup] = []
while self.running:
    seq_group = self.running.popleft()
    while not self.block_manager.can_append_slot(seq_group):
        if self.running:
            # Preempt the lowest-priority sequence groups.
            victim_seq_group = self.running.pop()
            self._preempt(victim_seq_group, blocks_to_swap_out)
            preempted.append(victim_seq_group)
        else:
            # No other sequence groups can be preempted.
            # Preempt the current sequence group.
            self._preempt(seq_group, blocks_to_swap_out)
            preempted.append(seq_group)
            break
    else:
        # Append new slots to the sequence group.
        self._append_slot(seq_group, blocks_to_copy)
        running.append(seq_group)
self.running = running
# Swap in the sequence groups in the SWAPPED state if possible.
self.swapped = self.policy.sort_by_priority(now, self.swapped)
if not preempted:
    num_curr_seqs = sum(seq_group.get_max_num_running_seqs()
                        for seq_group in self.running)
    curr_loras = set(
        seq_group.lora_int_id
        for seq_group in self.running) if self.lora_enabled else None
    leftover_swapped = deque()
    while self.swapped:
        seq_group = self.swapped[0]
        lora_int_id = 0
        if self.lora_enabled:
            lora_int_id = seq_group.lora_int_id
            if lora_int_id > 0 and lora_int_id not in curr_loras and len(
                    curr_loras) >= self.lora_config.max_loras:
                # We don't have a space for another LoRA, so
                # we ignore this request for now.
                leftover_swapped.appendleft(seq_group)
                self.swapped.popleft()
                continue
        # If the sequence group cannot be swapped in, stop.
        if not self.block_manager.can_swap_in(seq_group):
            break
        # The total number of sequences in the RUNNING state should not
        # exceed the maximum number of sequences.
        num_new_seqs = seq_group.get_max_num_running_seqs()
        if (num_curr_seqs + num_new_seqs >
                self.scheduler_config.max_num_seqs):
            break
        if lora_int_id > 0:
            curr_loras.add(lora_int_id)
        self.swapped.popleft()
        self._swap_in(seq_group, blocks_to_swap_in)
        self._append_slot(seq_group, blocks_to_copy)
        num_curr_seqs += num_new_seqs
        self.running.append(seq_group)
    self.swapped.extendleft(leftover_swapped)
# Each sequence in the generation phase only takes one token slot.
# Therefore, the number of batched tokens is equal to the number of
```



```
    # sequences in the RUNNING state.
    num_batched_tokens = sum(
        seq_group.num_seqs(status=SequenceStatus.RUNNING)
        for seq_group in self.running)
    scheduler_outputs = SchedulerOutputs(
        scheduled_seq_groups=self.running,
        prompt_run=False,
        num_batched_tokens=num_batched_tokens,
        blocks_to_swap_in=blocks_to_swap_in,
        blocks_to_swap_out=blocks_to_swap_out,
        blocks_to_copy=blocks_to_copy,
        ignored_seq_groups=[],
    )
    return scheduler_outputs
```

5. ./vllm_npu/vllm_npu/engine

   a. ./vllm_npu/vllm_npu/engine/__init__.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
from vllm_npu.config import DeviceConfig
from vllm_npu.utils import get_ip
from vllm_npu.engine.llm_engine import DEVICE_TO_WORKER_MODULE_MAP
from vllm_npu.engine.ray_utils import initialize_cluster
from vllm.engine import arg_utils, llm_engine, ray_utils
arg_utils.DeviceConfig = DeviceConfig
llm_engine.DEVICE_TO_WORKER_MODULE_MAP = DEVICE_TO_WORKER_MODULE_MAP
llm_engine.get_ip = get_ip
llm_engine.initialize_cluster = initialize_cluster
ray_utils.get_ip = get_ip
ray_utils.initialize_cluster = initialize_cluster
```

   b. ./vllm_npu/vllm_npu/engine/llm_engine.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
DEVICE_TO_WORKER_MODULE_MAP = {
    "cuda": "vllm.worker.worker",
    "neuron": "vllm.worker.neuron_worker",
    "npu": "vllm_npu.worker.ascend_worker",
}
```

   c. ./vllm_npu/vllm_npu/engine/ray_utils.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
# Part of codes in this file was copied from project [vLLM Team][vllm]
from typing import Optional, Tuple, TYPE_CHECKING
from vllm.config import ParallelConfig
from vllm.utils import is_hip
from vllm.logger import init_logger
logger = init_logger(__name__)
try:
    import ray
except ImportError as e:
    logger.warning(f"Failed to import Ray with {e!r}. "
                   "For distributed inference, please install Ray with "
                   "`pip install ray`.")
    ray = None
if TYPE_CHECKING:
    from ray.util.placement_group import PlacementGroup
def initialize_cluster(
    parallel_config: ParallelConfig,
    engine_use_ray: bool = False,
    ray_address: Optional[str] = None,
) -> Optional["PlacementGroup"]:
    """Initialize the distributed cluster probably with Ray.
    Args:
        parallel_config: The configurations for parallel execution.
        engine_use_ray: Whether to use Ray for async engine.
        ray_address: The address of the Ray cluster. If None, uses
            the default Ray cluster address.
    Returns:
        An optional `PlacementGroup`. It includes the specification
        of the resources for each distributed worker. None if Ray is
        not used.
```



```
    """
    if parallel_config.worker_use_ray or engine_use_ray:
        if ray is None:
            raise ImportError(
                "Ray is not installed. Please install Ray to use distributed "
                "serving.")
        # Connect to a ray cluster.
        if is_hip():
            ray.init(address=ray_address,
                     ignore_reinit_error=True,
                     num_gpus=parallel_config.world_size)
        else:
            """start adapt"""
            ray.init(address=ray_address, ignore_reinit_error=True,
                       num_gpus=parallel_config.world_size)
            """end adapt"""
    if not parallel_config.worker_use_ray:
        assert parallel_config.world_size == 1, (
            "Ray is required if parallel_config.world_size > 1.")
        return None
    # Create placement group for worker processes
    current_placement_group = ray.util.get_current_placement_group()
    if current_placement_group:
        # We are in a placement group
        bundles = current_placement_group.bundle_specs
        # Verify that we can use the placement group.
        gpu_bundles = 0
        for bundle in bundles:
            bundle_gpus = bundle.get("GPU", 0)
            if bundle_gpus > 1:
                raise ValueError(
                    "Placement group bundle cannot have more than 1 GPU.")
            if bundle_gpus:
                gpu_bundles += 1
        if parallel_config.world_size > gpu_bundles:
            raise ValueError(
                "The number of required GPUs exceeds the total number of "
                "available GPUs in the placement group.")
    else:
        num_gpus_in_cluster = ray.cluster_resources().get("GPU", 0)
        if parallel_config.world_size > num_gpus_in_cluster:
            raise ValueError(
                "The number of required GPUs exceeds the total number of "
                "available GPUs in the cluster.")
        # Create a new placement group
        placement_group_specs = ([{"GPU": 1}] * parallel_config.world_size)
        current_placement_group = ray.util.placement_group(
            placement_group_specs)
        # Wait until PG is ready - this will block until all
        # requested resources are available, and will timeout
        # if they cannot be provisioned.
        ray.get(current_placement_group.ready(), timeout=1800)
    return current_placement_group
```

6. ./vllm_npu/vllm_npu/model_executor

    a. ./vllm_npu/vllm_npu/model_executor/ascend_model_loader.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
# Part of code in this file was copied from project [vLLM Team][vllm]
"""Utilities for selecting and loading models."""
import contextlib
import torch
import torch.nn as nn
from vllm.config import DeviceConfig, ModelConfig
from vllm.logger import init_logger
from vllm.model_executor.weight_utils import initialize_dummy_weights
from vllm_npu.model_executor.models.ascend.mindie_llm_wrapper import MindIELlmWrapper
logger = init_logger(__name__)
@contextlib.contextmanager
def _set_default_torch_dtype(dtype: torch.dtype):
```

```
    """Sets the default torch dtype to the given dtype."""
    old_dtype = torch.get_default_dtype()
    torch.set_default_dtype(dtype)
    yield
    torch.set_default_dtype(old_dtype)
def get_model(
    model_config: ModelConfig,
    device_config: DeviceConfig,
    **kwargs,
) -> nn.Module:
    """Get model with MindIELlmWrapper
    Args:
        model_config (ModelConfig): Model configs.
        device_config (DeviceConfig): Device configs.
        mindie_model_config (dict[str, Any]): MindIE configs.
    Raises:
        ValueError: LoRA not supported.
    Returns:
        nn.Module: Model.
    """
    if kwargs.get("lora_config"):
        logger.info(
            "Using LoRA(s) with MindIE backend:\n"
            "Please make sure your '--lora-modules' matches with your 'lora_adapter.json' in the
model directory!\n"
            "Current config for LoRA(s): %s",
            kwargs.get("lora_config"),
        )
    with _set_default_torch_dtype(model_config.dtype):
        with torch.device(device_config.device):
            model = MindIELlmWrapper(model_config.mindie_config)
        if model_config.load_format == "dummy":
            initialize_dummy_weights(model)
        else:
            model.load_weights(model_config.load_format)
        model = model.npu()
    model = model.eval()
    # For compatibility, move the config to the first hierarchy
    model.config = model.mindie_model.model_wrapper.model_runner.config
    return model
```

b. ./vllm_npu/vllm_npu/model_executor/utils.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
import importlib
import torch
from vllm.config import DeviceConfig, ModelConfig
DEVICE_TO_MODEL_LOADER_MAP = {
    "cuda": "vllm.model_executor.model_loader",
    "neuron": "vllm.model_executor.neuron_model_loader",
    "npu": "vllm_npu.model_executor.ascend_model_loader",
}
def get_model(model_config: ModelConfig, device_config: DeviceConfig, **kwargs) ->
torch.nn.Module:
    model_loader_module = DEVICE_TO_MODEL_LOADER_MAP[device_config.device_type]
    imported_model_loader = importlib.import_module(f"{model_loader_module}")
    get_model_fn = imported_model_loader.get_model
    return get_model_fn(model_config, device_config, **kwargs)
```

c. ./vllm_npu/vllm_npu/model_executor/__init__.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
import vllm_npu.model_executor.ascend_model_loader
import vllm_npu.model_executor.models.ascend.mindie_llm_wrapper
from vllm_npu.model_executor.utils import DEVICE_TO_MODEL_LOADER_MAP, get_model
from vllm.model_executor import utils
from vllm import model_executor
model_executor.get_model = get_model
utils.DEVICE_TO_MODEL_LOADER_MAP = DEVICE_TO_MODEL_LOADER_MAP
```

d. ./vllm_npu/vllm_npu/model_executor/layers/__init__.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
```

e. ./vllm_npu/vllm_npu/model_executor/layers/sampler.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
# Part of codes in this file was copied from project [vLLM Team][vllm]
"""A layer that samples the next tokens from the model's outputs."""
from random import randint
from typing import Dict, List, Optional, Tuple
import numpy as np
import torch
import torch.nn as nn
from mindie_llm.text_generator.adapter.generator_torch import GeneratorTorch
from mindie_llm.text_generator.utils.sampling_metadata import SamplingData, SamplingParam
from vllm.model_executor.layers.sampler import _apply_logits_processors,
_build_sampler_output, _get_logprobs
from vllm.model_executor.sampling_metadata import SamplingMetadata
from vllm.sampling_params import SamplingParams, SamplingType
from vllm.sequence import SamplerOutput
_SAMPLING_EPS = 1e-5
def _to_tensor(data, dtype=None):
    if dtype:
        return torch.tensor(data, dtype=dtype, device=torch.device("npu"))
    else:
        return torch.tensor(data, device=torch.device("npu"))
class Sampler(nn.Module):
    """Adapts the vLLM sample function to the backend sample function in MindIE."""
    def __init__(self, atb_model: GeneratorTorch) -> None:
        super().__init__()
        self.atb_model = atb_model
    def forward(self, logits: torch.Tensor, sampling_metadata: SamplingMetadata) ->
Optional[SamplerOutput]:
        _, vocab_size = logits.shape
        if not sampling_metadata.perform_sampling:
            return None
        # Apply logits processors (if any).
        logits = _apply_logits_processors(logits, sampling_metadata)
        sampling_data, sampling_param = self.get_sample_data_and_param(sampling_metadata,
vocab_size)
        logprobs = torch.log_softmax(logits, dim=-1, dtype=torch.float)
        next_tokens = self.atb_model.sample(logits, sampling_data, sampling_param)
        # Sample the next tokens.
        sample_results = self.get_sample_results(sampling_metadata, next_tokens)
        # Get the logprobs query results.
        prompt_logprobs, sample_logprobs = _get_logprobs(logprobs, sampling_metadata,
sample_results)
        return _build_sampler_output(sample_results, sampling_metadata, prompt_logprobs,
sample_logprobs)
    def get_sample_data_and_param(
        self,
        sampling_metadata: "SamplingMetadata",
        vocab_size: int,
    ) -> tuple[SamplingData, SamplingParam]:
        """Get sample data and parameter from sampling metadata.
        Args:
            sampling_metadata (SamplingMetadata): Sampling metadata.
            vocab_size (int): Vocabulary size.
        Returns:
            tuple[SamplingData, SamplingParam]: Sample data and parameter.
        """
        all_input_tokens: List[List[int]] = []
        prompt_tokens: List[List[int]] = []
        output_tokens: List[List[int]] = []
        top_ks: List[int] = []
        temperatures: List[float] = []
        top_ps: List[float] = []
        min_ps: List[float] = []
        presence_penalties: List[float] = []
        frequency_penalties: List[float] = []
        repetition_penalties: List[float] = []
        seeds: List[int] = []
        do_penalties = False
```

```
    do_top_p_top_k = False
    do_min_p = False
    for i, seq_group in enumerate(sampling_metadata.seq_groups):
        seq_ids, sampling_params = seq_group
        temperature = sampling_params.temperature
        p = sampling_params.presence_penalty
        f = sampling_params.frequency_penalty
        r = sampling_params.repetition_penalty
        top_p = sampling_params.top_p
        min_p = sampling_params.min_p
        # k should not be greater than the vocab size.
        top_k = min(sampling_params.top_k, vocab_size)
        top_k = vocab_size if top_k == -1 else top_k
        is_greedy = sampling_params.sampling_type == SamplingType.GREEDY
        seed = sampling_params.seed or (
            0 if is_greedy else randint(torch.iinfo(torch.long).min, torch.iinfo(torch.long).max)
        )
        if temperature < _SAMPLING_EPS:
            # NOTE: Zero temperature means deterministic sampling
            # (i.e., greedy sampling or beam search).
            # Set the temperature to 1 to avoid division by zero.
            temperature = 1.0
        if not do_top_p_top_k and (top_p < 1.0 - _SAMPLING_EPS or top_k != vocab_size):
            do_top_p_top_k = True
        if not do_min_p and min_p > _SAMPLING_EPS:
            do_min_p = True
        if not do_penalties and any(
            [abs(p) >= _SAMPLING_EPS, abs(f) >= _SAMPLING_EPS, abs(r - 1.0) >=
_SAMPLING_EPS]
        ):
            do_penalties = True
        if i < sampling_metadata.num_prompts and sampling_params.prompt_logprobs is not
None:
            # For tokens in the prompt that we only need to get their logprobs
            prompt_len = sampling_metadata.prompt_lens[i]
            temperatures += [temperature] * (prompt_len - 1)
            seeds += [seed] * (prompt_len - 1)
            top_ps += [top_p] * (prompt_len - 1)
            top_ks += [top_k] * (prompt_len - 1)
            min_ps += [min_p] * (prompt_len - 1)
            presence_penalties += [0] * (prompt_len - 1)
            frequency_penalties += [0] * (prompt_len - 1)
            repetition_penalties += [1] * (prompt_len - 1)
            prompt_tokens.extend([] for _ in range(prompt_len - 1))
            output_tokens.extend([] for _ in range(prompt_len - 1))
            all_input_tokens.extend([] for _ in range(prompt_len - 1))
        for seq_id in seq_ids:
            seq_data = sampling_metadata.seq_data[seq_id]
            prompt_tokens.append(seq_data.prompt_token_ids)
            output_tokens.append(seq_data.output_token_ids)
            all_input_tokens.append(seq_data.prompt_token_ids + seq_data.output_token_ids)
        temperatures += [temperature] * len(seq_ids)
        top_ps += [top_p] * len(seq_ids)
        top_ks += [top_k] * len(seq_ids)
        min_ps += [min_p] * len(seq_ids)
        presence_penalties += [p] * len(seq_ids)
        frequency_penalties += [f] * len(seq_ids)
        repetition_penalties += [r] * len(seq_ids)
    max_tokens_len = max([len(tokens) for tokens in all_input_tokens], default=0)
    padded_all_input_tokens = [
        tokens + [vocab_size] * (max_tokens_len - len(tokens))
        for tokens in all_input_tokens
    ]
    output_max_len = max([len(tokens) for tokens in output_tokens], default=0)
    padded_output_tokens = [tokens + [vocab_size] * (output_max_len - len(tokens)) for
tokens in output_tokens]
    # To use from_numpy or not to, this is a question.
    # Convert to tensor if not None
    all_input_ids = padded_all_input_tokens and _to_tensor(padded_all_input_tokens,
```



```
torch.int32)
        output_ids = padded_output_tokens and _to_tensor(padded_output_tokens, torch.int32)
        sampling_data = SamplingData(all_input_ids=all_input_ids, output_ids=output_ids)
        repetition_penalties = np.array(repetition_penalties, dtype=np.float32)
        frequency_penalties = np.array(frequency_penalties, dtype=np.float32)
        presence_penalties = np.array(presence_penalties, dtype=np.float32)
        temperatures = np.array(temperatures, dtype=np.float32)
        top_ks = np.array(top_ks, dtype=np.int32)
        top_ps = np.array(top_ps, dtype=np.float32)
        seeds = np.array(seeds, dtype=np.int32)
        do_sample = np.array([True])
        # Temporary solution, need to find out a better way to deal with this
        if is_greedy:
            sampling_param = None
        else:
            sampling_param = SamplingParam.from_numpy(
                repetition_penalty=repetition_penalties,
                frequency_penalty=frequency_penalties,
                presence_penalty=presence_penalties,
                temperature=temperatures,
                top_k=top_ks,
                top_p=top_ps,
                seed=seeds,
                do_sample=do_sample,
                to_tensor=_to_tensor,
            )
        return sampling_data, sampling_param
    def get_sample_results(
        self,
        sampling_metadata: SamplingMetadata,
        samples: np.array,
    ) -> List[Tuple[List[int], List[int]]]:
        """Reconstructs the sample results from the backend samples.
        Args:
            sampling_metadata (SamplingMetadata): Sampling metadata.
            samples (np.array): Backend samples.
        Raises:
            ValueError: Beam search not supported temporarily.
        Returns:
            List[Tuple[List[int], List[int]]]: Sample results.
        """
        categorized_seq_group_ids = {t: [] for t in SamplingType}
        categorized_sample_indices = sampling_metadata.categorized_sample_indices
        for i, seq_group in enumerate(sampling_metadata.seq_groups):
            _, sampling_params = seq_group
            sampling_type = sampling_params.sampling_type
            categorized_seq_group_ids[sampling_type].append(i)
        sample_results_dict: Dict[int, Tuple[List[int], List[int]]] = {}
        sample_metadata = {}
        # Counterintiutively, having two loops here is actually faster.
        # The first loop can run without waiting on GPU<->CPU sync.
        for sampling_type in SamplingType:
            if len(categorized_sample_indices[sampling_type]) == 0:
                continue
            seq_group_ids = categorized_seq_group_ids[sampling_type]
            seq_groups = [sampling_metadata.seq_groups[i] for i in seq_group_ids]
            sample_metadata[sampling_type] = (seq_group_ids, seq_groups)
        for sampling_type in SamplingType:
            if sampling_type not in sample_metadata:
                continue
            seq_group_ids, seq_groups = sample_metadata[sampling_type]
            if sampling_type in (SamplingType.GREEDY, SamplingType.RANDOM,
SamplingType.RANDOM_SEED):
                sample_results = self.extract_next_tokens_and_parents(seq_groups, samples)
            elif sampling_type == SamplingType.BEAM:
                raise ValueError(f"Unsupported sampling type: beam search.")
            sample_results_dict.update(zip(seq_group_ids, sample_results))
        sample_results = [sample_results_dict.get(i) for i in
range(len(sampling_metadata.seq_groups))]
```



```
        return sample_results
    def extract_next_tokens_and_parents(
        self,
        selected_seq_groups: List[Tuple[List[int], SamplingParams]],
        samples: np.array,
    ) -> List[Tuple[List[int], List[int]]]:
        samples = samples.tolist()
        sample_idx = 0
        results = []
        for seq_group in selected_seq_groups:
            seq_ids, _ = seq_group
            num_parent_seqs = len(seq_ids)
            parent_ids = list(range(num_parent_seqs))
            next_token_ids = [samples[sample_idx]]
            results.append((next_token_ids, parent_ids))
            sample_idx += num_parent_seqs
        return results
```

f. ./vllm_npu/vllm_npu/model_executor/models/ascend/__init__.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
```

g. ./vllm_npu/vllm_npu/model_executor/models/ascend/mindie_llm_wrapper.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
import math
from typing import List, Literal, Optional
import torch
from mindie_llm.text_generator.adapter.generator_torch import GeneratorTorch
from torch import nn
from vllm.lora.request import LoRARequest
from vllm.model_executor import SamplingMetadata
from vllm.model_executor.input_metadata import InputMetadata
from vllm.sequence import SamplerOutput
from vllm.worker.cache_engine import KVCache
from vllm_npu.model_executor.layers.sampler import Sampler
DEVICE_TYPE = "npu"
DEFAULT_FORMAT = "auto"
class MindIELlmWrapper(nn.Module):
    def __init__(self, mindie_config):
        super(MindIELlmWrapper, self).__init__()
        self.mindie_config = mindie_config
        self.mindie_model = None
        self.sampler = None
    def forward(
        self,
        input_ids: torch.Tensor,
        positions: torch.Tensor,
        kv_caches: List[KVCache],
        input_metadata: InputMetadata,
        lora_requests: List[LoRARequest],
    ) -> torch.Tensor:
        if kv_caches[0][0] is None:
            kv_caches, block_tables, slots = self.create_dummy_kv_cache(input_metadata, input_ids)
        else:
            if input_metadata.is_prompt:
                block_tables = torch.tensor([0], dtype=torch.int32, device="npu")
            else:
                block_tables = input_metadata.block_tables
            slots = input_metadata.slot_mapping
        if input_metadata.is_prompt:
            input_lengths = input_metadata.prompt_lens.to(torch.int32)
            max_seq_len = int(input_metadata.prompt_lens.max())
            lm_head_indices = (input_metadata.prompt_lens.cumsum(dim=-1) - 1).to(torch.int64)
        else:
            input_lengths = input_metadata.context_lens
            max_seq_len = input_metadata.max_context_len
            lm_head_indices = None
        adapter_ids = [lora_request.lora_name if lora_request else None for lora_request in
lora_requests]
        logits = self.mindie_model.forward_tensor(
```

```
            input_ids,
            positions,
            input_metadata.is_prompt,
            kv_caches,
            block_tables,
            slots,
            input_lengths,
            max_seq_len,
            lm_head_indices,
            adapter_ids=adapter_ids,  # batch_size len
        )
        return logits
    def sample(
        self,
        hidden_states: torch.Tensor,
        sampling_metadata: SamplingMetadata,
    ) -> Optional[SamplerOutput]:
        # hidden_states is logits
        next_tokens = self.sampler(hidden_states, sampling_metadata)
        return next_tokens
    def load_weights(self, load_format: Literal["auto", "safetensors", "pt"] = DEFAULT_FORMAT):
        """Load model weights.
        Args:
            load_format (Literal[auto, safetensors, pt], optional): The format of weights. Defaults to
"auto".
        Raises:
            ValueError: Format not supported.
        """
        if load_format not in ["auto", "safetensors", "pt"]:
            raise ValueError("load-format support [safetensors, pt]")
        self.weight_dtype = torch.get_default_dtype()
        torch.set_default_dtype(torch.float32)
        # Replacing the deprecated method with GeneratorTorch
        self.mindie_model = GeneratorTorch(self.mindie_config)
        self.sampler = Sampler(self.mindie_model)
        self.sampler.logits_as_hidden_states = True
        torch.set_default_dtype(self.weight_dtype)
    # when warmup, create dummy kvcache, block_tables, slot_mapping
    def create_dummy_kv_cache(self, input_metadata, input_ids):
        dummy_block_num = 1
        dummy_block_size = 128
        kv_cache = [
            (
                torch.empty(
                    (
                        dummy_block_num,
                        dummy_block_size,
                        self.mindie_model.model_info.num_kv_heads,
                        self.mindie_model.model_info.head_size,
                    ),
                    dtype=self.weight_dtype,
                    device=DEVICE_TYPE,
                ),
                torch.empty(
                    (
                        dummy_block_num,
                        dummy_block_size,
                        self.mindie_model.model_info.num_kv_heads,
                        self.mindie_model.model_info.head_size,
                    ),
                    dtype=self.weight_dtype,
                    device=DEVICE_TYPE,
                ),
            )
            for _ in range(self.mindie_model.model_info.num_layers)
        ]
        max_s = max(input_metadata.prompt_lens)
        max_need_block = math.ceil(max_s / dummy_block_size)
        batch_size = len(input_metadata.prompt_lens)
```



```
        block_tables = torch.zeros(batch_size, max_need_block, dtype=int, device=DEVICE_TYPE)
        slot = [i for i in range(dummy_block_size)]
        slots = []
        warm_up_len = len(input_ids)
        while warm_up_len > 0:
            if warm_up_len > dummy_block_size:
                slots.extend(slot)
                warm_up_len -= dummy_block_size
            else:
                slots.extend(slot[:warm_up_len])
                warm_up_len = 0
        slots = torch.tensor(slots, dtype=torch.long, device=DEVICE_TYPE)
        return kv_cache, block_tables, slots
```

h. ./vllm_npu/vllm_npu/model_executor/models/__init__.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
```

7. ./vllm_npu/vllm_npu/worker

a. ./vllm_npu/vllm_npu/worker/ascend_worker.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
# Part of codes in this file was copied from project [vLLM Team][vllm]
"""A Ascend worker class."""
import gc
from typing import Dict, List, Optional, Set, Tuple
import torch
import torch.distributed
from mindie_llm.modeling.backend_type import BackendType
from vllm.config import (
    CacheConfig,
    DeviceConfig,
    LoRAConfig,
    ModelConfig,
    ParallelConfig,
    SchedulerConfig,
)
from vllm.lora.request import LoRARequest
from vllm.model_executor import set_random_seed
from vllm.model_executor.parallel_utils.communication_op import broadcast_tensor_dict
from vllm.model_executor.parallel_utils.custom_all_reduce import init_custom_ar
from vllm.model_executor.parallel_utils.parallel_state import ensure_model_parallel_initialized
from vllm.sequence import SamplerOutput, SequenceGroupMetadata
from vllm.worker.cache_engine import CacheEngine
from vllm.worker.model_runner import ModelRunner
class Worker:
    """A worker class that executes the model on a group of NPUs."""
    def __init__(
        self,
        model_config: ModelConfig,
        parallel_config: ParallelConfig,
        scheduler_config: SchedulerConfig,
        device_config: DeviceConfig,
        local_rank: int,
        rank: int,
        distributed_init_method: str,
        lora_config: Optional[LoRAConfig] = None,
        kv_cache_dtype: Optional[str] = "auto",
        is_driver_worker: bool = False,
    ) -> None:
        self.model_config = model_config
        self.parallel_config = parallel_config
        self.scheduler_config = scheduler_config
        self.device_config = device_config
        self.local_rank = local_rank
        self.rank = rank
        self.distributed_init_method = distributed_init_method
        self.lora_config = lora_config
        self.is_driver_worker = is_driver_worker
        if self.is_driver_worker and self.rank != 0:
            raise ValueError("The driver worker must have rank 0.")
        self.device = None
```

```
        self.model_runner = ModelRunner(
            model_config,
            parallel_config,
            scheduler_config,
            device_config,
            self.lora_config,
            kv_cache_dtype,
            is_driver_worker,
        )
        # Uninitialized cache engine. Will be initialized by
        # self.init_cache_engine().
        self.cache_config = None
        self.cache_engine = None
        self.cache_events = None
        self.gpu_cache = None
    def init_model(self, cupy_port: Optional[int] = None) -> None:
        self.device = torch.device(f"npu:{self.local_rank}")
        torch.npu.set_device(self.device)
        # Initialize the distributed environment.
        init_distributed_environment(self.parallel_config, self.rank, self.distributed_init_method)
        # Initialize the model.
        set_random_seed(self.model_config.seed)
    def load_model(self):
        # Add an attribute dynamically, not graceful, but useful
        self.model_config.mindie_config = {
            "backend_type": BackendType.ATB,
            "model_id": self.model_config.model,
            "rank": self.rank,
            "local_rank": self.local_rank,
            "world_size": self.parallel_config.world_size,
            "npu_device_id": self.local_rank,
        }
        self.model_runner.load_model()
        del self.model_config.mindie_config
    @torch.inference_mode()
    def profile_num_available_blocks(
        self, block_size: int, gpu_memory_utilization: float, cpu_swap_space: int, cache_dtype: str
    ) -> Tuple[int, int]:
        """Profiles the peak memory usage of the model and returns the maximum
        number of GPU and CPU cache blocks that can be allocated.
        Args:
            block_size: The size of the cache block.
            gpu_memory_utilization: The fraction of the total GPU memory to use.
            cpu_swap_space: The size of the CPU swap space in bytes.
        """
        # Profile the memory usage of the model and get the maximum number of
        # cache blocks that can be allocated with the remaining free memory.
        torch.npu.empty_cache()
        torch.npu.reset_peak_memory_stats()
        # Execute a forward pass with dummy inputs to profile the memory usage
        # of the model.
        self.model_runner.profile_run()
        dummy_block_size = 128
        dummy_num_blocks = dummy_block_size // block_size
        # Calculate the number of blocks that can be allocated with the
        # profiled peak memory.
        torch.npu.synchronize()
        peak_memory = torch.npu.max_memory_allocated()
        total_gpu_memory = torch.npu.get_device_properties(self.rank).total_memory
        cache_block_size = CacheEngine.get_cache_block_size(
            block_size, cache_dtype, self.model_config, self.parallel_config
        )
        num_gpu_blocks = (
            int((total_gpu_memory * gpu_memory_utilization - peak_memory) // cache_block_size)
+ dummy_num_blocks
        )
        num_cpu_blocks = int(cpu_swap_space // cache_block_size)
        num_gpu_blocks = max(num_gpu_blocks, 0)
        num_cpu_blocks = max(num_cpu_blocks, 0)
```



```
        if self.model_runner.lora_manager:
            self.model_runner.remove_all_loras()
        gc.collect()
        torch.npu.empty_cache()
        return num_gpu_blocks, num_cpu_blocks
    def init_cache_engine(self, cache_config: CacheConfig) -> None:
        self.cache_config = cache_config
        self.cache_engine = CacheEngine(self.cache_config, self.model_config, self.parallel_config)
        self.cache_events = self.cache_engine.events
        self.gpu_cache = self.cache_engine.gpu_cache
        self.model_runner.set_block_size(self.cache_engine.block_size)
    def warm_up_model(self) -> None:
        pass
    def cache_swap(
        self,
        blocks_to_swap_in: Dict[int, int],
        blocks_to_swap_out: Dict[int, int],
        blocks_to_copy: Dict[int, List[int]],
    ) -> None:
        # Issue cache operations.
        issued_cache_op = False
        if blocks_to_swap_in:
            self.cache_engine.swap_in(blocks_to_swap_in)
            issued_cache_op = True
        if blocks_to_swap_out:
            self.cache_engine.swap_out(blocks_to_swap_out)
            issued_cache_op = True
        if blocks_to_copy:
            self.cache_engine.copy(blocks_to_copy)
            issued_cache_op = True
        cache_events = self.cache_events if issued_cache_op else None
        if cache_events is not None:
            for event in cache_events:
                event.wait()
    @torch.inference_mode()
    def execute_model(
        self,
        seq_group_metadata_list: Optional[List[SequenceGroupMetadata]] = None,
        blocks_to_swap_in: Optional[Dict[int, int]] = None,
        blocks_to_swap_out: Optional[Dict[int, int]] = None,
        blocks_to_copy: Optional[Dict[int, List[int]]] = None,
    ) -> Optional[SamplerOutput]:
        if self.is_driver_worker:
            if seq_group_metadata_list is None:
                raise ValueError("seq_group_metadata_list must be provided for the driver worker.")
            num_seq_groups = len(seq_group_metadata_list)
            if any(x is None for x in [blocks_to_swap_in, blocks_to_swap_out, blocks_to_copy]):
                raise ValueError(
                    "blocks_to_swap_in, blocks_to_swap_out, and blocks_to_copy must be provided for
the driver worker."
                )
            data = {
                "num_seq_groups": num_seq_groups,
                "blocks_to_swap_in": blocks_to_swap_in,
                "blocks_to_swap_out": blocks_to_swap_out,
                "blocks_to_copy": blocks_to_copy,
            }
            broadcast_tensor_dict(data, src=0)
        else:
            data = broadcast_tensor_dict(src=0)
            num_seq_groups = data["num_seq_groups"]
            blocks_to_swap_in = data["blocks_to_swap_in"]
            blocks_to_swap_out = data["blocks_to_swap_out"]
            blocks_to_copy = data["blocks_to_copy"]
        self.cache_swap(blocks_to_swap_in, blocks_to_swap_out, blocks_to_copy)
        # If there is no input, we don't need to execute the model.
        if num_seq_groups == 0:
            return {}
        output = self.model_runner.execute_model(seq_group_metadata_list, self.gpu_cache)
```



```
        return output
    def add_lora(self, lora_request: LoRARequest) -> bool:
        return self.model_runner.add_lora(lora_request)
    def remove_lora(self, lora_id: int) -> bool:
        return self.model_runner.remove_lora(lora_id)
    def list_loras(self) -> Set[int]:
        return self.model_runner.list_loras()
def init_distributed_environment(
    parallel_config: ParallelConfig,
    rank: int,
    distributed_init_method: Optional[str] = None,
) -> None:
    """Initialize the distributed environment."""
    if torch.distributed.is_initialized():
        torch_world_size = torch.distributed.get_world_size()
        if torch_world_size != parallel_config.world_size:
            raise RuntimeError(
                "torch.distributed is already initialized but the torch world "
                "size does not match parallel_config.world_size "
                f"({torch_world_size} vs. {parallel_config.world_size})."
            )
    elif not distributed_init_method:
        raise ValueError(
            "distributed_init_method must be set if torch.distributed "
            f"is not already initialized: {distributed_init_method}"
        )
    else:
        torch.distributed.init_process_group(
            backend="nccl",
            world_size=parallel_config.world_size,
            rank=rank,
            init_method=distributed_init_method,
        )
    ensure_model_parallel_initialized(parallel_config.tensor_parallel_size,
parallel_config.pipeline_parallel_size)
    # Initialize a custom fast all-reduce implementation.
    if not parallel_config.disable_custom_all_reduce:
        init_custom_ar()
```

b. ./vllm_npu/vllm_npu/worker/cache_engine.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
from typing import Tuple
def get_key_block_shape(self) -> Tuple[int, int, int]:
    return (
        self.block_size,
        self.num_heads,
        self.head_size,
    )
def get_value_block_shape(self) -> Tuple[int, int, int]:
    return (
        self.block_size,
        self.num_heads,
        self.head_size,
    )
```

c. ./vllm_npu/vllm_npu/worker/__init__.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
import vllm_npu.worker.ascend_worker
from vllm_npu.worker.cache_engine import get_key_block_shape, get_value_block_shape
from vllm_npu.worker.model_runner import (
    _make_tensor_with_pad,
    _prepare_decode,
    _prepare_prompt,
    execute_model,
    load_model,
    profile_run,
)
from vllm.worker import cache_engine, model_runner
cache_engine.CacheEngine.get_key_block_shape = get_key_block_shape
cache_engine.CacheEngine.get_value_block_shape = get_value_block_shape
```



```
model_runner._make_tensor_with_pad = _make_tensor_with_pad
model_runner.ModelRunner._prepare_decode = _prepare_decode
model_runner.ModelRunner._prepare_prompt = _prepare_prompt
model_runner.ModelRunner.execute_model = execute_model
model_runner.ModelRunner.load_model = load_model
model_runner.ModelRunner.profile_run = profile_run
```

d. ./vllm_npu/vllm_npu/worker/model_runner.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
# Part of codes in this file was copied from project [vLLM Team][vllm]
from collections import namedtuple
from typing import List, Optional, Set, Tuple, Union, NamedTuple
import torch
from vllm.lora.request import LoRARequest
from vllm.model_executor import InputMetadata, get_model
from vllm.sampling_params import SamplingParams
from vllm.sequence import SamplerOutput, SequenceData, SequenceGroupMetadata
from vllm.worker.model_runner import _BATCH_SIZES_TO_CAPTURE, _PAD_SLOT_ID,
_pad_to_max
class PreparedPromptData(NamedTuple):
    tokens: torch.Tensor
    positions: torch.Tensor
    metadata: InputMetadata
    prompt_lens: List[int]
    subquery_lens: List[int]
    lora_index_mapping: List[int]
    lora_prompt_mapping: List[int]
    requests: List[LoRARequest]
class PreparedDecodeData(NamedTuple):
    tokens: torch.Tensor
    positions: torch.Tensor
    metadata: InputMetadata
    index_mapping: List[int]
    prompt_mapping: List[int]
    requests: List[LoRARequest]
def load_model(self) -> None:
    self.model = get_model(
        self.model_config,
        self.device_config,
        lora_config=self.lora_config,
        parallel_config=self.parallel_config,
        scheduler_config=self.scheduler_config,
    )
def _prepare_prompt(
    self,
    seq_group_metadata_list: List[SequenceGroupMetadata],
) -> PreparedPromptData:
    assert len(seq_group_metadata_list) > 0
    input_tokens: List[List[int]] = []
    input_positions: List[List[int]] = []
    slot_mapping: List[List[int]] = []
    lora_index_mapping: List[int] = []
    lora_prompt_mapping: List[int] = []
    lora_requests: List[LoRARequest] = []
    prompt_lens: List[int] = []
    context_lens: List[int] = []
    subquery_lens: List[int] = []
    prefix_block_tables: List[List[int]] = []
    for seq_group_metadata in seq_group_metadata_list:
        assert seq_group_metadata.is_prompt
        seq_ids = list(seq_group_metadata.seq_data.keys())
        assert len(seq_ids) == 1
        seq_id = seq_ids[0]
        seq_data = seq_group_metadata.seq_data[seq_id]
        prompt_tokens = seq_data.get_token_ids()
        prompt_len = len(prompt_tokens)
        prompt_lens.append(prompt_len)
        prefix_len = 0
        prefix = seq_group_metadata.prefix
        if prefix is not None and prefix.computed:
```

```
            prefix_len = prefix.get_length()
            prompt_tokens = prompt_tokens[prefix_len:]
            prefix_block_tables.append(prefix.get_block_numbers())
        else:
            prefix_block_tables.append([])
        # actual prompt lens
        context_lens.append(prefix_len)
        subquery_lens.append(prompt_len - prefix_len)
        input_tokens.append(prompt_tokens)
        # NOTE(woosuk): Here we assume that the first token in the prompt
        # is always the first token in the sequence.
        input_positions.append(list(range(prefix_len, prefix_len + len(prompt_tokens))))
        lora_id = seq_group_metadata.lora_int_id
        lora_requests.append(seq_group_metadata.lora_request)
        lora_index_mapping.append([lora_id] * (prompt_len - prefix_len))
        lora_prompt_mapping.extend(
            [lora_id] * (prompt_len - prefix_len if
seq_group_metadata.sampling_params.prompt_logprobs else 1)
        )
        if seq_group_metadata.block_tables is None:
            # During memory profiling, the block tables are not initialized
            # yet. In this case, we just use a dummy slot mapping.
            slot_mapping.append([_PAD_SLOT_ID] * prompt_len)
            continue
        # Compute the slot mapping.
        slot_mapping.append([])
        block_table = seq_group_metadata.block_tables[seq_id]
        # Mask the [0, start_idx) tokens of the prompt with _PAD_SLOT_ID,
        # where start_idx is max(0, prompt_len - sliding_window).
        # For example, if the prompt len is 10, sliding window is 8, and
        # block size is 4, the first two tokens are masked and the slot
        # mapping will be [-1, -1, 2, 3, 4, 5, 6, 7, 0, 1].
        start_idx = 0
        if self.sliding_window is not None:
            assert prefix_len == 0, "Prefix caching is currently not supported with " "sliding window
attention"
            start_idx = max(0, prompt_len - self.sliding_window)
        for i in range(prefix_len, prompt_len):
            if i < start_idx:
                slot_mapping[-1].append(_PAD_SLOT_ID)
                continue
            block_number = block_table[i // self.block_size]
            block_offset = i % self.block_size
            slot = block_number * self.block_size + block_offset
            slot_mapping[-1].append(slot)
    max_prompt_len = max(subquery_lens)
    input_tokens = _make_tensor_with_pad(input_tokens, max_prompt_len, pad=0,
dtype=torch.long, device=self.device)
    input_positions = _make_tensor_with_pad(
        input_positions, max_prompt_len, pad=0, dtype=torch.long, device=self.device
    )
    slot_mapping = _make_tensor_with_pad(
        slot_mapping, max_prompt_len, pad=_PAD_SLOT_ID, dtype=torch.long, device=self.device
    )
    lora_index_mapping = [_pad_to_max(mapping, max_prompt_len, pad=0) for mapping in
lora_index_mapping]
    context_lens_tensor = torch.tensor(context_lens, dtype=torch.int, device=self.device)
    # Prepare prefix block tables
    max_prompt_block_table_len = max(len(t) for t in prefix_block_tables)
    block_tables = _make_tensor_with_pad(
        prefix_block_tables, max_len=max_prompt_block_table_len, pad=0, dtype=torch.int,
device=self.device
    )
    start_loc_tensor = torch.arange(
        0, len(prompt_lens) * max_prompt_len, max_prompt_len, dtype=torch.long,
device=self.device
    )
    prompt_lens_tensor = torch.tensor(prompt_lens, dtype=torch.long, device=self.device)
    input_metadata = InputMetadata(
```



```
            is_prompt=True,
            slot_mapping=slot_mapping,
            prompt_lens=prompt_lens_tensor,
            max_seq_len=max_prompt_len,
            start_loc=start_loc_tensor,
            max_context_len=None,
            context_lens=context_lens_tensor,
            block_tables=block_tables,
            use_cuda_graph=False,
            kv_cache_dtype=self.kv_cache_dtype,
        )
        return PreparedPromptData(
            input_tokens,
            input_positions,
            input_metadata,
            prompt_lens,
            subquery_lens,
            lora_index_mapping,
            lora_prompt_mapping,
            lora_requests,
        )
    def _prepare_decode(self, seq_group_metadata_list: List[SequenceGroupMetadata]) ->
    PreparedDecodeData:
        if not seq_group_metadata_list:
            raise ValueError("seq_group_metadata_list is empty")
        input_tokens: List[List[int]] = []
        input_positions: List[List[int]] = []
        slot_mapping: List[List[int]] = []
        context_lens: List[int] = []
        block_tables: List[List[int]] = []
        lora_index_mapping: List[int] = []
        lora_prompt_mapping: List[int] = []
        lora_requests: List[LoRARequest] = []
        max_num_blocks_per_seq = 0
        for seq_group_metadata in seq_group_metadata_list:
            if seq_group_metadata.is_prompt:
                raise ValueError("seq_group_metadata.is_prompt is True")
            seq_ids = list(seq_group_metadata.seq_data.keys())
            lora_id = seq_group_metadata.lora_int_id
            lora_requests.append(seq_group_metadata.lora_request)
            for seq_id in seq_ids:
                seq_data = seq_group_metadata.seq_data[seq_id]
                generation_token = seq_data.get_last_token_id()
                input_tokens.append([generation_token])
                seq_len = seq_data.get_len()
                position = seq_len - 1
                input_positions.append([position])
                context_len = seq_len if self.sliding_window is None else min(seq_len,
    self.sliding_window)
                context_lens.append(context_len)
                block_table = seq_group_metadata.block_tables[seq_id]
                max_num_blocks_per_seq = max(max_num_blocks_per_seq, len(block_table))
                block_number = block_table[position // self.block_size]
                block_offset = position % self.block_size
                slot = block_number * self.block_size + block_offset
                slot_mapping.append([slot])
                lora_index_mapping.append([lora_id])
                lora_prompt_mapping.append(lora_id)
                if self.sliding_window is not None:
                    sliding_window_blocks = self.sliding_window // self.block_size
                    block_table = block_table[-sliding_window_blocks:]
                block_tables.append(block_table)
        batch_size = len(input_tokens)
        max_context_len = max(context_lens)
        use_captured_graph = (
            not self.model_config.enforce_eager
            and batch_size <= _BATCH_SIZES_TO_CAPTURE[-1]
            and max_context_len <= self.max_context_len_to_capture
        )
```



```
    input_tokens = _make_tensor_with_pad(input_tokens, max_len=1, pad=0, dtype=torch.long,
device=self.device)
    input_positions = _make_tensor_with_pad(input_positions, max_len=1, pad=0,
dtype=torch.long, device=self.device)
    slot_mapping = _make_tensor_with_pad(
        slot_mapping, max_len=1, pad=_PAD_SLOT_ID, dtype=torch.long, device=self.device
    )
    context_lens = torch.tensor(context_lens, dtype=torch.int, device=self.device)
    if use_captured_graph:
        # The shape of graph_block_tables is
        # [max batch size, max context len // block size].
        input_block_tables = self.graph_block_tables[:batch_size]
        for i, block_table in enumerate(block_tables):
            if block_table:
                input_block_tables[i, : len(block_table)] = block_table
        block_tables = torch.tensor(input_block_tables, device=self.device)
    else:
        padded_block_tables = [_pad_to_max(block_table, max_num_blocks_per_seq, 0) for
block_table in block_tables]
        block_tables = torch.tensor(padded_block_tables, dtype=torch.int, device="npu")
    lora_index_mapping = [_pad_to_max(mapping, 1, pad=0) for mapping in lora_index_mapping]
    input_metadata = InputMetadata(
        is_prompt=False,
        slot_mapping=slot_mapping,
        prompt_lens=None,
        max_seq_len=None,
        start_loc=None,
        max_context_len=max_context_len,
        context_lens=context_lens,
        block_tables=block_tables,
        use_cuda_graph=use_captured_graph,
        kv_cache_dtype=self.kv_cache_dtype,
    )
    return PreparedDecodeData(
        input_tokens, input_positions, input_metadata, lora_index_mapping, lora_prompt_mapping,
lora_requests
    )
@torch.inference_mode()
def execute_model(
    self,
    seq_group_meta_list: Optional[List[SequenceGroupMetadata]],
    kv_caches: List[Tuple[torch.Tensor, torch.Tensor]],
) -> Optional[SamplerOutput]:
    (tokens, positions, input_meta, sampling_meta, lora_requests, _) =
self.prepare_input_tensors(seq_group_meta_list)
    hidden_states = self.model(
        input_ids=tokens,
        positions=positions,
        kv_caches=kv_caches,
        input_metadata=input_meta,
        lora_requests=lora_requests,
    )
    # Sample the next token.
    output = self.model.sample(hidden_states=hidden_states,
sampling_metadata=sampling_meta)
    return output
@torch.inference_mode()
def profile_run(self) -> None:
    # Enable top-k sampling to reflect the accurate memory usage.
    vocab_size = self.model_config.get_vocab_size()
    sampling_params = SamplingParams(top_p=0.99, top_k=vocab_size - 1)
    max_num_batched_tokens = self.scheduler_config.max_num_batched_tokens
    max_num_seqs = self.scheduler_config.max_num_seqs
    # Profile memory usage with max_num_sequences sequences and the total
    # number of tokens equal to max_num_batched_tokens.
    seqs: List[SequenceGroupMetadata] = []
    for group_id in range(max_num_seqs):
        seq_len = max_num_batched_tokens // max_num_seqs + (group_id <
max_num_batched_tokens % max_num_seqs)
```



```
        seq_data = SequenceData([0] * seq_len)
        seq = SequenceGroupMetadata(
            request_id=str(group_id),
            is_prompt=True,
            seq_data={group_id: seq_data},
            sampling_params=sampling_params,
            block_tables=None,
            lora_request=None,  # Warmup for LoRA has been done in the MindIE backend
        )
        seqs.append(seq)
    # Run the model with the dummy inputs.
    num_layers = self.model_config.get_num_layers(self.parallel_config)
    kv_caches = [(None, None)] * num_layers
    self.execute_model(seqs, kv_caches)
    torch.cuda.synchronize()
    return
def _make_tensor_with_pad(
    x: List[List[int]], max_len: int, pad: int, dtype: torch.dtype, device: Optional[Union[str,
torch.device]]
) -> torch.Tensor:
    unpad_x = [i for l in x for i in l]
    return torch.tensor(unpad_x, dtype=dtype, device=device)
```

8. 其他文件

a. ./install.sh

```
#!/bin/bash
if [ -d "./vllm" ]; then
    echo "./vllm directory has already exist!"
    exit 1
fi
git clone -b v0.3.3 https://github.com/vllm-project/vllm.git vllm
cp -r cover/* vllm/
cd vllm
pip install -r requirements.txt
python3 setup.py install
cd ../vllm_npu
pip install -r requirements.txt
python3 setup.py install
```

b. ./README.md

```
# Vllm-MindIE
#### 介绍
昇腾推理引擎对接Vllm开源框架v0.3.3稳定版本补丁
#### 软件架构
软件架构说明
#### 安装教程
确保昇腾推理基础环境安装完成后，执行`install.sh`文件即可完成vllm及昇腾补丁的安装：
```sh
bash install.sh
```
使用说明
这里提供了vllm离线模式与在线服务的启动demo作为参考。
*   离线模式：
    ```sh
    bash examples/test_offine.sh
    ```
*   在线服务：
    ```sh
    bash examples/start_server.sh
    ```
```

c. ./requirements.txt

```
numpy
decorator
attrs
psutil
absl-py
cloudpickle
scipy
tornado
```



```
transformers
accelerate
pandas
ray == 2.9.3
```

d. ./setup.py

```
import io
import os
import re
from typing import List
import setuptools
ROOT_DIR = os.path.dirname(__file__)
def get_path(*filepath) -> str:
    return os.path.join(ROOT_DIR, *filepath)
def find_version(filepath: str):
    """Extract version information from the given filepath.
    """
    with open(filepath) as fp:
        version_match = re.search(r"^__version__ = ['\"]([^'\"]*)['\"]",
                                  fp.read(), re.M)
        if version_match:
            return version_match.group(1)
        raise RuntimeError("Unable to find version string.")
def read_readme() -> str:
    """Read the README file."""
    return io.open(get_path("README.md"), "r", encoding="utf-8").read()
def get_requirements() -> List[str]:
    """Get Python package dependencies from requirements.txt."""
    with open(get_path("requirements.txt")) as f:
        requirements = f.read().strip().split("\n")
    return requirements
setuptools.setup(
    name="vllm_npu",
    version=find_version(get_path("vllm_npu", "__init__.py")),
    author="Huawei",
    license="Apache 2.0",
    description=("A high-throughput and memory-efficient inference and "
                 "serving engine for LLMs"),
    long_description=read_readme(),
    long_description_content_type="text/markdown",
    url="",
    project_urls={
        "Homepage": "",
        "Documentation": "",
    },
    classifiers=[
        "Programming Language :: Python :: 3.8",
        "Programming Language :: Python :: 3.9",
        "Programming Language :: Python :: 3.10",
        "Programming Language :: Python :: 3.11",
        "Topic :: Scientific/Engineering :: Artificial Intelligence",
    ],
    packages=setuptools.find_packages(exclude=("benchmarks", "examples", "tests")),
    python_requires=">=3.8",
    install_requires=get_requirements(),
)
```

e. ./vllm_npu/config.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
# Part of codes in this file was copied from project [vLLM Team][vllm]
from typing import Optional
import torch
from transformers import PretrainedConfig
from vllm.logger import init_logger
from vllm.utils import is_neuron
logger = init_logger(__name__)
class DeviceConfig:
    def __init__(self, device: str = "auto") -> None:
        if device == "auto":
            # Automated device type detection
```

```
            if getattr(torch.version, "cann", None) is not None:
                self.device_type = "npu"
            elif torch.cuda.is_available():
                self.device_type = "cuda"
            elif is_neuron():
                self.device_type = "neuron"
            else:
                raise RuntimeError("No supported device detected.")
        else:
            # Device type is assigned explicitly
            self.device_type = device
        # Some device types require processing inputs on CPU
        if self.device_type in ["neuron"]:
            self.device = torch.device("cpu")
        else:
            # Set device with device type
            self.device = torch.device(self.device_type)
    @property
    def is_neuron(self):
        return self.device_type == "neuron"
def _get_and_verify_max_len(
    hf_config: PretrainedConfig,
    max_model_len: Optional[int],
) -> int:
    """Get and verify the model's maximum length."""
    derived_max_model_len = float("inf")
    possible_keys = [
        # OPT
        "max_position_embeddings",
        # GPT-2
        "n_positions",
        # MPT
        "max_seq_len",
        # ChatGLM2
        "seq_length",
        # Command-R
        "model_max_length",
        # Others
        "max_sequence_length",
        "max_seq_length",
        "seq_len",
    ]
    for key in possible_keys:
        max_len_key = getattr(hf_config, key, None)
        if max_len_key is not None:
            derived_max_model_len = min(derived_max_model_len, max_len_key)
    if derived_max_model_len == float("inf"):
        if max_model_len is not None:
            # If max_model_len is specified, we use it.
            return max_model_len
        default_max_len = 2048
        logger.warning(
            "The model's config.json does not contain any of the following "
            "keys to determine the original maximum length of the model: "
            f"{possible_keys}. Assuming the model's maximum length is "
            f"{default_max_len}."
        )
        derived_max_model_len = default_max_len
    rope_scaling = getattr(hf_config, "rope_scaling", None)
    if rope_scaling is not None:
        if "type" in rope_scaling:
            rope_type = rope_scaling["type"]
        elif "rope_type" in rope_scaling:
            rope_type = rope_scaling["rope_type"]
        else:
            raise ValueError("rope_scaling must have a 'type' or 'rope_type' key.")
        # The correct one should be "longrope", kept "su" here
        # to be backward compatible
        if rope_type not in ("su", "longrope", "llama3"):
```



```
            assert "factor" in rope_scaling
            scaling_factor = rope_scaling["factor"]
            if rope_type == "yarn":
                derived_max_model_len = rope_scaling["original_max_position_embeddings"]
            derived_max_model_len *= scaling_factor
    if max_model_len is None:
        max_model_len = derived_max_model_len
    elif max_model_len > derived_max_model_len:
        raise ValueError(
            f"User-specified max_model_len ({max_model_len}) is greater than "
            f"the derived max_model_len ({max_len_key}={derived_max_model_len}"
            " in model's config.json). This may lead to incorrect model "
            "outputs or CUDA errors. Make sure the value is correct and "
            "within the model context size."
        )
    return int(max_model_len)
```

f. ./vllm_npu/__init__.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
import torch
import torch_npu
from torch_npu.contrib import transfer_to_npu
from vllm_npu.npu_adaptor import (
    BlockDiagonalCausalMask,
    LowerTriangularMaskWithTensorBias,
    cache_ops,
    context_attention_fwd,
    cuda_utils,
    ops,
)
import vllm_npu.core
import vllm_npu.engine
import vllm_npu.model_executor
import vllm_npu.worker
from vllm_npu.config import DeviceConfig, _get_and_verify_max_len
from vllm_npu.utils import get_ip
from vllm import config, utils
utils.get_ip = get_ip
config.DeviceConfig = DeviceConfig
config._get_and_verify_max_len = _get_and_verify_max_len
__version__ = "0.3.3"
```

g. ./vllm_npu/npu_adapter.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
import importlib
import sys
def replace_modules(old_module_name, new_module_name):
    if old_module_name in sys.modules:
        del sys.modules[old_module_name]
    sys.modules[old_module_name] = importlib.import_module(new_module_name)
_default_ops = (
    'xformers',
    'xformers.ops',
    'vllm._C',
    'xformers.ops.fmha.attn_bias',
    'vllm.model_executor.layers.triton_kernel.prefix_prefill'
)
for _ops in _default_ops:
    replace_modules(_ops, 'vllm_npu')
# dummy class to avoid import error.
class ops:
    pass
class cuda_utils:
    pass
class cache_ops:
    pass
class BlockDiagonalCausalMask:
    pass
class LowerTriangularMaskWithTensorBias:
    pass
```



```
def context_attention_fwd():
    pass
```

h. ./vllm_npu/utils.py

```
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.
import socket
def get_ip() -> str:
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as ipv4_socket:
            ipv4_socket.connect(("localhost", 80))
            return ipv4_socket.getsockname()[0]
    except OSError:
        with socket.socket(socket.AF_INET6, socket.SOCK_DGRAM) as ipv6_socket:
            ipv6_socket.connect(("localhost", 80))
            return ipv6_socket.getsockname()[0]
```

7.2.3.2 vLLM 0.4.2 版本参考适配代码

适配代码仓的目录结构如下所示：

```
├── cover
│   ├── requirements-ascend.txt
│   ├── setup.py
│   └── vllm
│       └── __init__.py
├── examples
│   ├── start_server.sh
│   ├── test_offline.py
│   └── test_offline.sh
├── install.sh
│   └──vllm_npu
├── requirements.txt
├── setup.py
│   └── vllm_npu
├── __init__.py
├── attention
│   ├── __init__.py
│   ├── backends.py
│   └── selector.py
├── config.py
├── core
│   └── __init__.py
├── engine
│   ├── __init__.py
│   ├── ascend_engine.py
│   └── async_ascend_engine.py
├── executor
│   ├── __init__.py
│   ├── ascend_executor.py
│   ├── ascend_ray_executor.py
│   └── ray_utils.py
├── model_executor
│   ├── __init__.py
│   ├── ascend_model_loader.py
│   ├── layers
│   │   ├── __init__.py
│   │   └── ascend_sampler.py
│   └── models
│   │   ├── __init__.py
│   │   └── ascend
│   │   │   ├── __init__.py
│   │   │   └── mindie_llm_wrapper.py
├── npu_adaptor.py
├── usage
│   ├── __init__.py
│   └── usage_lib.py
├── utils.py
│   └── worker
├── __init__.py
```

```
├── ascend_model_runner.py
├── ascend_worker.py
└── cache_engine.py
```

其中主要包括如下四个部分：

1. cover文件夹下包含了对vllm框架源码的修改内容。
2. examples文件夹下包含了离线模式和在线模式的使用实例代码。
3. vllm_npu文件夹下包含了补丁仓的源码内容。
4. install.sh为一键式安装脚本，在将所有的代码文件都还原后，即可运行该脚本一键安装昇腾适配版的vllm框架，其中会自动拉取源码安装vllm原生框架并打上适配补丁。

代码仓中各个文件的代码内容：

- cover
  - requirements-ascend.txt

    ```
    cmake >= 3.21
    ninja  # For faster builds.
    psutil
    sentencepiece  # Required for LLaMA tokenizer.
    numpy
    requests
    py-cpuinfo
    transformers >= 4.40.0  # Required for StarCoder2 & Llava, Llama 3.
    tokenizers >= 0.19.1  # Required for Llama 3.
    fastapi
    openai
    uvicorn[standard]
    pydantic >= 2.0  # Required for OpenAI server.
    prometheus_client >= 0.18.0
    prometheus-fastapi-instrumentator >= 7.0.0
    tiktoken == 0.6.0  # Required for DBRX tokenizer
    lm-format-enforcer == 0.10.1
    typing_extensions
    filelock >= 3.10.4 # filelock starts to support `mode` argument from 3.10.4
    ray == 2.9.3
    pynvml == 11.5.0
    outlines == 0.0.34
    ```

  - setup.py

    ```
    import importlib.util
    import io
    import logging
    import os
    import re
    import subprocess
    import sys
    from shutil import which
    from typing import Dict, List
    import torch
    from packaging.version import Version, parse
    from setuptools import Extension, find_packages, setup
    from setuptools.command.build_ext import build_ext
    from torch.utils.cpp_extension import CUDA_HOME, BuildExtension
    def load_module_from_path(module_name, path):
        spec = importlib.util.spec_from_file_location(module_name, path)
        module = importlib.util.module_from_spec(spec)
        sys.modules[module_name] = module
        spec.loader.exec_module(module)
        return module
    ROOT_DIR = os.path.dirname(__file__)
    logger = logging.getLogger(__name__)
    # cannot import envs directly because it depends on vllm,
    #  which is not installed yet
    envs = load_module_from_path('envs', os.path.join(ROOT_DIR, 'vllm', 'envs.py'))
    ```



```
VLLM_TARGET_DEVICE = 'npu'
# vLLM only supports Linux platform
assert sys.platform.startswith(
    "linux"), "vLLM only supports Linux platform (including WSL)."
MAIN_CUDA_VERSION = "12.1"
def is_sccache_available() -> bool:
    return which("sccache") is not None
def is_ccache_available() -> bool:
    return which("ccache") is not None
def is_ninja_available() -> bool:
    return which("ninja") is not None
def remove_prefix(text, prefix):
    if text.startswith(prefix):
        return text[len(prefix):]
    return text
class CMakeExtension(Extension):
    def __init__(self, name: str, cmake_lists_dir: str = '.', **kwa) -> None:
        super().__init__(name, sources=[], **kwa)
        self.cmake_lists_dir = os.path.abspath(cmake_lists_dir)

def _is_cuda() -> bool:
    return VLLM_TARGET_DEVICE == "cuda" \
            and torch.version.cuda is not None \
            and not _is_neuron()
def _is_hip() -> bool:
    return (VLLM_TARGET_DEVICE == "cuda"
            or VLLM_TARGET_DEVICE == "rocm") and torch.version.hip is not None
def _is_neuron() -> bool:
    torch_neuronx_installed = True
    try:
        subprocess.run(["neuron-ls"], capture_output=True, check=True)
    except (FileNotFoundError, PermissionError, subprocess.CalledProcessError):
        torch_neuronx_installed = False
    return torch_neuronx_installed or envs.VLLM_BUILD_WITH_NEURON
def _is_cpu() -> bool:
    return VLLM_TARGET_DEVICE == "cpu"
def _is_npu() -> bool:
    return VLLM_TARGET_DEVICE == "npu"
def _install_punica() -> bool:
    return envs.VLLM_INSTALL_PUNICA_KERNELS
def get_hipcc_rocm_version():
    # Run the hipcc --version command
    result = subprocess.run(['hipcc', '--version'],
                            stdout=subprocess.PIPE,
                            stderr=subprocess.STDOUT,
                            text=True)
    # Check if the command was executed successfully
    if result.returncode != 0:
        print("Error running 'hipcc --version'")
        return None
    # Extract the version using a regular expression
    match = re.search(r'HIP version: (\S+)', result.stdout)
    if match:
        # Return the version string
        return match.group(1)
    else:
        print("Could not find HIP version in the output")
        return None
def get_neuronxcc_version():
    import sysconfig
    site_dir = sysconfig.get_paths()["purelib"]
    version_file = os.path.join(site_dir, "neuronxcc", "version",
                                "__init__.py")
    # Check if the command was executed successfully
    with open(version_file, "rt") as fp:
        content = fp.read()
    # Extract the version using a regular expression
    match = re.search(r"__version__ = '(\S+)'", content)
    if match:
```



```
        # Return the version string
        return match.group(1)
    else:
        raise RuntimeError("Could not find HIP version in the output")
def get_nvcc_cuda_version() -> Version:
    """Get the CUDA version from nvcc.
    Adapted from https://github.com/NVIDIA/apex/blob/
8b7a1ff183741dd8f9b87e7bafd04cfde99cea28/setup.py
    """
    assert CUDA_HOME is not None, "CUDA_HOME is not set"
    nvcc_output = subprocess.check_output([CUDA_HOME + "/bin/nvcc", "-V"],
                                          universal_newlines=True)
    output = nvcc_output.split()
    release_idx = output.index("release") + 1
    nvcc_cuda_version = parse(output[release_idx].split(",")[0])
    return nvcc_cuda_version
def get_path(*filepath) -> str:
    return os.path.join(ROOT_DIR, *filepath)
def find_version(filepath: str) -> str:
    """Extract version information from the given filepath.
    Adapted from https://github.com/ray-project/ray/blob/
0b190ee1160eeca9796bc091e07eaebf4c85b511/python/setup.py
    """
    with open(filepath) as fp:
        version_match = re.search(r"^__version__ = ['\"]([^'\"]*)['\"]",
                                  fp.read(), re.M)
        if version_match:
            return version_match.group(1)
        raise RuntimeError("Unable to find version string.")
def get_vllm_version() -> str:
    version = find_version(get_path("vllm", "__init__.py"))
    # return version
    if _is_cuda():
        cuda_version = str(get_nvcc_cuda_version())
        if cuda_version != MAIN_CUDA_VERSION:
            cuda_version_str = cuda_version.replace(".", "")[:3]
            version += f"+cu{cuda_version_str}"
    elif _is_hip():
        # Get the HIP version
        hipcc_version = get_hipcc_rocm_version()
        if hipcc_version != MAIN_CUDA_VERSION:
            rocm_version_str = hipcc_version.replace(".", "")[:3]
            version += f"+rocm{rocm_version_str}"
    elif _is_neuron():
        # Get the Neuron version
        neuron_version = str(get_neuronxcc_version())
        if neuron_version != MAIN_CUDA_VERSION:
            neuron_version_str = neuron_version.replace(".", "")[:3]
            version += f"+neuron{neuron_version_str}"
    elif _is_npu():
        version += "+npu"
    elif _is_cpu():
        version += "+cpu"
    else:
        raise RuntimeError("Unknown runtime environment")
    return version
def read_readme() -> str:
    """Read the README file if present."""
    p = get_path("README.md")
    if os.path.isfile(p):
        return io.open(get_path("README.md"), "r", encoding="utf-8").read()
    else:
        return ""
def get_requirements() -> List[str]:
    """Get Python package dependencies from requirements.txt."""
    def _read_requirements(filename: str) -> List[str]:
        with open(get_path(filename)) as f:
            requirements = f.read().strip().split("\n")
        resolved_requirements = []
```



```
        for line in requirements:
            if line.startswith("-r "):
                resolved_requirements += _read_requirements(line.split()[1])
            else:
                resolved_requirements.append(line)
        return resolved_requirements
    if _is_cuda():
        requirements = _read_requirements("requirements-cuda.txt")
        cuda_major, cuda_minor = torch.version.cuda.split(".")
        modified_requirements = []
        for req in requirements:
            if "vllm-nccl-cu12" in req:
                req = req.replace("vllm-nccl-cu12",
                                  f"vllm-nccl-cu{cuda_major}")
            elif ("vllm-flash-attn" in req
                  and not (cuda_major == "12" and cuda_minor == "1")):
                # vllm-flash-attn is built only for CUDA 12.1.
                # Skip for other versions.
                continue
            modified_requirements.append(req)
        requirements = modified_requirements
    elif _is_hip():
        requirements = _read_requirements("requirements-rocm.txt")
    elif _is_neuron():
        requirements = _read_requirements("requirements-neuron.txt")
    elif _is_npu():
        requirements = _read_requirements("requirements-ascend.txt")
    elif _is_cpu():
        requirements = _read_requirements("requirements-cpu.txt")
    else:
        raise ValueError(
            "Unsupported platform, please use CUDA, ROCm, Neuron, NPU or CPU.")
    return requirements
ext_modules = []
if _is_cuda():
    ext_modules.append(CMakeExtension(name="vllm._moe_C"))
"""
if not _is_neuron():
    ext_modules.append(CMakeExtension(name="vllm._C"))
    if _install_punica():
        ext_modules.append(CMakeExtension(name="vllm._punica_C"))
"""
package_data = {
    "vllm": ["py.typed", "model_executor/layers/fused_moe/configs/*.json"]
}
if envs.VLLM_USE_PRECOMPILED:
    ext_modules = []
    package_data["vllm"].append("*.so")
setup(
    name="vllm",
    version=get_vllm_version(),
    author="vLLM Team",
    license="Apache 2.0",
    description=("A high-throughput and memory-efficient inference and "
                 "serving engine for LLMs"),
    long_description=read_readme(),
    long_description_content_type="text/markdown",
    url="https://github.com/vllm-project/vllm",
    project_urls={
        "Homepage": "https://github.com/vllm-project/vllm",
        "Documentation": "https://vllm.readthedocs.io/en/latest/",
    },
    classifiers=[
        "Programming Language :: Python :: 3.8",
        "Programming Language :: Python :: 3.9",
        "Programming Language :: Python :: 3.10",
        "Programming Language :: Python :: 3.11",
        "License :: OSI Approved :: Apache Software License",
        "Topic :: Scientific/Engineering :: Artificial Intelligence",
```



```
    ],
    packages=find_packages(exclude=("benchmarks", "csrc", "docs", "examples",
                                    "tests*")),
    python_requires=">=3.8",
    install_requires=get_requirements(),
    ext_modules=ext_modules,
    extras_require={
        "tensorizer": ["tensorizer==2.9.0"],
    },
    cmdclass={"build_ext": BuildExtension} if not _is_neuron() else {},
    package_data=package_data,
)
```

- vllm
  - __init__.py

```
"""vLLM: a high-throughput and memory-efficient inference engine for LLMs"""
import vllm_npu
from vllm.engine.arg_utils import AsyncEngineArgs, EngineArgs
from vllm.engine.async_llm_engine import AsyncLLMEngine
from vllm.engine.llm_engine import LLMEngine
from vllm.entrypoints.llm import LLM
from vllm.executor.ray_utils import initialize_ray_cluster
from vllm.model_executor.models import ModelRegistry
from vllm.outputs import CompletionOutput, RequestOutput
from vllm.sampling_params import SamplingParams
__version__ = "0.4.2"
__all__ = [
    "LLM",
    "ModelRegistry",
    "SamplingParams",
    "RequestOutput",
    "CompletionOutput",
    "LLMEngine",
    "EngineArgs",
    "AsyncLLMEngine",
    "AsyncEngineArgs",
    "initialize_ray_cluster",
]
```

- vllm_npu
  - requirements.txt

```
numpy
decorator
attrs
psutil
absl-py
cloudpickle
scipy
tornado
transformers
accelerate
pandas
```

  - setup.py

```
import io
import os
import re
from typing import List
import setuptools
ROOT_DIR = os.path.dirname(__file__)
def get_path(*filepath) -> str:
    return os.path.join(ROOT_DIR, *filepath)
def find_version(filepath: str):
    """Extract version information from the given filepath.
    """
    with open(filepath) as fp:
        version_match = re.search(r"^__version__ = ['\"]([^'\"]*)['\"]",
                                  fp.read(), re.M)
```

```
        if version_match:
            return version_match.group(1)
        raise RuntimeError("Unable to find version string.")
def read_readme() -> str:
    """Read the README file."""
    return io.open(get_path("README.md"), "r", encoding="utf-8").read()
def get_requirements() -> List[str]:
    """Get Python package dependencies from requirements.txt."""
    with open(get_path("requirements.txt")) as f:
        requirements = f.read().strip().split("\n")
    return requirements
setuptools.setup(
    name="vllm_npu",
    version=find_version(get_path("vllm_npu", "__init__.py")),
    author="Huawei",
    license="Apache 2.0",
    description=("A high-throughput and memory-efficient inference and "
                 "serving engine for LLMs"),
    long_description=read_readme(),
    long_description_content_type="text/markdown",
    url="",
    project_urls={
        "Homepage": "",
        "Documentation": "",
    },
    classifiers=[
        "Programming Language :: Python :: 3.8",
        "Programming Language :: Python :: 3.9",
        "Programming Language :: Python :: 3.10",
        "Programming Language :: Python :: 3.11",
        "Topic :: Scientific/Engineering :: Artificial Intelligence",
    ],
    packages=setuptools.find_packages(exclude=("benchmarks", "examples", "tests")),
    python_requires=">=3.8",
    install_requires=get_requirements(),
)
```

- vllm_npu
  - attention
    - __init__.py

      ```
      from vllm_npu.attention.selector import get_attn_backend
      import vllm.attention.selector as selector
      import vllm.worker.model_runner as mr
      import vllm.worker.cache_engine as ce
      selector.get_attn_backend = get_attn_backend
      mr.get_attn_backend = get_attn_backend
      ce.get_attn_backend = get_attn_backend
      ```

    - backends.py

      ```
      # Part of codes in this file was copied from project [vLLM Team][vllm]
      from dataclasses import dataclass
      from typing import List, Optional, Tuple
      from vllm.attention.backends.abstract import (AttentionBackend,
                                                    AttentionMetadataPerStage)
      import torch
      def get_kv_cache_shape(
              num_blocks: int,
              block_size: int,
              num_kv_heads: int,
              head_size: int,
          ) -> Tuple[int, ...]:
              return (2, num_blocks, block_size, num_kv_heads, head_size)
      class AscendAttentionBackend(AttentionBackend):
          @staticmethod
          def get_name() -> str:
              return "ascend-attn-backend"
          @staticmethod
          def get_impl_cls():
              return None
      ```



```
    @staticmethod
    def make_metadata(*args, **kwargs) -> "AttentionMetadata":
        return AttentionMetadata(*args, **kwargs)
    @staticmethod
    def get_kv_cache_shape(
        num_blocks: int,
        block_size: int,
        num_kv_heads: int,
        head_size: int,
    ) -> Tuple[int, ...]:
        return get_kv_cache_shape(num_blocks, block_size, num_kv_heads, head_size)
    @staticmethod
    def swap_blocks(
        src_kv_cache: torch.Tensor,
        dst_kv_cache: torch.Tensor,
        src_to_dst: torch.Tensor,
    ) -> None:
        pass
    @staticmethod
    def copy_blocks(
        kv_caches: List[torch.Tensor],
        src_to_dists: torch.Tensor,
    ) -> None:
        pass
@dataclass
class AttentionMetadata(AttentionMetadataPerStage):
    """Metadata for AscendAttentionBackend.
    """
    # Currently, input sequences can only contain all prompts
    # or all decoding. True if all sequences are prompts.
    is_prompt: bool
    # (batch_size,). The sequence length per sequence. Sequence length means
    # the computed tokens + new tokens None if it is a decoding.
    seq_lens: Optional[List[int]]
    # seq_lens stored as a tensor.
    seq_lens_tensor: Optional[torch.Tensor]
    # Maximum query length in the batch.
    max_query_len: Optional[int]
    # Maximum sequence length in the batch.
    max_seq_len: Optional[int]
    # (batch_size + 1,). The cumulative subquery lengths of the sequences in
    # the batch, used to index into subquery. E.g., if the subquery length
    # is [4, 6], it is [0, 4, 10].
    subquery_start_loc: Optional[torch.Tensor]
    # (batch_size + 1,). The cumulative sequence lengths of the sequences in
    # the batch, used to index into sequence. E.g., if the sequence length is
    # [4, 6], it is [0, 4, 10].
    seq_start_loc: Optional[torch.Tensor]
    # (batch_size,) A tensor of context lengths (tokens that are computed
    # so far).
    context_lens_tensor: Optional[torch.Tensor]
    block_tables: Optional[torch.Tensor]
    # Whether or not if cuda graph is enabled.
    use_cuda_graph: bool
```

- selector.py

```
# Part of codes in this file was copied from project [vLLM Team][vllm]
from functools import lru_cache
from typing import Type
import torch
from vllm.attention.backends.abstract import AttentionBackend
from vllm.logger import init_logger
logger = init_logger(__name__)
@lru_cache(maxsize=None)
def get_attn_backend(dtype: torch.dtype) -> Type[AttentionBackend]:
    logger.info("Using Ascend backend.")
    from vllm_npu.attention.backends import AscendAttentionBackend
    return AscendAttentionBackend
```


- core
  - __init__.py: 空白文件
- engine
  - __init__.py

```
from vllm.engine.llm_engine import LLMEngine
from vllm.engine.async_llm_engine import AsyncLLMEngine
from vllm_npu.engine.ascend_engine import from_engine_args
from vllm_npu.engine.async_ascend_engine import from_engine_args_async
LLMEngine.from_engine_args = from_engine_args
AsyncLLMEngine.from_engine_args = from_engine_args_async
```

  - ascend_engine.py

```
from vllm.engine.arg_utils import EngineArgs
from vllm.usage.usage_lib import UsageContext
from vllm_npu.executor.ray_utils import initialize_ray_cluster
@classmethod
def from_engine_args(
    cls,
    engine_args: EngineArgs,
    usage_context: UsageContext = UsageContext.ENGINE_CONTEXT,
) -> "LLMEngine":
    """Creates an LLM engine from the engine arguments."""
    # Create the engine configs.
    engine_config = engine_args.create_engine_config()
    # Initialize the cluster and specify the executor class.
    if engine_config.device_config.device_type == "neuron":
        from vllm.executor.neuron_executor import NeuronExecutor
        executor_class = NeuronExecutor
    elif engine_config.device_config.device_type == "cpu":
        from vllm.executor.cpu_executor import CPUExecutor
        executor_class = CPUExecutor
    elif engine_config.device_config.device_type == "npu":
        if engine_config.parallel_config.worker_use_ray:
            initialize_ray_cluster(engine_config.parallel_config)
            from vllm_npu.executor.ascend_ray_executor import RayAscendExecutor
            executor_class = RayAscendExecutor
        else:
            from vllm_npu.executor.ascend_executor import AscendExecutor
            executor_class = AscendExecutor
    elif engine_config.parallel_config.worker_use_ray:
        initialize_ray_cluster(engine_config.parallel_config)
        from vllm.executor.ray_gpu_executor import RayGPUExecutor
        executor_class = RayGPUExecutor
    else:
        if engine_config.parallel_config.world_size != 1:
            raise ValueError("Ray is required if parallel_config.world_size > 1.")
        from vllm.executor.gpu_executor import GPUExecutor
        executor_class = GPUExecutor
    # Create the LLM engine.
    engine = cls(
        **engine_config.to_dict(),
        executor_class=executor_class,
        log_stats=not engine_args.disable_log_stats,
        usage_context=usage_context,
    )
    return engine
```

  - async_ascend_engine.py

```
from vllm.engine.arg_utils import AsyncEngineArgs
from vllm.usage.usage_lib import UsageContext
from vllm_npu.executor.ray_utils import initialize_ray_cluster
@classmethod
def from_engine_args_async(
    cls,
    engine_args: AsyncEngineArgs,
    start_engine_loop: bool = True,
    usage_context: UsageContext = UsageContext.ENGINE_CONTEXT,
```



```
) -> "AsyncLLMEngine":
    """Creates an async LLM engine from the engine arguments."""
    # Create the engine configs.
    engine_config = engine_args.create_engine_config()
    if engine_config.device_config.device_type == "neuron":
        from vllm.executor.neuron_executor import NeuronExecutorAsync
        executor_class = NeuronExecutorAsync
    elif engine_config.device_config.device_type == "cpu":
        if engine_config.parallel_config.worker_use_ray:
            raise RuntimeError("Ray is not supported with the CPU backend.")
        from vllm.executor.cpu_executor import CPUExecutorAsync
        executor_class = CPUExecutorAsync
    elif engine_config.device_config.device_type == "npu":
        if engine_config.parallel_config.worker_use_ray:
            initialize_ray_cluster(engine_config.parallel_config)
            from vllm_npu.executor.ascend_ray_executor import
RayAscendExecutorAsync
            executor_class = RayAscendExecutorAsync
        else:
            from vllm_npu.executor.ascend_executor import AscendExecutorAsync
            executor_class = AscendExecutorAsync
    elif engine_config.parallel_config.worker_use_ray:
        initialize_ray_cluster(engine_config.parallel_config)
        from vllm.executor.ray_gpu_executor import RayGPUExecutorAsync
        executor_class = RayGPUExecutorAsync
    else:
        if engine_config.parallel_config.world_size != 1:
            raise RuntimeError("Ray is required if parallel_config.world_size > 1.")
        from vllm.executor.gpu_executor import GPUExecutorAsync
        executor_class = GPUExecutorAsync
    # Create the async LLM engine.
    engine = cls(
        engine_config.parallel_config.worker_use_ray,
        engine_args.engine_use_ray,
        **engine_config.to_dict(),
        executor_class=executor_class,
        log_requests=not engine_args.disable_log_requests,
        log_stats=not engine_args.disable_log_stats,
        max_log_len=engine_args.max_log_len,
        start_engine_loop=start_engine_loop,
        usage_context=usage_context,
    )
    return engine
```

- executor
  - __init__.py

```
from vllm_npu.executor.ray_utils import initialize_ray_cluster
from vllm.executor import ray_utils
ray_utils.initialize_ray_cluster = initialize_ray_cluster
```

  - ascend_executor.py

```
# Part of codes in this file was copied from project [vLLM Team][vllm]
from typing import List, Set, Tuple
from vllm.executor.executor_base import ExecutorAsyncBase, ExecutorBase
from vllm.logger import init_logger
from vllm.lora.request import LoRARequest
from vllm.sequence import ExecuteModelRequest, SamplerOutput
from vllm.utils import (get_distributed_init_method, get_ip, get_open_port,
                        make_async)
from vllm_npu.worker.ascend_worker import AscendWorker
logger = init_logger(__name__)
class AscendExecutor(ExecutorBase):
    def _init_executor(self) -> None:
        assert (not self.speculative_config
            ), "Speculative decoding is not yet supported for Ascend backend."
        # Instantiate the worker and load the model to the device.
        self._init_worker()
    def _init_worker(self):
        distributed_init_method = get_distributed_init_method(
```



```
                get_ip(), get_open_port())
        self.driver_worker = AscendWorker(
            self.model_config,
            self.parallel_config,
            self.scheduler_config,
            self.device_config,
            self.cache_config,
            self.load_config,
            local_rank=0,
            rank=0,
            distributed_init_method=distributed_init_method,
            lora_config=self.lora_config,
            is_driver_worker=True,
        )
        self.driver_worker.init_device()
        self.driver_worker.load_model()
    def determine_num_available_blocks(self) -> Tuple[int, int]:
        """Determine the number of available KV blocks by invoking the
        underlying worker.
        """
        return self.driver_worker.determine_num_available_blocks()
    def initialize_cache(self, num_gpu_blocks: int,
                         num_cpu_blocks: int) -> None:
        """Initialize the KV cache by invoking the underlying worker.
        """
        self.driver_worker.initialize_cache(num_gpu_blocks, num_cpu_blocks)
    def execute_model(
            self,
            execute_model_req: ExecuteModelRequest) -> List[SamplerOutput]:
        output = self.driver_worker.execute_model(execute_model_req)
        return output
    def add_lora(self, lora_request: LoRARequest) -> bool:
        return self.driver_worker.add_lora(lora_request)
    def remove_lora(self, lora_id: int) -> bool:
        return self.driver_worker.remove_lora(lora_id)
    def list_loras(self) -> Set[int]:
        return self.driver_worker.list_loras()
    def check_health(self) -> None:
        # NeuronExecutor will always be healthy as long as
        # it's running.
        return
class AscendExecutorAsync(AscendExecutor, ExecutorAsyncBase):
    async def execute_model_async(
        self,
        execute_model_req: ExecuteModelRequest,
    ) -> List[SamplerOutput]:
        output = await make_async(
            self.driver_worker.execute_model
        )(execute_model_req=execute_model_req,)
        return output
    # async def check_health_async(self) -> None:
    #     # AscendExecutor will always be healthy as long as
    #     # it's running.
    #     return
```

- ascend_ray_executor.py

```
import asyncio
import os
import pickle
from collections import defaultdict
from itertools import islice, repeat
from typing import TYPE_CHECKING, Any, Dict, List, Optional, Tuple
import vllm.envs as envs
from vllm.executor.distributed_gpu_executor import (  # yapf: disable
    DistributedGPUExecutor, DistributedGPUExecutorAsync)
from vllm.executor.ray_utils import RayWorkerWrapper, ray
from vllm.logger import init_logger
from vllm.sequence import ExecuteModelRequest, SamplerOutput
from vllm.utils import (get_distributed_init_method, get_ip, get_open_port,
                        get_vllm_instance_id, make_async)
```



```
if ray is not None:
    from ray.util.scheduling_strategies import PlacementGroupSchedulingStrategy
if TYPE_CHECKING:
    from ray.util.placement_group import PlacementGroup
logger = init_logger(__name__)
USE_RAY_COMPILED_DAG = envs.VLLM_USE_RAY_COMPILED_DAG
class RayAscendExecutor(DistributedGPUExecutor):
    def _init_executor(self) -> None:
        assert (not self.speculative_config
                ), "Speculative decoding not yet supported for RayNPU backend."
        assert self.parallel_config.worker_use_ray
        placement_group = self.parallel_config.placement_group
        # Disable Ray usage stats collection.
        ray_usage = os.environ.get("RAY_USAGE_STATS_ENABLED", "0")
        if ray_usage != "1":
            os.environ["RAY_USAGE_STATS_ENABLED"] = "0"
        # Create the parallel GPU workers.
        self._init_workers_ray(placement_group)
        self.forward_dag = None
        if USE_RAY_COMPILED_DAG:
            self.forward_dag = self._compiled_ray_dag()
    def _init_workers_ray(self, placement_group: "PlacementGroup",
                          **ray_remote_kwargs):
        if self.parallel_config.tensor_parallel_size == 1:
            # For single GPU case, we use a ray worker with constrained memory.
            num_gpus = self.cache_config.gpu_memory_utilization
        else:
            # Otherwise, the ray workers are allocated with a full GPU.
            num_gpus = 1
        # The driver dummy worker does not actually use any resources.
        # It holds the resource for the driver worker.
        self.driver_dummy_worker: Optional[RayWorkerWrapper] = None
        # The remaining workers are the actual ray actors.
        self.workers: List[RayWorkerWrapper] = []
        if self.parallel_config.ray_workers_use_nsight:
            ray_remote_kwargs = self._configure_ray_workers_use_nsight(
                ray_remote_kwargs)
        # Create the workers.
        driver_ip = get_ip()
        for bundle_id, bundle in enumerate(placement_group.bundle_specs):
            if not bundle.get("GPU", 0):
                continue
            scheduling_strategy = PlacementGroupSchedulingStrategy(
                placement_group=placement_group,
                placement_group_capture_child_tasks=True,
                placement_group_bundle_index=bundle_id,
            )
            worker = ray.remote(
                num_cpus=0,
                num_gpus=num_gpus,
                scheduling_strategy=scheduling_strategy,
                **ray_remote_kwargs,
            )(RayWorkerWrapper).remote(
                worker_module_name="vllm_npu.worker.ascend_worker",
                worker_class_name="AscendWorker",
                trust_remote_code=self.model_config.trust_remote_code,
            )
            worker_ip = ray.get(worker.get_node_ip.remote())
            if worker_ip == driver_ip and self.driver_dummy_worker is None:
                # If the worker is on the same node as the driver, we use it
                # as the resource holder for the driver process.
                self.driver_dummy_worker = worker
                self.driver_worker = RayWorkerWrapper(
                    worker_module_name="vllm_npu.worker.ascend_worker",
                    worker_class_name="AscendWorker",
                    trust_remote_code=self.model_config.trust_remote_code,
                )
            else:
                # Else, added to the list of workers.
```



```
            self.workers.append(worker)
    if self.driver_dummy_worker is None:
        raise ValueError(
            "Ray does not allocate any GPUs on the driver node. Consider "
            "adjusting the Ray placement group or running the driver on a "
            "GPU node.")
    # Get the set of GPU IDs used on each node.
    worker_node_and_gpu_ids = self._run_workers("get_node_and_gpu_ids",
                                                use_dummy_driver=True)
    node_workers = defaultdict(list)
    node_gpus = defaultdict(list)
    for i, (node_id, gpu_ids) in enumerate(worker_node_and_gpu_ids):
        node_workers[node_id].append(i)
        node_gpus[node_id].extend(gpu_ids)
    for node_id, gpu_ids in node_gpus.items():
        node_gpus[node_id] = sorted(gpu_ids)
    VLLM_INSTANCE_ID = get_vllm_instance_id()
    # Set environment variables for the driver and workers.
    all_args_to_update_environment_variables = [({
        "CUDA_VISIBLE_DEVICES":
        ",".join(map(str, node_gpus[node_id])),
        "VLLM_INSTANCE_ID":
        VLLM_INSTANCE_ID,
        "VLLM_TRACE_FUNCTION":
        str(envs.VLLM_TRACE_FUNCTION),
    }, ) for (node_id, _) in worker_node_and_gpu_ids]
    self._run_workers("update_environment_variables",
                      all_args=all_args_to_update_environment_variables)
    distributed_init_method = get_distributed_init_method(
        driver_ip, get_open_port())
    # Initialize the actual workers inside worker wrapper.
    init_worker_all_kwargs = [
        self._get_worker_kwargs(
            local_rank=node_workers[node_id].index(rank),
            rank=rank,
            distributed_init_method=distributed_init_method,
        ) for rank, (node_id, _) in enumerate(worker_node_and_gpu_ids)
    ]
    self._run_workers("init_worker", all_kwargs=init_worker_all_kwargs)
    self._run_workers("init_device")
    self._run_workers("load_model",
                      max_concurrent_workers=self.parallel_config.
                      max_parallel_loading_workers)
def execute_model(
        self,
        execute_model_req: ExecuteModelRequest) -> List[SamplerOutput]:
    all_outputs = self._run_workers(
        "execute_model",
        driver_kwargs={"execute_model_req": execute_model_req},
        use_ray_compiled_dag=USE_RAY_COMPILED_DAG)
    # Only the driver worker returns the sampling results.
    return all_outputs[0]
def _run_workers(
    self,
    method: str,
    *args,
    driver_args: Optional[Tuple[Any, ...]] = None,
    driver_kwargs: Optional[Dict[str, Any]] = None,
    all_args: Optional[List[Tuple[Any, ...]]] = None,
    all_kwargs: Optional[List[Dict[str, Any]]] = None,
    use_dummy_driver: bool = False,
    max_concurrent_workers: Optional[int] = None,
    use_ray_compiled_dag: bool = False,
    **kwargs,
) -> Any:
    """Runs the given method on all workers. Can be used in the following
    ways:
    - args/kwargs: All workers share the same args/kwargs
    - args/kwargs and driver_args/driver_kwargs: Driver worker has
```



```
        different args
    - all_args/all_kwargs: args/kwargs for each worker are specified
        individually
    """
    if max_concurrent_workers:
        raise NotImplementedError(
            "max_concurrent_workers is not supported yet.")
    if driver_args is None:
        driver_args = args if all_args is None else all_args[0]
    if driver_kwargs is None:
        driver_kwargs = kwargs if all_kwargs is None else all_kwargs[0]
    count = len(self.workers)
    all_worker_args = repeat(args, count) if all_args is None \
        else islice(all_args, 1, None)
    all_worker_kwargs = repeat(kwargs, count) if all_kwargs is None \
        else islice(all_kwargs, 1, None)
    if use_ray_compiled_dag:
        assert self.forward_dag is not None
        output_channels = self.forward_dag.execute(1)
    else:
        # Start the ray workers first.
        ray_worker_outputs = [
            worker.execute_method.remote(method, *worker_args,
                                         **worker_kwargs)
            for (worker, worker_args, worker_kwargs
                 ) in zip(self.workers, all_worker_args, all_worker_kwargs)
        ]
    # Start the driver worker after all the ray workers.
    if not use_dummy_driver:
        driver_worker_output = self.driver_worker.execute_method(
            method, *driver_args, **driver_kwargs)
    else:
        assert self.driver_dummy_worker is not None
        driver_worker_output = ray.get(
            self.driver_dummy_worker.execute_method.remote(
                method, *driver_args, **driver_kwargs))
    # Get the results of the ray workers.
    if self.workers:
        if use_ray_compiled_dag:
            try:
                ray_worker_outputs = [
                    pickle.loads(chan.begin_read())
                    for chan in output_channels
                ]
            finally:
                # Has to call end_read in order to reuse the DAG.
                for chan in output_channels:
                    chan.end_read()
        else:
            ray_worker_outputs = ray.get(ray_worker_outputs)
    return [driver_worker_output] + ray_worker_outputs
def _compiled_ray_dag(self):
    import pkg_resources
    required_version = "2.9"
    current_version = pkg_resources.get_distribution("ray").version
    if current_version < required_version:
        raise ValueError(f"Ray version {required_version} or greater is "
                         f"required, but found {current_version}")
    from ray.dag import InputNode, MultiOutputNode
    assert self.parallel_config.worker_use_ray
    with InputNode() as input_data:
        forward_dag = MultiOutputNode([
            worker.execute_model_compiled_dag_remote.
            bind(  # type: ignore[attr-defined]
                input_data) for worker in self.workers
        ])
    return forward_dag.experimental_compile()
def check_health(self) -> None:
    """Raises an error if engine is unhealthy."""
```



```
        self._check_if_any_actor_is_dead()
    def _check_if_any_actor_is_dead(self):
        if not self.workers:
            return
        dead_actors = []
        for actor in self.workers:
            actor_state = ray.state.actors(actor._ray_actor_id.hex())  # pylint:
disable=protected-access
            if actor_state["State"] == "DEAD":
                dead_actors.append(actor)
        if dead_actors:
            raise RuntimeError("At least one Worker is dead. "
                               f"Dead Workers: {dead_actors}. ")
class RayAscendExecutorAsync(RayAscendExecutor, DistributedGPUExecutorAsync):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.driver_executor = make_async(self.driver_worker.execute_method)
    async def _run_workers_async(
        self,
        method: str,
        *args,
        driver_args: Optional[Tuple[Any, ...]] = None,
        driver_kwargs: Optional[Dict[str, Any]] = None,
        **kwargs,
    ) -> Any:
        """Runs the given method on all workers."""
        coros = []
        if driver_args is None:
            driver_args = args
        if driver_kwargs is None:
            driver_kwargs = kwargs
        coros.append(
            self.driver_executor(method, *driver_args, **driver_kwargs))
        # Run the ray workers asynchronously.
        for worker in self.workers:
            coros.append(worker.execute_method.remote(method, *args, **kwargs))
        all_outputs = await asyncio.gather(*coros)
        return all_outputs
```

- ray_utils.py

```
from typing import Optional, Tuple, TYPE_CHECKING
from vllm.config import ParallelConfig
from vllm.utils import is_hip
from vllm.logger import init_logger
logger = init_logger(__name__)
try:
    import ray
except ImportError as e:
    logger.warning(f"Failed to import Ray with {e!r}. "
                   "For distributed inference, please install Ray with "
                   "`pip install ray`.")
    ray = None
if TYPE_CHECKING:
    from ray.util.placement_group import PlacementGroup
def initialize_ray_cluster(
    parallel_config: ParallelConfig,
    ray_address: Optional[str] = None,
):
    """Initialize the distributed cluster with Ray.
    it will connect to the Ray cluster and create a placement group
    for the workers, which includes the specification of the resources
    for each distributed worker.
    Args:
        parallel_config: The configurations for parallel execution.
        ray_address: The address of the Ray cluster. If None, uses
            the default Ray cluster address.
    """
    if ray is None:
        raise ImportError(
            "Ray is not installed. Please install Ray to use multi-node "
```



```
        "serving.")
# Connect to a ray cluster.
if is_hip():
    ray.init(address=ray_address,
             ignore_reinit_error=True,
             num_gpus=parallel_config.world_size)
else:
    """start adapt"""
    # without setting num_gpus, the function will try to detect num of
    # GPUs, but in ascend environment it may fail to detect gpus, thus
    # needed to be manually setted.
    ray.init(address=ray_address, ignore_reinit_error=True,
             num_gpus=parallel_config.world_size)
    """end adapt"""
if parallel_config.placement_group:
    # Placement group is already set.
    return
# Create placement group for worker processes
current_placement_group = ray.util.get_current_placement_group()
if current_placement_group:
    # We are in a placement group
    bundles = current_placement_group.bundle_specs
    # Verify that we can use the placement group.
    gpu_bundles = 0
    for bundle in bundles:
        bundle_gpus = bundle.get("GPU", 0)
        if bundle_gpus > 1:
            raise ValueError(
                "Placement group bundle cannot have more than 1 GPU.")
        if bundle_gpus:
            gpu_bundles += 1
    if parallel_config.world_size > gpu_bundles:
        raise ValueError(
            "The number of required GPUs exceeds the total number of "
            "available GPUs in the placement group.")
else:
    num_gpus_in_cluster = ray.cluster_resources().get("GPU", 0)
    if parallel_config.world_size > num_gpus_in_cluster:
        raise ValueError(
            "The number of required GPUs exceeds the total number of "
            "available GPUs in the cluster.")
    # Create a new placement group
    placement_group_specs = ([{"GPU": 1}] * parallel_config.world_size)
    current_placement_group = ray.util.placement_group(
        placement_group_specs)
    # Wait until PG is ready - this will block until all
    # requested resources are available, and will timeout
    # if they cannot be provisioned.
    ray.get(current_placement_group.ready(), timeout=1800)
# Set the placement group in the parallel config
parallel_config.placement_group = current_placement_group
```

- model_executor
  - __init__.py

```
import vllm.model_executor.model_loader as vllm_model_loader
import vllm_npu.model_executor.ascend_model_loader as ascend_model_loader
vllm_model_loader.get_architecture_class_name =
ascend_model_loader.get_architecture_class_name
```

  - ascend_model_loader.py

```
# Part of code in this file was copied from project [vLLM Team][vllm] for adapting
usage
import contextlib
import torch
import torch.nn as nn
from vllm.config import DeviceConfig, ModelConfig, LoadConfig
from vllm.model_executor.model_loader.weight_utils import
initialize_dummy_weights
from vllm_npu.model_executor.models.ascend.mindie_llm_wrapper import
```



```
MindIELlmWrapper
def get_architecture_class_name(model_config: ModelConfig) -> str:
    architectures = getattr(model_config.hf_config, "architectures", [])
    if (model_config.quantization is not None
            and model_config.quantization != "fp8"
            and "MixtralForCausalLM" in architectures):
        architectures = ["QuantMixtralForCausalLM"]
    return architectures[0]

@contextlib.contextmanager
def _set_default_torch_dtype(dtype: torch.dtype):
    """Sets the default torch dtype to the given dtype."""
    old_dtype = torch.get_default_dtype()
    torch.set_default_dtype(dtype)
    yield
    torch.set_default_dtype(old_dtype)
def get_model(model_config: ModelConfig, device_config: DeviceConfig,
              load_config: LoadConfig,
              mindie_model_config, **kwargs) -> nn.Module:
    lora_config = kwargs.get("lora_config", None)
    model_class = MindIELlmWrapper
    # Get the (maybe quantized) linear method.
    linear_method = None
    with _set_default_torch_dtype(model_config.dtype):
        # Create a model instance.
        # The weights will be initialized as empty tensors.
        with torch.device(device_config.device):
            if hasattr(model_class, "supported_lora_modules"):
                model = model_class(mindie_model_config, linear_method,
                                    lora_config)
            elif lora_config:
                raise ValueError(
                    f"Model {model_class.__name__} does not support LoRA, "
                    "but LoRA is enabled. Support for this model may "
                    "be added in the future. If this is important to you, "
                    "please open an issue on github.")
            else:
                model = model_class(mindie_model_config, linear_method)
        if load_config.load_format == "dummy":
            initialize_dummy_weights(model)
        else:
            # Load the weights from the cached or downloaded files.
            model.load_weights(model_config.model, load_config.download_dir,
                               load_config.load_format, model_config.revision)
        model = model.npu()
    return model.eval()
```

- layers
- __init__.py：空白文件
- ascend_sampler.py

```
# Part of codes in this file was copied from project [vLLM Team][vllm]
import random
from typing import Dict, List, Optional, Tuple
import torch
import torch.nn as nn
import numpy as np
from vllm.sampling_params import SamplingType
from vllm.sequence import SamplerOutput
from vllm.model_executor.sampling_metadata import SamplingMetadata,
SequenceGroupToSample
from vllm.model_executor.layers.sampler import _get_logprobs,
_build_sampler_output
from mindie_llm.text_generator.utils.sampling_metadata import SamplingData,
SamplingParam
_SAMPLING_EPS = 1e-5
SampleResultType = List[Tuple[List[int], List[int]]]
def _to_tensor(data, dtype=None):
    if dtype:
```

```
        return torch.tensor(data, dtype=dtype, device=torch.device("npu"))
    else:
        return torch.tensor(data, device=torch.device("npu"))
class AscendSampler(nn.Module):
    def __init__(self, mindie_model):
        super().__init__()
        self.mindie_model = mindie_model
        self.include_gpu_probs_tensor = False
    def forward(
        self,
        logits: torch.Tensor,
        sampling_metadata: SamplingMetadata,
    ) -> Optional[SamplerOutput]:
        _, vocab_size = logits.shape
        mindie_sampling_data, mindie_sampling_param =
self.construct_data(sampling_metadata, vocab_size)
        probs = torch.softmax(logits, dim=-1, dtype=torch.float)
        logprobs = torch.log_softmax(logits, dim=-1, dtype=torch.float)
        next_tokens = self.mindie_model.sample(
            logits,
            sampling_data=mindie_sampling_data,
            sampling_param=mindie_sampling_param,
        )

        sample_results, maybe_sampled_tokens_tensor = recover_data(
            sampling_metadata=sampling_metadata,
            sampled_tokens=next_tokens,
            logprobs=logprobs,
            include_gpu_probs_tensor=self.include_gpu_probs_tensor,
        )
        if self.include_gpu_probs_tensor:
            if maybe_sampled_tokens_tensor is None:
                raise RuntimeError("maybe_sampled_tokens_tensor is None")
            on_device_tensors = (probs, logprobs, maybe_sampled_tokens_tensor)
        else:
            on_device_tensors = None
        # Get the logprobs query results.
        prompt_logprobs, sample_logprobs = _get_logprobs(
            logprobs, sampling_metadata, sample_results)
        return _build_sampler_output(sample_results,
                                     sampling_metadata,
                                     prompt_logprobs,
                                     sample_logprobs,
                                     on_device_tensors=on_device_tensors)
    def construct_data(
        self,
        sampling_metadata: SamplingMetadata,
        vocab_size: int,
    ) -> Tuple[SamplingData, SamplingParam]:
        all_input_tokens: List[List[int]] = []
        prompt_tokens: List[List[int]] = []
        output_tokens: List[List[int]] = []
        top_ks: List[int] = []
        temperatures: List[float] = []
        top_ps: List[float] = []
        min_ps: List[float] = []
        presence_penalties: List[float] = []
        frequency_penalties: List[float] = []
        repetition_penalties: List[float] = []
        sampling_seeds: List[int] = []
        sample_indices: List[int] = []
        do_samples: List[bool] = []  # To Do
        do_penalties = False
        do_top_p_top_k = False
        do_min_p = False
        greedy_flag = False

        if sampling_metadata.seq_groups is None:
            raise RuntimeError("sampling_metadata.seq_group is None, no data
```



```
received.")
        for seq_group in sampling_metadata.seq_groups:
            do_samples.append(seq_group.do_sample)
            seq_ids = seq_group.seq_ids
            sampling_params = seq_group.sampling_params
            temperature = sampling_params.temperature
            p = sampling_params.presence_penalty
            f = sampling_params.frequency_penalty
            r = sampling_params.repetition_penalty
            top_p = sampling_params.top_p
            min_p = sampling_params.min_p
            is_greedy = sampling_params.sampling_type == SamplingType.GREEDY
            seed = sampling_params.seed
            if seed is None:
                if is_greedy:
                    seed = 0
                else:
                    lo, hi = torch.iinfo(torch.long).min, torch.iinfo(torch.long).max
                    seed = random.randint(lo, hi)
            if is_greedy:
                greedy_flag = True
            # k should not be greater than the vocab size.
            top_k = min(sampling_params.top_k, vocab_size)
            top_k = vocab_size if top_k == -1 else top_k
            if temperature < _SAMPLING_EPS:
                temperature = 1.0
            if not do_top_p_top_k and (top_p < 1.0 - _SAMPLING_EPS
                                       or top_k != vocab_size):
                do_top_p_top_k = True
            if not do_min_p and min_p > _SAMPLING_EPS:
                do_min_p = True
            if not do_penalties:
                if abs(p) >= _SAMPLING_EPS:
                    do_penalties = True
                elif abs(f) >= _SAMPLING_EPS:
                    do_penalties = True
                elif abs(r - 1.0) >= _SAMPLING_EPS:
                    do_penalties = True
            is_prompt = seq_group.is_prompt
            if (seq_group.is_prompt
                    and sampling_params.prompt_logprobs is not None):
                # For tokens in the prompt that we only need to get
                # their logprobs
                query_len = seq_group.query_len
                if query_len is None:
                    raise RuntimeError("query_len is None")
                prefill_len = len(seq_group.prompt_logprob_indices)
                temperatures += [temperature] * prefill_len
                sampling_seeds += [seed] * prefill_len
                top_ps += [top_p] * prefill_len
                top_ks += [top_k] * prefill_len
                min_ps += [min_p] * prefill_len
                presence_penalties += [0] * prefill_len
                frequency_penalties += [0] * prefill_len
                repetition_penalties += [1] * prefill_len
                prompt_tokens.extend([] for _ in range(prefill_len))
                output_tokens.extend([] for _ in range(prefill_len))
                all_input_tokens.extend([] for _ in range(prefill_len))
            if seq_group.do_sample:
                sample_lens = len(seq_group.sample_indices)
                if sample_lens != len(seq_ids):
                    raise ValueError("sample_lens != len(seq_ids)")
                for seq_id in seq_ids:
                    seq_data = seq_group.seq_data[seq_id]
                    prompt_tokens.append(seq_data.prompt_token_ids)
                    output_tokens.append(seq_data.output_token_ids)
                    all_input_tokens.append(seq_data.prompt_token_ids +
seq_data.output_token_ids)
                temperatures += [temperature] * len(seq_ids)
```



```
                sampling_seeds += [seed] * len(seq_ids)
                top_ps += [top_p] * len(seq_ids)
                top_ks += [top_k] * len(seq_ids)
                min_ps += [min_p] * len(seq_ids)
                presence_penalties += [p] * len(seq_ids)
                frequency_penalties += [f] * len(seq_ids)
                repetition_penalties += [r] * len(seq_ids)
        repetition_penalties = np.array(repetition_penalties, dtype=np.float32)
        frequency_penalties = np.array(frequency_penalties, dtype=np.float32)
        presence_penalties = np.array(presence_penalties, dtype=np.float32)
        temperatures = np.array(temperatures, dtype=np.float32)
        top_ks = np.array(top_ks, dtype=np.int32)
        top_ps = np.array(top_ps, dtype=np.float32)
        sampling_seeds = np.array(sampling_seeds)
        do_samples = np.array(do_samples)
        max_tokens_len = max([len(tokens) for tokens in all_input_tokens], default=0)
        padded_all_input_tokens = [
            tokens + [vocab_size] * (max_tokens_len - len(tokens))
            for tokens in all_input_tokens
        ]
        padded_all_input_tokens = np.array(padded_all_input_tokens, dtype=np.int32)
        output_max_len = max([len(tokens) for tokens in output_tokens], default=0)
        padded_output_tokens = [
            tokens + [vocab_size] * (output_max_len - len(tokens))
            for tokens in output_tokens
        ]
        padded_output_tokens = np.array(padded_output_tokens, dtype=np.int32)
        all_input_ids_tensor = _to_tensor(
            padded_all_input_tokens,
            torch.int32
        ) if padded_all_input_tokens is not None else None
        output_ids_tensor = _to_tensor(
            padded_output_tokens,
            torch.int32
        ) if padded_output_tokens is not None else None
        mindie_sampling_data = SamplingData(
            all_input_ids=all_input_ids_tensor,
            output_ids=output_ids_tensor
        )
        if greedy_flag:
            mindie_sampling_param = None
        else:
            mindie_sampling_param = SamplingParam.from_numpy(
                repetition_penalty=repetition_penalties,
                frequency_penalty=frequency_penalties,
                presence_penalty=presence_penalties,
                temperature=temperatures,
                top_k=top_ks,
                top_p=top_ps,
                seed=sampling_seeds,
                do_sample=do_samples,
                to_tensor=_to_tensor,
            )
        return (mindie_sampling_data, mindie_sampling_param)
def recover_data(
    sampling_metadata: SamplingMetadata,
    sampled_tokens: np.ndarray,
    logprobs: torch.Tensor,
    include_gpu_probs_tensor: bool,
) -> Tuple[SampleResultType, Optional[torch.Tensor]]:
    categorized_seq_group_ids: Dict[SamplingType,
                                    List[int]] = {t: []
                                                  for t in SamplingType}
    categorized_sample_indices = sampling_metadata.categorized_sample_indices
    for i, seq_group in enumerate(sampling_metadata.seq_groups):
        sampling_params = seq_group.sampling_params
        sampling_type = sampling_params.sampling_type
        categorized_seq_group_ids[sampling_type].append(i)
    sample_results_dict: Dict[int, Tuple[List[int], List[int]]] = {}
```



```
    sample_metadata = {}
    # Create output tensor for sampled token ids.
    if include_gpu_probs_tensor:
        sampled_token_ids_tensor = torch.empty(logprobs.shape[0],
                                               1,
                                               dtype=torch.long,
                                               device=logprobs.device)
    else:
        sampled_token_ids_tensor = None
    for sampling_type in SamplingType:
        sample_indices = categorized_sample_indices[sampling_type][:, 0]
        num_tokens = len(sample_indices)
        if num_tokens == 0:
            continue
        seq_group_id = categorized_seq_group_ids[sampling_type]
        seq_groups = [sampling_metadata.seq_groups[i] for i in seq_group_id]
        sample_metadata[sampling_type] = (seq_group_id, seq_groups)
    for sampling_type in SamplingType:
        if sampling_type not in sample_metadata:
            continue
        (seq_group_id, seq_groups) = sample_metadata[sampling_type]
        if sampling_type in (SamplingType.GREEDY, SamplingType.RANDOM,
SamplingType.RANDOM_SEED):
            sample_results = normal_wrap(seq_groups, sampled_tokens)
        elif sampling_type == SamplingType.BEAM:
            sample_results = beam_wrap(seq_groups, sampled_tokens)
        sample_results_dict.update(zip(seq_group_id, sample_results))
    sample_results = [
        sample_results_dict.get(i, ([], []))
        for i in range(len(sampling_metadata.seq_groups))
    ]
    return sample_results, sampled_token_ids_tensor
def normal_wrap(
    selected_seq_groups: List[SequenceGroupToSample],
    samples: np.ndarray,
):
    samples = samples.tolist()
    sample_idx = 0
    results: SampleResultType = []
    for seq_group in selected_seq_groups:
        if not seq_group.do_sample:
            results.append(([], []))
            continue
        seq_ids = seq_group.seq_ids
        num_parent_seqs = len(seq_ids)
        parent_ids = list(range(num_parent_seqs))
        next_token_ids = [samples[sample_idx]]
        results.append((next_token_ids, parent_ids))
        sample_idx += num_parent_seqs
    return results
def beam_wrap(
    selected_seq_groups: List[SequenceGroupToSample],
    samples: np.ndarray,
):
    raise ValueError(f"Unsupported sampling type: beam search")
```

- models
- __init__.py：空白文件
- ascend
- __init__.py：空白文件
- mindie_llm_wrapper.py

```
from typing import List, Optional
import math
import torch
from torch import nn
from vllm.model_executor import SamplingMetadata
from vllm.sequence import SamplerOutput
```



```
from vllm.attention import AttentionMetadata
from mindie_llm.text_generator.adapter.generator_torch import GeneratorTorch
from vllm_npu.worker.cache_engine import KVCache
from vllm_npu.model_executor.layers.ascend_sampler import AscendSampler
class MindIELlmWrapper(nn.Module):
    def __init__(self, mindie_model_config, linear_method=None, lora_config=None):
        super(MindIELlmWrapper, self).__init__()

        self.mindie_model_config = mindie_model_config
        self.rank = mindie_model_config['rank']
        self.local_rank = mindie_model_config['local_rank']
        self.npu_id = self.local_rank
        self.world_size = mindie_model_config['world_size']
        self.mindie_model = None
        self.sampler = None
    def forward(
            self,
            input_ids: torch.Tensor,
            positions: torch.Tensor,
            kv_caches: List[KVCache],
            attn_metadata: AttentionMetadata,
    ) -> torch.Tensor:
        is_prompt = attn_metadata.num_prefill_tokens > 0

        if kv_caches[0][0] is None:
            kv_caches, block_tables, slots = self.create_dummy_kv_cache(attn_metadata,
input_ids)
        else:
            if is_prompt:
                block_tables = torch.tensor([0], dtype=torch.int32, device="npu")
            else:
                block_tables = attn_metadata.decode_metadata.block_tables
            slots = attn_metadata.slot_mapping
        if is_prompt:
            input_lengths =
attn_metadata.prefill_metadata.seq_lens_tensor.to(torch.int32)
            max_seq_len = int(attn_metadata.prefill_metadata.seq_lens_tensor.max())
            lm_head_indices =
(attn_metadata.prefill_metadata.seq_lens_tensor.cumsum(dim=-1) -
1).to(torch.int64)
        else:
            input_lengths = attn_metadata.decode_metadata.seq_lens_tensor
            max_seq_len = attn_metadata.decode_metadata.max_seq_len
            lm_head_indices = None

        logits = self.mindie_model.forward_tensor(input_ids, positions, is_prompt,
kv_caches, block_tables, slots,
                                  input_lengths, max_seq_len, lm_head_indices)
        return logits
    def compute_logits(self, hidden_states: torch.Tensor,
                       sampling_metadata: SamplingMetadata) -> torch.Tensor:
        return hidden_states
    def sample(
        self,
        logits: torch.Tensor,
        sampling_metadata: SamplingMetadata,
    ) -> Optional[SamplerOutput]:
        # hidden_states is logits
        next_tokens = self.sampler(logits, sampling_metadata)
        return next_tokens
    def load_weights(self,
                     model_name_or_path: str,
                     cache_dir: Optional[str] = None,
                     load_format: str = "auto",
                     revision: Optional[str] = None):
        if load_format not in ['auto', 'safetensors', 'pt']:
            raise ValueError('load-format support [safetensors, pt]')
        self.weight_dtype = torch.get_default_dtype()
        torch.set_default_dtype(torch.float32)
```



```
        self.mindie_model = GeneratorTorch(self.mindie_model_config)
        self.sampler = AscendSampler(self.mindie_model)
        torch.set_default_dtype(self.weight_dtype)
    # when warmup, create dummy kvcache, block_tables, slot_mapping
    def create_dummy_kv_cache(self, attn_metadata, input_ids):
        dummy_block_num = 1
        dummy_block_size = 128
        model_runner = self.mindie_model.model_wrapper.model_runner
        kv_cache = [
            (
                torch.empty(
                    (dummy_block_num, dummy_block_size, model_runner.num_kv_heads,
model_runner.head_size),
                    dtype=self.weight_dtype,
                    device="npu",
                ),
                torch.empty(
                    (dummy_block_num, dummy_block_size, model_runner.num_kv_heads,
model_runner.head_size),
                    dtype=self.weight_dtype,
                    device="npu",
                ),
            )
            for _ in range(model_runner.num_layers)
        ]
        max_s = max(attn_metadata.prefill_metadata.seq_lens_tensor)
        max_need_block = math.ceil(max_s / dummy_block_size)
        batch_size = len(attn_metadata.prefill_metadata.seq_lens_tensor)
        block_tables = torch.zeros(batch_size, max_need_block, dtype=int,
device="npu")
        slot = [i for i in range(dummy_block_size)]
        slots = []
        warm_up_len = len(input_ids)
        while warm_up_len > 0:
            if warm_up_len > dummy_block_size:
                slots.extend(slot)
                warm_up_len -= dummy_block_size
            else:
                slots.extend(slot[:warm_up_len])
                warm_up_len = 0
        slots = torch.tensor(slots, dtype=torch.long, device="npu")
        return kv_cache, block_tables, slots
```

- usage
  - __init__.py

    ```
    import types
    from vllm.engine.llm_engine import usage_message
    import vllm_npu.usage.usage_lib as vllm_npu_usage_lib
    usage_message._report_usage_once =
    types.MethodType(vllm_npu_usage_lib._report_usage_once, usage_message)
    ```

  - usage_lib.py

    ```
    # Part of code in this file was copied from project [vLLM Team][vllm] for adapting
    usage
    import platform
    from typing import Any, Dict
    import cpuinfo
    import psutil
    import torch
    import vllm.envs as envs
    from vllm.usage.usage_lib import UsageContext, _detect_cloud_provider,
    _get_current_timestamp_ns
    def _report_usage_once(self, model_architecture: str,
                           usage_context: UsageContext,
                           extra_kvs: Dict[str, Any]) -> None:
        # Platform information
        if torch.npu.is_available():
            device_property = torch.npu.get_device_properties()
            self.gpu_count = torch.npu.device_count()
    ```



```
            self.gpu_type = device_property.name
            self.gpu_memory_per_device = device_property.total_memory
        self.provider = _detect_cloud_provider()
        self.architecture = platform.machine()
        self.platform = platform.platform()
        self.total_memory = psutil.virtual_memory().total
        info = cpuinfo.get_cpu_info()
        self.num_cpu = info.get("count", None)
        self.cpu_type = info.get("brand_raw", "")
        self.cpu_family_model_stepping = ",".join([
            str(info.get("family", "")),
            str(info.get("model", "")),
            str(info.get("stepping", ""))
        ])
        # vLLM information
        import vllm  # delayed import to prevent circular import
        self.context = usage_context.value
        self.vllm_version = vllm.__version__
        self.model_architecture = model_architecture
        # Metadata
        self.log_time = _get_current_timestamp_ns()
        self.source = envs.VLLM_USAGE_SOURCE
        data = vars(self)
        if data["_report_usage_once"] is not None:
            del data["_report_usage_once"]
        if extra_kvs:
            data.update(extra_kvs)
        self._write_to_file(data)
        self._send_to_server(data)
```

- worker
  - __init__.py

    ```
    from vllm_npu.worker.cache_engine import _allocate_kv_cache
    from vllm.worker import cache_engine
    cache_engine.CacheEngine._allocate_kv_cache = _allocate_kv_cache
    ```

  - ascend_model_runner.py

    ```
    # Part of codes in this file was copied from project [vLLM Team][vllm]
    import torch
    from typing import List, Optional, Tuple
    from vllm.logger import init_logger
    from vllm.sequence import (SamplerOutput, SequenceGroupMetadata)
    from vllm.sampling_params import SamplingParams
    from vllm.worker.model_runner import _prepare_fake_inputs, ModelRunner
    from vllm.utils import is_hip
    from vllm.lora.request import LoRARequest
    from vllm.config import (DeviceConfig, LoadConfig, LoRAConfig, ModelConfig,
                             ParallelConfig, SchedulerConfig, VisionLanguageConfig)
    from vllm_npu.model_executor.ascend_model_loader import get_model
    logger = init_logger(__name__)
    LORA_WARMUP_RANK = 8
    class AscendModelRunner(ModelRunner):
        def __init__(
            self,
            model_config: ModelConfig,
            parallel_config: ParallelConfig,
            scheduler_config: SchedulerConfig,
            device_config: DeviceConfig,
            load_config: LoadConfig,
            lora_config: Optional[LoRAConfig],
            mindie_model_config,
            kv_cache_dtype: Optional[str] = "auto",
            is_driver_worker: bool = False,
            vision_language_config: Optional[VisionLanguageConfig] = None,
        ):
            super(AscendModelRunner, self).__init__(
                model_config,
                parallel_config,
                scheduler_config,
    ```



```
        device_config,
        load_config,
        lora_config,
        kv_cache_dtype,
        is_driver_worker,
        vision_language_config
    )
    self.mindie_model_config = mindie_model_config

def load_model(self) -> None:
    self.model = get_model(
        model_config=self.model_config,
        device_config=self.device_config,
        load_config=self.load_config,
        mindie_model_config=self.mindie_model_config,
    )
    if self.kv_cache_dtype == "fp8" and is_hip():
        # Currently scaled KV cache is only enabled on ROCm
        if self.model_config.quantization_param_path is not None:
            if callable(getattr(self.model, "load_kv_cache_scales", None)):
                self.model.load_kv_cache_scales(
                    self.model_config.quantization_param_path)
            else:
                raise RuntimeError(
                    "Using FP8 KV cache and scaling factors provided but "
                    "model %s does not support loading scaling factors.",
                    self.model.__class__)
        else:
            logger.warning(
                "Using FP8 KV cache but no scaling factors "
                "provided. Defaulting to scaling factors of 1.0. "
                "This may lead to less accurate results!")
    elif self.model_config.quantization_param_path is not None:
        logger.warning("KV cache scaling factors provided, "
                       "but the KV cache data type is not FP8. "
                       "KV cache scaling factors will not be used.")
@torch.inference_mode()
def execute_model(
    self,
    seq_group_metadata_list: List[SequenceGroupMetadata],
    kv_caches: List[Tuple[torch.Tensor, torch.Tensor]],
) -> Optional[SamplerOutput]:
    (input_tokens, input_positions, attn_metadata, sampling_metadata,
        lora_requests, lora_mapping, multi_modal_input
        ) = self.prepare_input_tensors(seq_group_metadata_list)
    if self.lora_config:
        self.set_active_loras(lora_requests, lora_mapping)
    # Currently cuda graph is only supported by the decode phase.
    prefill_meta = attn_metadata.prefill_metadata
    decode_meta = attn_metadata.decode_metadata
    model_executable = self.model
    execute_model_kwargs = {
        "input_ids": input_tokens,
        "positions": input_positions,
        "kv_caches": kv_caches,
        "attn_metadata": attn_metadata,
    }
    hidden_states = model_executable(**execute_model_kwargs)
    # Only perform sampling in the driver worker.
    if not self.is_driver_worker:
        return None
    # Sample the next token.
    output = self.model.sample(
        logits=hidden_states,
        sampling_metadata=sampling_metadata,
    )
    return output
@torch.inference_mode()
def profile_run(self) -> None:
```



```
    # Enable top-k sampling to reflect the accurate memory usage.
    sampling_params = SamplingParams(top_p=0.99, top_k=self.vocab_size - 1)
    max_num_batched_tokens = self.scheduler_config.max_num_batched_tokens
    max_num_seqs = self.scheduler_config.max_num_seqs

    dummy_lora_requests = []
    dummy_lora_requests_per_seq = []
    if self.lora_config:
        assert self.lora_manager is not None
        with self.lora_manager.dummy_lora_cache():
            for idx in range(self.lora_config.max_loras):
                lora_id = idx + 1
                dummy_lora_request = LoRARequest(
                    lora_name=f"warmup_{lora_id}",
                    lora_int_id=lora_id,
                    lora_local_path="/not/a/real/path",
                )
                self.lora_manager.add_dummy_lora(dummy_lora_request,
                                                 rank=LORA_WARMUP_RANK)
                dummy_lora_requests.append(dummy_lora_request)
            dummy_lora_requests_per_seq = [
                dummy_lora_requests[idx % len(dummy_lora_requests)]
                for idx in range(max_num_seqs)
            ]
    seqs: List[SequenceGroupMetadata] = []

    if self.vision_language_config:
        max_num_seqs = min(
            max_num_seqs,
            int(max_num_batched_tokens /
                self.vision_language_config.image_feature_size))
    for group_id in range(max_num_seqs):
        seq_len = (max_num_batched_tokens // max_num_seqs +
                   (group_id < max_num_batched_tokens % max_num_seqs))
        seq_data, fake_multi_modal_input = _prepare_fake_inputs(
            seq_len, self.vision_language_config)
        seq = SequenceGroupMetadata(
            request_id=str(group_id),
            is_prompt=True,
            seq_data={group_id: seq_data},
            sampling_params=sampling_params,
            block_tables=None,
            lora_request=dummy_lora_requests_per_seq[group_id]
            if dummy_lora_requests_per_seq else None,
            multi_modal_data=fake_multi_modal_input,
        )
        seqs.append(seq)
    # Run the model with the dummy inputs.
    num_layers = self.model_config.get_num_layers(self.parallel_config)
    kv_caches = [(None, None)] * num_layers
    self.execute_model(seqs, kv_caches)
    torch.npu.synchronize()
    return
```

- ascend_worker.py

```
# Part of codes in this file was copied from project [vLLM Team][vllm]
"""A Ascend worker class."""
import gc
from typing import Dict, List, Tuple, Set, Optional, Any
import torch
import torch.distributed
from vllm.config import (CacheConfig, DeviceConfig, ModelConfig, LoadConfig,
                         ParallelConfig, SchedulerConfig, LoRAConfig,
VisionLanguageConfig)
from vllm.model_executor import set_random_seed
from vllm.distributed import (broadcast_tensor_dict,
                              ensure_model_parallel_initialized,
                              init_distributed_environment)
from vllm.sequence import SamplerOutput, ExecuteModelRequest
from vllm.worker.cache_engine import CacheEngine
```



```
from vllm.lora.request import LoRARequest
from vllm.worker.worker_base import WorkerBase
from vllm.worker.worker import raise_if_cache_size_invalid
from vllm_npu.worker.ascend_model_runner import AscendModelRunner
class AscendWorker(WorkerBase):
    """A worker class that executes the model on a group of Ascend NPUs.
    """
    def __init__(
        self,
        model_config: ModelConfig,
        parallel_config: ParallelConfig,
        scheduler_config: SchedulerConfig,
        device_config: DeviceConfig,
        cache_config: CacheConfig,
        load_config: LoadConfig,
        local_rank: int,
        rank: int,
        distributed_init_method: str,
        lora_config: Optional[LoRAConfig] = None,
        vision_language_config: Optional[VisionLanguageConfig] = None,
        is_driver_worker: bool = False,
    ) -> None:
        self.model_config = model_config
        self.parallel_config = parallel_config
        self.scheduler_config = scheduler_config
        self.device_config = device_config
        self.cache_config = cache_config
        self.local_rank = local_rank
        self.rank = rank
        self.distributed_init_method = distributed_init_method
        self.lora_config = lora_config
        self.load_config = load_config
        self.is_driver_worker = is_driver_worker
        if self.is_driver_worker:
            assert self.rank == 0, "The driver worker must have rank 0."
        if self.model_config.trust_remote_code:
            # note: lazy import to avoid importing torch before initializing
            from vllm.utils import init_cached_hf_modules
            init_cached_hf_modules()
        self.vision_language_config = vision_language_config
        mindie_model_config = {
            'backend_type': 'atb',
            'model_id': model_config.model,
            'rank': rank,
            'local_rank': local_rank,
            'world_size': parallel_config.world_size,
            'npu_device_id': local_rank,
        }
        self.model_runner = AscendModelRunner(
            model_config,
            parallel_config,
            scheduler_config,
            device_config,
            load_config=load_config,
            lora_config=self.lora_config,
            kv_cache_dtype=self.cache_config.cache_dtype,
            is_driver_worker=is_driver_worker,
            mindie_model_config=mindie_model_config)
        # Uninitialized cache engine. Will be initialized by
        # self.initialize_cache().
        self.cache_engine: CacheEngine
        self.gpu_cache: List[torch.Tensor]
    def init_device(self) -> None:
        self.device = torch.device(f"npu:{self.local_rank}")
        torch.npu.set_device(self.device)
        # Initialize the distributed environment.
        init_worker_distributed_environment(self.parallel_config, self.rank,
                                            self.distributed_init_method,
                                            self.local_rank)
```



```
        # Initialize the model.
        set_random_seed(self.model_config.seed)
    def load_model(self):
        self.model_runner.load_model()
    @torch.inference_mode()
    def determine_num_available_blocks(self) -> Tuple[int, int]:
        """Profiles the peak memory usage of the model and returns the maximum
        number of NPU and CPU cache blocks that can be allocated.
        """
        # Profile the memory usage of the model and get the maximum number of
        # cache blocks that can be allocated with the remaining free memory.
        torch.npu.empty_cache()
        torch.npu.reset_peak_memory_stats()
        # Execute a forward pass with dummy inputs to profile the memory usage
        # of the model.
        self.model_runner.profile_run()
        block_size = self.cache_config.block_size
        dummy_block_size = 128
        dummy_num_blocks = dummy_block_size // block_size
        # Calculate the number of blocks that can be allocated with the
        # profiled peak memory.
        torch.npu.synchronize()
        peak_memory = torch.npu.max_memory_allocated()
        total_gpu_memory = torch.npu.get_device_properties(self.rank).total_memory
        cache_block_size = CacheEngine.get_cache_block_size(
            self.cache_config, self.model_config, self.parallel_config)
        num_gpu_blocks = int(
            (total_gpu_memory * self.cache_config.gpu_memory_utilization -
peak_memory) //
            cache_block_size) + dummy_num_blocks
        num_cpu_blocks = int(self.cache_config.swap_space_bytes // cache_block_size)
        num_gpu_blocks = max(num_gpu_blocks, 0)
        num_cpu_blocks = max(num_cpu_blocks, 0)
        if self.model_runner.lora_manager:
            self.model_runner.remove_all_loras()
        gc.collect()
        torch.npu.empty_cache()
        return num_gpu_blocks, num_cpu_blocks
    def initialize_cache(self, num_gpu_blocks: int,
                         num_cpu_blocks: int) -> None:
        raise_if_cache_size_invalid(num_gpu_blocks,
                                    self.cache_config.block_size,
                                    self.model_config.max_model_len)
        self.cache_config.num_gpu_blocks = num_gpu_blocks
        self.cache_config.num_cpu_blocks = num_cpu_blocks

        self._init_cache_engine()
        self._warm_up_model()
    def _init_cache_engine(self):
        assert self.cache_config.num_gpu_blocks is not None
        self.cache_engine = CacheEngine(self.cache_config, self.model_config,
                                        self.parallel_config)
        self.gpu_cache = self.cache_engine.gpu_cache
        self.model_runner.set_block_size(self.cache_engine.block_size)
    def _warm_up_model(self) -> None:
        pass
    def cache_swap(
        self,
        blocks_to_swap_in: Dict[int, int],
        blocks_to_swap_out: Dict[int, int],
        blocks_to_copy: Dict[int, List[int]],
    ) -> None:
        if blocks_to_swap_in:
            self.cache_engine.swap_in(blocks_to_swap_in)
        if blocks_to_swap_out:
            self.cache_engine.swap_out(blocks_to_swap_out)
        if blocks_to_copy:
            self.cache_engine.copy(blocks_to_copy)
    @torch.inference_mode()
```



```
    def execute_model(
        self,
        execute_model_req: Optional[ExecuteModelRequest] = None
    ) -> List[SamplerOutput]:
        if execute_model_req is None:
            seq_group_metadata_list = None
        else:
            seq_group_metadata_list = execute_model_req.seq_group_metadata_list
        if self.is_driver_worker:
            assert seq_group_metadata_list is not None
            assert execute_model_req is not None
            num_seq_groups = len(seq_group_metadata_list)
            blocks_to_swap_in = execute_model_req.blocks_to_swap_in
            blocks_to_swap_out = execute_model_req.blocks_to_swap_out
            blocks_to_copy = execute_model_req.blocks_to_copy
            data: Dict[str, Any] = {
                "num_seq_groups": num_seq_groups,
                "blocks_to_swap_in": blocks_to_swap_in,
                "blocks_to_swap_out": blocks_to_swap_out,
                "blocks_to_copy": blocks_to_copy,
            }
            broadcast_tensor_dict(data, src=0)
        else:
            data = broadcast_tensor_dict(src=0)
            num_seq_groups = data["num_seq_groups"]
            blocks_to_swap_in = data["blocks_to_swap_in"]
            blocks_to_swap_out = data["blocks_to_swap_out"]
            blocks_to_copy = data["blocks_to_copy"]
        self.cache_swap(blocks_to_swap_in, blocks_to_swap_out, blocks_to_copy)
        # If there is no input, we don't need to execute the model.
        if num_seq_groups == 0:
            return []
        output = self.model_runner.execute_model(seq_group_metadata_list,
                                                  self.gpu_cache)
        # Worker only supports single-step execution. Wrap the output in a list
        # to conform to interface.
        return [output]
    def add_lora(self, lora_request: LoRARequest) -> bool:
        return self.model_runner.add_lora(lora_request)
    def remove_lora(self, lora_id: int) -> bool:
        return self.model_runner.remove_lora(lora_id)
    def list_loras(self) -> Set[int]:
        return self.model_runner.list_loras()
    def get_cache_block_size_bytes(self) -> int:
        """Get the size of the KV cache block size in bytes.
        """
        return CacheEngine.get_cache_block_size(self.cache_config,
                                                self.model_config,
                                                self.parallel_config)
def init_worker_distributed_environment(
    parallel_config: ParallelConfig,
    rank: int,
    distributed_init_method: Optional[str] = None,
    local_rank: int = -1,
) -> None:
    """Initialize the distributed environment."""
    init_distributed_environment(parallel_config.world_size, rank,
                                 distributed_init_method, local_rank)
    ensure_model_parallel_initialized(parallel_config.tensor_parallel_size,
                                      parallel_config.pipeline_parallel_size)
def _check_if_gpu_supports_dtype(torch_dtype: torch.dtype):
    # Check if the GPU supports the dtype.
    if torch_dtype == torch.bfloat16:
        compute_capability = torch.cuda.get_device_capability()
        if compute_capability[0] < 8:
            gpu_name = torch.cuda.get_device_name()
            raise ValueError(
                "Bfloat16 is only supported on GPUs with compute capability "
                f"of at least 8.0. Your {gpu_name} GPU has compute capability "
```



```
                f"{compute_capability[0]}.{compute_capability[1]}. "
                "You can use float16 instead by explicitly setting the"
                "`dtype` flag in CLI, for example: --dtype=half.")
```

- cache_engine.py

```
from typing import Tuple, List
import torch
KVCache = Tuple[torch.Tensor, torch.Tensor]
def _allocate_kv_cache(
    self,
    num_blocks: int,
    device: str,
) -> List[KVCache]:
    """Allocates KV cache on the specified device."""
    kv_cache: List[KVCache] = []
    key_block_shape = (self.block_size, self.num_heads, self.head_size)
    value_block_shape = (self.block_size, self.num_heads, self.head_size)
    for _ in range(self.num_layers):
        key_blocks = torch.empty(
            size=(num_blocks, *key_block_shape),
            dtype=self.dtype,
            device=device,
        )
        value_blocks = torch.empty(
            size=(num_blocks, *value_block_shape),
            dtype=self.dtype,
            device=device,
        )
        kv_cache.append((key_blocks, value_blocks))
    return kv_cache
```

- config.py

```
import warnings
import torch
class DeviceConfig:
    def __init__(self, device: str = "auto") -> None:
        if device == "auto":
            # Automated device type detection
            if getattr(torch.version, "cann", None) is not None:
                self.device_type = "npu"
            else:
                warnings.warn(
                    "Failed to detect cann in your environment. \
                    Please check whether you have installed cann correctly. \
                    Now the device type for processing input is set to cpu."
                )
                self.device_type = "cpu"
        else:
            # Device type is assigned explicitly
            self.device_type = device
        self.device = torch.device(self.device_type)
```

- utils.py

```
# Part of codes in this file was copied from project [vLLM Team][vllm]
import socket
import warnings
from functools import lru_cache
import vllm.envs as envs
@lru_cache(maxsize=None)
def is_ascend() -> bool:
    try:
        import torch_npu
    except ImportError:
        torch_npu = None
    return torch_npu is not None
def get_ip() -> str:
    host_ip = envs.VLLM_HOST_IP
    if host_ip:
        return host_ip
    # IP is not set, try to get it from the network interface
```



```
    # try ipv4
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        s.connect(("localhost", 80))  # Doesn't need to be reachable
        socket_name = s.getsockname()[0]
        s.close()
        return socket_name
    except Exception:
        warnings.warn("Encounted with connection errors. Using 0.0.0.0 by default.")
        s.close()
    # try ipv6
    try:
        s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
        # Google's public DNS server, see
        # https://developers.google.com/speed/public-dns/docs/using#addresses
        s.connect(("localhost", 80))  # Doesn't need to be reachable
        socket_name = s.getsockname()[0]
        s.close()
        return socket_name
    except Exception:
        warnings.warn("Encounted with connection errors. Using 0.0.0.0 by default.")
        s.close()
    s.close()
    warnings.warn(
        "Failed to get the IP address, using 0.0.0.0 by default."
        "The value can be set by the environment variable"
        " VLLM_HOST_IP or HOST_IP.",
        stacklevel=2)
    return "0.0.0.0"
```

- npu_adaptor.py

```
import importlib
import sys
def replace_modules(old_module_name, new_module_name):
    if old_module_name in sys.modules:
        del sys.modules[old_module_name]
    sys.modules[old_module_name] = importlib.import_module(new_module_name)
_default_ops = (
    'xformers',
    'xformers.ops',
    'vllm._C',
    'xformers.ops.fmha.attn_bias',
    'vllm.model_executor.layers.ops.sample'
)
for _ops in _default_ops:
    replace_modules(_ops, 'vllm_npu')
# dummy class to avoid import error.
class ops:
    pass
class cuda_utils:
    pass
class cache_ops:
    pass
class BlockDiagonalCausalMask:
    pass
class LowerTriangularMaskWithTensorBias:
    pass
def context_attention_fwd():
    pass
def get_num_triton_sampler_splits():
    pass
def sample():
    pass
```

- __init__.py

```
import torch
import torch_npu
from torch_npu.contrib import transfer_to_npu
from vllm_npu.npu_adaptor import (BlockDiagonalCausalMask,
                                  LowerTriangularMaskWithTensorBias,
```



```
                                cache_ops, cuda_utils, ops,
                                context_attention_fwd,
                                get_num_triton_sampler_splits, sample)
        import vllm_npu.core
        import vllm_npu.engine
        import vllm_npu.worker
        import vllm_npu.model_executor
        import vllm_npu.executor
        import vllm_npu.attention
        import vllm_npu.usage
        from vllm_npu.utils import get_ip
        from vllm_npu.config import DeviceConfig
        import vllm.utils as utils
        import vllm.executor.ray_utils as ray_utils
        import vllm.config as vconfig
        import vllm.engine.arg_utils as varg_utils
        utils.get_ip = get_ip
        ray_utils.get_ip = get_ip
        vconfig.DeviceConfig = DeviceConfig
        varg_utils.DeviceConfig = DeviceConfig
        __version__ = "0.4.2"
```

- examples
  - start_server.sh

```
#!/bin/bash
# ATB加速库环境变量
export ATB_LAYER_INTERNAL_TENSOR_REUSE=1
export ATB_WORKSPACE_MEM_ALLOC_GLOBAL=1
export ATB_OPERATION_EXECUTE_ASYNC=1
export TASK_QUEUE_ENABLE=1
export ATB_CONTENT_NCHW_TO_ND=1
export ATB_CONTEXT_WORKSPACE_RING=1
export HCCL_BUFFSIZE=120
export LCCL_DETERMINISTIC=1
export HCCL_DETERMINISTIC=true
export ATB_OPSRUNNER_KERNEL_CACHE_LOCAL_COUNT=8
export ATB_OPSRUNNER_KERNEL_CACHE_GLOABL_COUNT=16
export ATB_LAUNCH_KERNEL_WITH_TILING=0
export VLLM_NO_USAGE_STATS=1 # close vllm usage messages to avoid errors
python -m vllm.entrypoints.openai.api_server --model=facebook/opt-125m --trust-remote-code
--enforce-eager --worker-use-ray
```

  - test_offline.py

```
from vllm import LLM, SamplingParams
import json
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('--model_path', type=str, default="facebook/opt-125m")
# input prompts for test
prompts = [
    "Hello, my name is",
    "The president of the United States is",
    "The capital of France is",
    "The future of AI is",
]
sampling_params = SamplingParams(max_tokens=512, temperature=0)
args = parser.parse_args()
model_path = args.model_path
llm = LLM(model=model_path,
          block_size=128,
          max_model_len=4096, # max length of prompt
          tensor_parallel_size=8, # number of NPUs to be used
          max_num_seqs=256, # max batch number
          enforce_eager=True, # disable CUDA graph mode
          trust_remote_code=True, # If the model is a custom model not yet available in the
HuggingFace transformers library
)
outputs = llm.generate(prompts, sampling_params)
for i, output in enumerate(outputs):
```



```
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"req_num: {i}\nPrompt: {prompt!r}\nGenerated text: {generated_text!r}")
```

  - test_offline.sh

```
#!/bin/bash
# ATB加速环境变量
export ATB_LAYER_INTERNAL_TENSOR_REUSE=1
export ATB_WORKSPACE_MEM_ALLOC_GLOBAL=1
export ATB_OPERATION_EXECUTE_ASYNC=1
export TASK_QUEUE_ENABLE=1
export ATB_CONTENT_NCHW_TO_ND=1
export ATB_CONTEXT_WORKSPACE_RING=1
export HCCL_BUFFSIZE=120
export LCCL_DETERMINISTIC=1
export HCCL_DETERMINISTIC=true
export ATB_OPSRUNNER_KERNEL_CACHE_LOCAL_COUNT=8
export ATB_OPSRUNNER_KERNEL_CACHE_GLOABL_COUNT=16
export ATB_LAUNCH_KERNEL_WITH_TILING=0
export VLLM_TARGET_DEVICE="npu" # add environment variable to use npu device
export VLLM_NO_USAGE_STATS=1 # close vllm usage messages to avoid errors
python3 test_offline.py --model_path facebook/opt-125m
```

- install.sh

```
#!/bin/bash
if [ -d "./vllm" ]; then
    echo "./vllm directory has already exist!"
    exit 1
fi
git clone -b v0.4.2 https://github.com/vllm-project/vllm.git vllm
cp -r cover/* vllm/
cd vllm
pip install -r requirements-ascend.txt
python3 setup.py install
cd ../vllm_npu
pip install -r requirements.txt
python3 setup.py install
```

  将上述所有代码内容安装项目的目录结构完全还原后，进入项目根目录执行如下脚本，即可完成vLLM昇腾适配版的安装。

```
bash install.sh
```

7.3 TGI 基于 Text Generator 接口开发指南

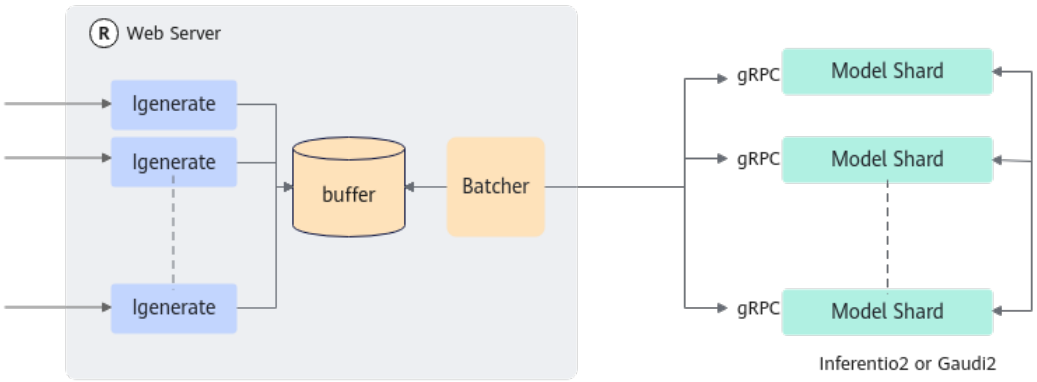
7.3.1 快速介绍

TGI 简介与 TGI 适配昇腾整体方案介绍

TGI（全称：Text Generation Inference）是HuggingFace支持的推理部署工具。TGI本身不支持NPU上的模型运行，通过提供昇腾环境下的TGI适配补丁，帮助客户在昇腾环境下运行TGI框架服务。适配后的系统：

- TGI服务的启动、与Client的交互协议、Batch调度和KV Cache分配功能仍然在TGI框架中完成。
- 模型推理与后处理能力：从开源TGI切换至MindIE LLM提供的统一接口。

图 7-4 TGI 开源架构



说明

上图中，昇腾适配TGI的部分为Model Shard的部分。

支持的版本特性及模型

表 7-21 支持的版本特性及模型

支持的TGI版本	浮点	量化	MoE	单Lora	多模态模型
v2.0.4	同MindIE LLM	同MindIE LLM	同MindIE LLM	同MindIE LLM	Qwen-VL
V0.9.4	同MindIE LLM	同MindIE LLM	-	同MindIE LLM	-

7.3.2 适配说明

7.3.2.1 TGI 昇腾框架适配说明

以TGI v2.0.4版本为例，适配的部分在server/text_generation_server包中，主要适配部分包括：MindIE LLM切入、Batch结构修改、MindIE LLM推理和后处理的调用、Cache管理在NPU上的配置。

步骤1 MindIE LLM切入：将原TGI框架从调用GPU模型切到MindIE LLM

适配文件：server/text_generation_server/models/__init__.py

```
# 1.引入tgi在npu上的适配包，tgi_npu
try:
    import torch_npu
    from tgi_npu import MindModel, VlmMindModel
    npu_module_imported = True
except (ImportError, NotImplementedError):
    npu_module_imported = False

# 2.多模态模型支持Qwen_VL模型；文本模型支持范围同MindIE LLM
if npu_module_imported and torch.npu.is_available():
    if model_type == QWEN_VL:
```

```
        return VlmMindModel(model_id)
    else:
        return MindModel(model_id)
```

步骤2 添加文本模型类MindModel：

MindModel类继承自TGI框架的FlashCausalLM类，主要适配部分为初始化MindIE LLM模型，并获得模型、Tokenizer及模型相关的信息。

```
class MindModel(FlashCausalLM):
    def __init__(
            self,
            model_id: str
    ):
        logger.warning("Initialize mindie-llm model.")
        rank = int(os.getenv("RANK", "0"))
        world_size = int(os.getenv("WORLD_SIZE", "1"))
        model_config = {
            'backend_type': BackendType.ATB,
            'rank': rank,
            'world_size': world_size,
            'model_id': model_id,
            'num_threads': 8,
            'local_rank': rank,
            'npu_device_id': rank
        }
# 1. 初始化mindie llm模型，获得的self.model_runner包含模型信息
        self.model_runner = GeneratorTorch(model_config)
        super(MindModel, self).__init__(
            model=self.model_runner.model_wrapper.model_runner.model,
            tokenizer=self.model_runner.tokenizer,
            num_layers=self.model_runner.model_info.num_layers,
            num_kv_heads=self.model_runner.model_info.num_kv_heads,
            head_size=self.model_runner.model_info.head_size,
            dtype=self.model_runner.model_info.dtype,
            device=self.model_runner.model_info.device,
            rank=self.model_runner.rank,
            world_size=self.model_runner.world_size,
        )
        logger.warning("MindModel from tgi_npu initialized.")

# 2. 为每个batch请求生成token
# 改用MindModel初始化生成的self.model_runner的forward_tensor推理接口（链接）进行推理，输入为MindFlashCausalLMBatch类（链接）中相应字段
self.model_runner.forward_tensor(
    input_ids=input_ids,
    position_ids=position_ids,
    is_prefill=cu_seqlen_prefill is not None,
    kv_cache=kv_cache,
    block_tables=block_tables,
    slots=slots,
    input_lengths=input_lengths,
    max_seq_len=max_s,
    lm_head_indices=lm_head_indices,
)

# 3. 后处理部分，输入参数适配MindIELLMHeterogeneousNextTokenChooser类中的采样方法
(
    next_input_ids,
    next_token_logprobs,
    logprobs,
    accepted_ids,
    speculative_ids,
) = batch.next_token_chooser(
    batch.all_input_ids_tensor[:, : batch.max_seqlen],
    next_token_logits,
    speculate,
    batch.speculative_ids,
    speculative_logits,
)
```


步骤3 （可选）添加多模态模型类VlmMindModel：

多模态模型类继承自MindModel，在初始化部分额外设置了Tokenize方法，该方法由MindIE LLM侧提供，专用于多模态的编码。

```
class VlmMindModel(MindModel):
    def __init__(
            self,
            model_id: str
    ):
        logger.warning("Initialize mindie-llm model for vlm.")
        # 使用父类初始化方法
        super(VlmMindModel, self).__init__(model_id)
        # 额外设置tokenize, 该方法是由MindIE-LLM提供的专用于多模态编码的方法
        self.tokenize = TokenizerWrapper(model_id).tokenize
        logger.warning("VlmMindModel from tgi_npu initialized.")
    @property
    def batch_type(self) -> Type[VlmMindFlashCausalLMBatch]:
        # 返回多模态Batch类型, 用于在server端接收gRPC请求后转换为多模态Batch
        return VlmMindFlashCausalLMBatch
```

步骤4 添加文本请求的Batch类MindFlashCausalLMBatch：

创建MindFlashCausalLMBatch类作为MindIE LLM的文本请求Batch，继承自原TGI框架的FlashCausalLMBatch类。

```
@dataclass
 class MindFlashCausalLMBatch(FlashCausalLMBatch):
# 1. 在基类FlashCausalLMBatch上增加两个字段，next_token_chooser（MindIE LLM的后处理类）、
all_input_ids_tensor为（
# 给next_token_chooser进行sampling）
    next_token_chooser: MindIELLMHeterogeneousNextTokenChooser
    all_input_ids_tensor: torch.Tensor

def from_tokenized(
        cls,
        pb: generate_pb2.Batch,
        tokenizer: PreTrainedTokenizerBase,
        batch_tokenized_inputs,
        dtype: torch.dtype,
        device: torch.device,
) -> "MindFlashCausalLMBatch":

...
# 2.构建后处理类
    next_token_chooser = MindIELLMHeterogeneousNextTokenChooser.from_pb(
        pb=next_token_chooser_parameters, dtype=dtype, device=device
    )

# 3. 构建传给后处理类的input id tensor
    # Padded all_input_ids_tensor
    all_input_ids_tensor = np.zeros(
        (len(all_input_ids), max_length), dtype=np.int64
    )
    for i, input_ids in enumerate(all_input_ids):
        all_input_ids_tensor[i, : len(input_ids)] = input_ids

    all_input_ids_tensor = torch.tensor(
        all_input_ids_tensor, dtype=torch.int64, device=device
    )
```

步骤5 (可选)添加多模态请求的Batch类VlmMindFlashCausalLMBatch：

Prefill和Warmup阶段的Batch是从Web Router通过grpc协议传给text_generation_server，再由text_generation_server反序列化成可供MindIE LLM推理的Batch。而文本和多模态Batch的处理方式不同，多模态支持的模型需要区别引入。

适配文件：server/text_generation_server/server.py

```
# 增加新引入的VlmMindFlashCausalLMBatch
from tgi_npu.vlm_mind_models import VlmMindFlashCausalLMBatch
VLM_BATCH_TYPES = {VlmMindFlashCausalLMBatch}
```

创建VlmMindFlashCausalLMBatch类作为MindIE LLM的多模态请求Batch，继承自NPU适配后的MindFlashCausalLMBatch类。主要是针对入参进行参数拆解组装，并使用Tokenzie对入参进行编码。

```
def split(string) -> List[Dict[str, str]]:
    parts = []
    cursor = 0
    for pattern in IMAGES.finditer(string):
        start = pattern.start()
        if start != cursor:
            parts.append({"text": string[cursor:start]})
        parts.append({"image": pattern.group(1)})
        cursor = pattern.end()
    if cursor != len(string):
        parts.append({"text": string[cursor:]})
    return parts
@dataclass
class VlmMindFlashCausalLMBatch(MindFlashCausalLMBatch):
    @classmethod
    def batch_tokenized_inputs(cls, requests, tokenize):
        inputs = []
        for r in requests:
            splits = split(r.inputs)
            single_input = tokenize(splits).tolist()
            inputs.append(single_input)
        return inputs
    @classmethod
    def from_pb_processor(
            cls,
            pb: generate_pb2.Batch,
            tokenizer: PreTrainedTokenizerBase,
            tokenize,
            dtype: torch.dtype,
            device: torch.device,
    ) -> "VlmMindFlashCausalLMBatch":
        batch_tokenized_inputs = cls.batch_tokenized_inputs(pb.requests, tokenize)
        batch = cls.from_tokenized(pb, tokenizer, batch_tokenized_inputs, dtype, device)
        return batch
```

步骤6 增加后处理类MindIELLMHeterogeneousNextTokenChooser：

后处理类根据Batch中每个Request的采样参数，为每个Batch构造一个后处理对象。Sampling为从推理生成的logits中采样出token(s)。

```
from mindie_llm.text_generator.utils.sampling_metadata import SamplingData, SamplingParam
from mindie_llm.text_generator.utils.config import SamplerConfig
from mindie_llm.text_generator.samplers.sampler import Sampler
 from mindie_llm.modeling.backend_type import BackendType

class MindIELLMHeterogeneousNextTokenChooser:

# 1. 初始化函数，从grpc接收的后采样参数构造mindie sampler
curr_rank = int(os.getenv("RANK", "0"))
 sample_method = Sampler(SamplerConfig(rank=curr_rank, backend_type=BackendType.ATB,
npu_id=curr_rank))
self.tensor_wrapper = TensorWrapper(BackendType.ATB, device)
 wrapper_dict = {WRAPPER_KEY: TensorWrapper(BackendType.ATB, device)}
self.sample_params = SamplingParam.from_numpy(
    repetition_penalty=np.array(repetition_penalty, dtype=np.float16),
    presence_penalty=None,
    frequency_penalty=np.array(frequency_penalty, dtype=np.float16),
    temperature=np.array(temperature, dtype=np.float16),
```



```
    top_k=np.array(top_k),
    top_p=np.array(top_p),
    seed=np.array(seeds).astype(np.int32),
    do_sample=np.array(do_sample),
    **wrapper_dict
)

self.choice = sample_method
self.dtype = dtype
self.device = device
self.seeds = seeds
 self.do_sample = self.sample_params.do_sample_meta.do_sample_array

# 2. 根据输入token id及推理的logits(scores)，采样出下一个tokenid（next_ids）
def __call__(self,
             input_ids: torch.Tensor,
             scores: torch.Tensor,
             speculate: int,
             speculated_ids: Optional[torch.Tensor] = None,
             speculative_scores: Optional[torch.Tensor] = None,
             ):
    batch_size = scores.shape[0]
    speculate_size = 1
    scores = scores.view(batch_size, speculate_size, -1)

    input_ids_int32 = input_ids.to(torch.int32)
    sample_data = SamplingData(all_input_ids=input_ids_int32, output_ids=input_ids_int32)
    next_ids = torch.zeros((batch_size, speculate_size), device=scores.device, dtype=torch.long)
    for j in range(speculate_size):
        _scores = scores[:, j]

        batch_logits, _next_ids = self.choice(batch_logits=_scores, batch_sampling_data=sample_data,
                                              batch_sampling_params=self.sample_params)
        scores[:, j] = _scores
        next_ids[:, j] = torch.from_numpy(_next_ids)
    next_ids = next_ids.view(batch_size * speculate_size)
    allscores = scores.view(batch_size * speculate_size, -1)
    alllogprobs = torch.log_softmax(allscores, -1)

    accepted_ids = torch.ones_like(next_ids)
    logprobs = alllogprobs

    next_logprobs = torch.gather(logprobs, 1, next_ids.view(-1, 1)).view(-1)

    speculative_ids = None
    return next_ids, next_logprobs, alllogprobs, accepted_ids, speculative_ids
```

步骤7 Cache管理：

Warmup阶段的目的是计算出TGI NPU能预留出的最大KV Cache，以此计算出最大Batch Token数量。

- 需要适配的部分是获取Warmup Batch计算时峰值NPU显存占用、NPU卡最大显存、以及计算出装满剩余显存的最大Token数量，峰值时Token数量的计算。

  类：MindModel

  方法：warmup（用下面方式修改基类FlashCausalLM的Warmup中获取GPU显存）

```
# NPU卡总显存
total_gpu_memory = torch_npu.npu.get_device_properties(self.device).total_memory

# 峰值显存
 peak_memory = torch_npu.npu.max_memory_allocated()
# 剩余显存
 total_free_memory = total_gpu_memory - peak_memory
 logger.warning(f">>>>total_free_memory {total_free_memory}, total_gpu_memory
{total_gpu_memory}, "
                f"MEMORY_FRACTION {MEMORY_FRACTION}")
```



```
# 剩余可用显存
free_memory = max(0, total_free_memory - (1 - MEMORY_FRACTION) * total_gpu_memory)
# 总共可支持的KV Block
num_blocks = (
    # Leave 5% for some wiggle room
        int((free_memory * 0.95) // total_cache_size)
        # Add batch.blocks as we allocated it above, so it is included in the peak memory.
        + cache_manager.num_blocks
)
#总共可支持的KV slot（total batch tokens），BLOCK_SIZE建议128
return int(num_blocks * BLOCK_SIZE)
```

- 为Batch中每个request分配KV Cache由CacheManager类负责，与GPU上的CacheManager类相比，进行如下适配：

  每个BLOCK_SIZE（即一个BLOCK中SLOT数量）建议设置为128。

  NPU卡上，910是ND数据排布, 310是NZ排布，KV Cache分配需要考虑。

  适配文件：server\text_generation_server\models\cache_manager.py

  类：CacheManager

  方法：__init__

```
from tgi_npu.info import NPUSocInfo
# 1. 建议修改为128
BLOCK_SIZE: int = 128
# Will be set in warmup
CACHE_MANAGER: Optional["CacheManager"] = None

class CacheManager:
    def __init__(
            self,
            num_blocks: int,
            num_layers: int,
            num_heads: int,
            head_size: int,
            repeat_slots: bool,
            dtype: torch.dtype,
            device: torch.device,
    ):
        self.block_size = BLOCK_SIZE
        self.num_blocks = num_blocks
        self.repeat_slots = repeat_slots
        soc_info = NPUSocInfo()
        # 2.根据npu卡设置kvcache中tensor的数据排布格式，for 910 ND, 310 NZ
        self.need_nz = soc_info.need_nz
        if self.need_nz:
            self.kv_cache = [
                (
                    torch.empty(
                        (num_blocks, num_heads * head_size // 16, self.block_size, 16),
                        dtype=dtype,
                        device=device,
                    ),
                    torch.empty(
                        (num_blocks, num_heads * head_size // 16, self.block_size, 16),
                        dtype=dtype,
                        device=device,
                    ),
                )
                for _ in range(num_layers)
            ]
        else:
            self.kv_cache = [
                (
                    torch.empty(
                        (num_blocks, self.block_size, num_heads, head_size),
                        dtype=dtype,
                        device=device,
                    ),
```



```
                torch.empty(
                    (num_blocks, self.block_size, num_heads, head_size),
                    dtype=dtype,
                    device=device,
                ),
            )
            for _ in range(num_layers)
        ]
    self.free_block_mask = torch.ones(num_blocks, dtype=torch.int32, device="cpu")
    self.slots = torch.arange(
        0, num_blocks * self.block_size, dtype=torch.int64
    ).view(num_blocks, self.block_size)
```

- NPU卡信息判断：

```
@dataclass
class NPUSocInfo:
    soc_name: str = ""
    soc_version: int = -1
    need_nz: bool = False
        self.soc_version = torch_npu._C._npu_get_soc_version()
        if self.soc_version in (100, 101, 102, 103, 104, 200, 201, 202, 203):
            self.need_nz = True
```

----结束

7.3.2.2 环境安装与启动服务

前提条件

- 已参见《MindIE安装指南》中“**安装驱动和固件**”章节完成驱动和固件的安装。
- 已参见《MindIE安装指南》中“**安装开发环境**”章节完成CANN、Python 3.10.2、PyTorch 2.1.0框架和Torch_NPU 2.1.0插件的安装。
- 已参见《MindIE安装指南》中“**物理机部署MindIE**”章节完成MindIE的安装。

安装步骤

步骤1 安装Rust与必要软件包：

```
# 对于ARM 64位CPU为aarch64，对于X86 64位CPU可将下面指令的aarch64替换为x86_64
wget https://static.rust-lang.org/dist/rust-1.79.0-aarch64-unknown-linux-gnu.tar.gz --no-check-certificate
tar -xvf rust-1.79.0-aarch64-unknown-linux-gnu.tar.gz
cd rust-1.79.0-aarch64-unknown-linux-gnu
bash install.sh

sudo apt update
apt install pkg-config
```

步骤2 设置相关环境变量：

首先在命令行里运行python，通过torch.__file__的路径确认Protoc所在目录，以Python 3.10.2为例：

```
Python 3.10.2 (main, Sep 23 2024, 08:51:58) [GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.__file__
'/usr/local/python3.10.2/lib/python3.10/site-packages/torch/__init__.py'
```

控制台输出的__init__.py所在目录的子文件夹bin下即为Protoc的放置路径。随后将Cargo的可执行文件目录和Protoc目录导出到$PATH（在进行下一步骤前该目录可能为空或不存在）：

```
# Cargo编译出的可执行文件目录
export PATH=$PATH:~/.cargo/bin/
```



```
# protoc所在目录
export PATH=/usr/local/python3.10.2/lib/python3.10/site-packages/torch/bin:$PATH
```

步骤3 参照**7.3.3 样例代码**设置文件目录与代码内容，并运行一键安装脚本install.sh进行环境安装与代码编译。脚本如下：

```
bash install.sh
```

安装完成后，可检查可执行文件和安装包是否已存在。

1）在~/.cargo/bin/目录下包括两个可执行文件：text-generation-launcher和text-generation-router

```
ll ~/.cargo/bin/
-rwxr-xr-x 1 root root  22714304 Nov  8 15:51 text-generation-launcher*
-rwxr-xr-x 1 root root 113330192 Nov  8 15:49 text-generation-router*
```

2）已安装两个python包：text-generation-server和tgi_npu

```
pip show text-generation-server
Name: text-generation-server
Version: 2.0.4
Summary: Text Generation Inference Python gRPC Server
Home-page:
Author: Olivier Dehaene
Author-email: olivier@huggingface.co
pip show tgi_npu
Name: tgi-npu
Version: 0.1.0
Summary: NPU MindIE Adapter for TGI v2.0.4
```

步骤4 安装并启动Nginx：

```
apt update
apt install nginx
service nginx start
```

步骤5 编辑tgi代理配置文件：

```
vi /etc/nginx/sites-available/tgi_proxy
```

将以下内容粘贴至配置文件tgi_proxy：

```
server {
    listen 12346 ssl; # 设置Nginx监听端口并启用SSL，HTTPS默认端口为443；注意，此端口需要与启动TGI服务端口不一致，发送请求时请指定使用此端口号（根据使用需求修改）
    server_name localhost; # 设置Nginx服务器名称，此处使用本机IP地址(等价于127.0.0.1)；注意，发送请求时请指定使用此名称（根据使用需求修改）

    ssl_certificate /path/to/your/certificate.crt; # 此处填写用于Nginx服务的证书(受信任的机构签发的crt文件)路径（根据实际路径修改）
    ssl_certificate_key /path/to/your/key.key; # 此处填写与证书相匹配的私钥（证书配套的key文件）路径（根据实际路径修改）

    location / {
        proxy_pass http://127.0.0.1:12347; # 此处填写启动TGI服务所使用的IP地址和端口（根据使用需求修改）
        proxy_set_header Host $host; # 保留原始主机头信息（无需修改）
        proxy_set_header X-Real-IP $remote_addr; # 向后端服务传递客户端的IP地址（无需修改）
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for; # 传递请求所经过的代理链信息（无需修改）
        proxy_set_header X-Forwarded-Proto $scheme; # 传递原始请求的协议类型（无需修改）
    }
}
```

步骤6 通过创建符号链接启用tgi_proxy配置文件并重启Nginx：

```
ln -s /etc/nginx/sites-available/tgi_proxy /etc/nginx/sites-enabled/
service nginx restart
```

步骤7 服务端使用拉起服务脚本拉起TGI在线推理服务：

```
# 控制框架占用显存比例
export CUDA_MEMORY_FRACTION=0.9
```



```
# 系统可见显卡id
export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
# 本地模型权重路径或在Huggingface代码仓中的位置
model_path=/home/data/models/qwen2_7B_instruct
# 以下启动参数与原生TGI一致
text-generation-launcher \
 --model-id $model_path \
 --port 12347 \
 --max-input-length 2048 \
 --max-total-tokens 2560 \
 --sharded true \
 --num-shard 8 \
 --max-batch-prefill-tokens 8192 \
 --max-waiting-tokens 20 \
 --max-concurrent-requests 256 \
 --waiting-served-ratio 1.2
```

步骤8 客户端使用curl、requests等方式向服务端发送基于HTTPS协议推理请求并接收响应：

```
curl  https://127.0.0.1:12346/generate -X POST -d '{"inputs":"Please introduce yourself.","parameters":{"max_new_tokens":64,"repetition_penalty":1.2}}' -H 'Content-Type: application/json'
```

说明

原生TGI基于HTTP协议提供推理服务，请求命令如下（由于HTTP协议安全性问题，不推荐此方式）：

```
curl  http://127.0.0.1:12347/generate -X POST -d '{"inputs":"Please introduce yourself.","parameters":{"max_new_tokens":64,"repetition_penalty":1.2}}' -H 'Content-Type: application/json'
```

----结束

7.3.3 样例代码

7.3.3.1 TGI v2.0.4 版本参考适配代码

文件目录结构如下：

```
Tgi-MindIE
|_____cover
      |_____models
            |________init__.py
      |_____cli.py
      |_____server.py
|_____tgi_npu
      |_____init__.py
      |____cache_manager.py
      |____info.py
      |____mind_model.py
      |____token_mindie.py
      |____vlm_mind_models.py
|_____pyproject.toml
|_____install.sh
|_____README.md
```

各源文件的含义和作用如下表所示：

源文件	含义及作用
cover/models/__init__.py	替换原仓中server/text_generation_server/models/__init__.py文件，将推理模型引导至MindIE LLM。
cover/cli.py	替换原仓中server/text_generation_server/cli.py文件，添加tgi_npu模块的日志打印过滤。

源文件	含义及作用
cover/server.py	替换原仓中server/text_generation_server/server.py文件，添加tgi_npu支持多模态模型入口功能。
tgi_npu/__init__.py	针对NPU硬件环境进行必要的初始化。
tgi_npu/cache_manager.py	KV Cache管理器，主要针对NPU进行KV Cache初始化。
tgi_npu/info.py	NPU信息。
tgi_npu/mind_model.py	定义了推理模型入口类MindModel以及对应的数据通信格式MindFlashCasualLMBatch，分别继承自原仓的FlashCasualLM以及FlashCasualLMBatch。在MindModel中，generate_token方法沿用了原版大部分代码，并结合MindIE LLM调用过程进行了修改。其中，Forward方法改为调用MindIE LLM提供的forward_tensor方法。warmup 结合NPU访存特点进行修改。
tgi_npu/token_mindie.py	后采样代码。
tgi_npu/vlm_mind_models.py	定义了多模态模型入口类VlmMindModel以及对应的数据通信格式。VlmMindFlashCasualLMBatch，分别继承自MindModel以及MindFlashCasualLMBatch。在VlmMindFlashCasualLMBatch中，batch_tokenized_inputs方法中使用了MindIE LLM模块提供的Tokenize方法，将输入编码为符合多模态模型输入要求的格式。
pyproject.toml	适配安装包配置文件。

样例代码：

- Tgi-MindIE/install.sh

```
#!/usr/bin/env bash
# install-origin
if [ -d "./tgi_origin" ]; then
    echo "./tgi_origin directory has already exist!"
    exit 1
fi

git clone -b v2.0.4 https://github.com/huggingface/text-generation-inference.git tgi_origin

cp cover/cli.py tgi_origin/server/text_generation_server/
cp cover/models/__init__.py tgi_origin/server/text_generation_server/models
sed -i "s/requires_padding, 16, window_size/requires_padding, 128, window_size/g" tgi_origin/
router/src/infer.rs
sed -i "s/prefill_logprobs: true/prefill_logprobs: false/g" tgi_origin/router/client/src/client.rs
sed -i "s/bnb, accelerate, quantize, peft, outlines/accelerate, quantize, peft, outlines/g" tgi_origin/
server/Makefile

cd tgi_origin && make install-server && make install-router && make install-launcher

cd .. && pip install -e .
```


- Tgi-MindIE/cover/models/__init__.py

```
# This file was copied from project[huggingface][text-generation-inference]

from typing import Optional

import torch
from loguru import logger

from text_generation_server.utils.speculate import get_speculate, set_speculate
from text_generation_server.models.model import Model

# The flag below controls whether to allow TF32 on matmul. This flag defaults to False
# in PyTorch 1.12 and later.
torch.backends.cuda.matmul.allow_tf32 = True

# The flag below controls whether to allow TF32 on cuDNN. This flag defaults to True.
torch.backends.cudnn.allow_tf32 = True

# Disable gradients
torch.set_grad_enabled(False)

__all__ = [
    "Model",
    "get_model",
]


def get_model(
    model_id: str,
    revision: Optional[str],
    sharded: bool,
    quantize: Optional[str],
    speculate: Optional[int],
    dtype: Optional[str],
    trust_remote_code: bool,
) -> Model:
    if speculate is not None:
        logger.warning("Speculate Decoding is not supported now!")
    set_speculate(0)

    try:
        import torch_npu
        from tgi_npu import MindModel
        npu_module_imported = True
    except (ImportError, NotImplementedError) as excp:
        npu_module_imported = False
        logger.error(f"Error catched: {str(excp)}")

    if npu_module_imported and torch.npu.is_available():
        return MindModel(model_id)
    else:
        logger.error("NPU enviroment error!!!!!!!!!!!!")
        raise ValueError("NPU enviroment error!!!!!!!!!!!!")
```

- Tgi-MindIE/cover/cli.py

```
import os
import sys
from pathlib import Path
from typing import Optional
from enum import Enum
import typer
from loguru import logger
from huggingface_hub import hf_hub_download
app = typer.Typer()
MODEL_SUFFIX = ".safetensors"
CONFIG_FILENAME = "config.json"


class Quantization(str, Enum):
```



```
    bitsandbytes = "bitsandbytes"
    bitsandbytes_nf4 = "bitsandbytes-nf4"
    bitsandbytes_fp4 = "bitsandbytes-fp4"
    gptq = "gptq"
    awq = "awq"
    eetq = "eetq"
    fp8 = "fp8"


class Dtype(str, Enum):
    float16 = "float16"
    bloat16 = "bfloat16"


@app.command()
def serve(
        model_id: str,
        revision: Optional[str] = None,
        sharded: bool = False,
        quantize: Optional[Quantization] = None,
        speculate: Optional[int] = None,
        dtype: Optional[Dtype] = None,
        trust_remote_code: bool = False,
        uds_path: Path = "/tmp/text-generation-server",
        logger_level: str = "INFO",
        json_output: bool = False,
        otlp_endpoint: Optional[str] = None,
):
    if sharded:
        assert (
                os.getenv("RANK", None) is not None
        ), "RANK must be set when sharded is True"
        assert (
                os.getenv("WORLD_SIZE", None) is not None
        ), "WORLD_SIZE must be set when sharded is True"
        assert (
                os.getenv("MASTER_ADDR", None) is not None
        ), "MASTER_ADDR must be set when sharded is True"
        assert (
                os.getenv("MASTER_PORT", None) is not None
        ), "MASTER_PORT must be set when sharded is True"

    # Remove default handler
    logger.remove()
    logger.add(
        sys.stdout,
        format="{message}",
        filter="text_generation_server",
        level=logger_level,
        serialize=json_output,
        backtrace=True,
        diagnose=False,
    )
    logger.add(
        sys.stdout,
        format="{message}",
        filter="tgi_npu",
        level=logger_level,
        serialize=json_output,
        backtrace=True,
        diagnose=False,
    )
```

- Tgi-MindIE/cover/server.py

```
import asyncio
import os
import torch
import torch_npu
import time
import signal
```



```
from grpc import aio
from loguru import logger
from grpc_reflection.v1alpha import reflection
from pathlib import Path
from typing import List, Optional
from text_generation_server.cache import Cache
from text_generation_server.interceptor import ExceptionInterceptor
from text_generation_server.models import Model, get_model
try:
    from tgi_npu.vlm_mind_models import VlmMindFlashCausalLMBatch
    VLM_BATCH_TYPES = {VlmMindFlashCausalLMBatch}
except (ImportError, NotImplementedError):
    # These imports can fail on CPU/Non flash.
    VLM_BATCH_TYPES = set()
from text_generation_server.pb import generate_pb2_grpc, generate_pb2
from text_generation_server.tracing import UDSOpenTelemetryAioServerInterceptor
from text_generation_server.models.globals import set_model_id
soc_version = torch_npu._C._npu_get_soc_version()
if soc_version not in [104,220,221,222,223,224]:
    logger.info("Some ops do not support in this soc !")
    option = {"NPU_FUZZY_COMPILE_BLACKLIST": "ReduceNansum"}
    torch.npu.set_option(option)
else:
    option = {"NPU_FUZZY_COMPILE_BLACKLIST": "GatherElements"}
torch.npu.set_option(option)
class SignalHandler:
    KEEP_PROCESSING = True

    def __init__(self):
        signal.signal(signal.SIGINT, self.exit_gracefully)
        signal.signal(signal.SIGTERM, self.exit_gracefully)
    def exit_gracefully(self, signum, frame):
        print(f"Exiting gracefully: Signal {signum}")
        self.KEEP_PROCESSING = False


class TextGenerationService(generate_pb2_grpc.TextGenerationServiceServicer):
    def __init__(
        self,
        model: Model,
        cache: Cache,
        quantize: Optional[str],
        server_urls: List[str],
    ):
        self.cache = cache
        self.model = model
        self.quantize = quantize
        self.server_urls = server_urls
        # For some reason, inference_mode does not work well with GLOO which we use on CPU
        if model.device.type == "cuda" or model.device.type == "npu":
            # Force inference mode for the lifetime of TextGenerationService
            self._inference_mode_raii_guard = torch._C._InferenceMode(True)
        self.step = 0

    async def Info(self, request, context):
        return self.model.info
    async def Health(self, request, context):
        if self.model.device.type == "cuda":
            torch.zeros((2, 2)).cuda()
        return generate_pb2.HealthResponse()
    async def ServiceDiscovery(self, request, context):
        return generate_pb2.ServiceDiscoveryResponse(urls=self.server_urls)
    async def ClearCache(self, request, context):
        if request.HasField("id"):
            self.cache.delete(request.id)
        else:
            self.cache.clear()
        return generate_pb2.ClearCacheResponse()
    async def FilterBatch(self, request, context):
```



```
        batch = self.cache.pop(request.batch_id)
        if batch is None:
            raise ValueError(f"Batch ID {request.batch_id} not found in cache.")
        filtered_batch = batch.filter(request.request_ids)
        self.cache.set(filtered_batch)
        return generate_pb2.FilterBatchResponse(batch=filtered_batch.to_pb())
    async def Warmup(self, request, context):
        for i, r in enumerate(request.batch.requests):
            r.parameters.typical_p = 1.0
            r.prefill_logprobs = False
        if self.quantize == "gptq":
            try:
                # When using GPTQ, Exllama kernels need some global kernels
                # For which we have the finale shapes only after the model has loaded
                # This will allocate those buffers.
                from text_generation_server.layers.gptq import (
                    create_exllama_buffers,
                    set_device,
                )
                set_device(self.model.device)
                create_exllama_buffers(request.max_prefill_tokens)
            except ImportError:
                pass

        if (
            self.model.batch_type in VLM_BATCH_TYPES
        ):  # Hack, i would rather use kwargs in the `from_pb` call
            for i, r in enumerate(request.batch.requests):
                r.inputs = r.inputs.split('!')[0]
            batch = self.model.batch_type.from_pb_processor(
                request.batch,
                self.model.tokenizer,
                self.model.tokenize,
                self.model.dtype,
                self.model.device,
            )
        else:
            batch = self.model.batch_type.from_pb(
                request.batch, self.model.tokenizer, self.model.dtype, self.model.device
            )
        max_supported_total_tokens = self.model.warmup(batch)
        return generate_pb2.WarmupResponse(
            max_supported_total_tokens=max_supported_total_tokens
        )
    async def Prefill(self, request, context):
        start = time.time_ns()
        if (
            self.model.batch_type in VLM_BATCH_TYPES
        ):  # Hack, i would rather use kwargs in the `from_pb` call
            batch = self.model.batch_type.from_pb_processor(
                request.batch,
                self.model.tokenizer,
                self.model.tokenize,
                self.model.dtype,
                self.model.device,
            )
        else:
            batch = self.model.batch_type.from_pb(
                request.batch, self.model.tokenizer, self.model.dtype, self.model.device
            )
        generations, next_batch, timings = self.model.generate_token(batch)
        self.cache.set(next_batch)
        return generate_pb2.PrefillResponse(
            generations=[generation.to_pb() for generation in generations],
            batch=next_batch.to_pb() if next_batch else None,
            forward_ns=timings[0],
            decode_ns=timings[1],
            total_ns=time.time_ns() - start,
        )
```


- Tgi-MindIE/tgi_npu/__init__.py

```
#!/usr/bin/env python3
# coding=utf-8
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.


import torch
import torch_npu
from loguru import logger

from tgi_npu.mind_models import MindModel


def init():
    torch._C._InferenceMode(True)
    soc_version = torch_npu._C._npu_get_soc_version()
    if soc_version not in [104, 220, 221, 222, 223, 224]:
        logger.info("Some op does not support for this soc!")
        option = {"NPU_FUZZY_COMPILE_BLACKLIST": "ReduceNansum"}
    else:
        option = {"NPU_FUZZY_COMPILE_BLACKLIST": "GatherElements"}
    try:
        torch.npu.set_option(option)
        logger.warning("Finish init for NPU device!")
    except Exception as e:
        logger.error(f"Failed to init for NPU device: {e}!")


init()
```

- Tgi-MindIE/tgi_npu/cache_manager.py

```
# Part of codes in this file was copied from project[huggingface][text-generation-inference]

import math
from typing import Optional, List, Tuple
import gc
import torch
from tgi_npu.info import NPUSocInfo

BLOCK_SIZE: int = 128
# Will be set in warmup
CACHE_MANAGER: Optional["CacheManager"] = None


class CacheManager:
    def __init__(
            self,
            num_blocks: int,
            num_layers: int,
            num_heads: int,
            head_size: int,
            repeat_slots: bool,
            dtype: torch.dtype,
            device: torch.device,
    ):
        self.block_size = BLOCK_SIZE
        self.num_blocks = num_blocks
        self.repeat_slots = repeat_slots
        # for NZ/ND data format display
        self.need_nz = NPUSocInfo().need_nz

        if self.need_nz:
            self.kv_cache = [
                (
                    torch.empty(
                        (num_blocks, num_heads * head_size // 16, self.block_size, 16),
                        dtype=dtype,
                        device=device,
                    ),
```



```
                torch.empty(
                    (num_blocks, num_heads * head_size // 16, self.block_size, 16),
                    dtype=dtype,
                    device=device,
                ),
            )
            for _ in range(num_layers)
        ]
    else:
        self.kv_cache = [
            (
                torch.empty(
                    (num_blocks, self.block_size, num_heads, head_size),
                    dtype=dtype,
                    device=device,
                ),
                torch.empty(
                    (num_blocks, self.block_size, num_heads, head_size),
                    dtype=dtype,
                    device=device,
                ),
            )
            for _ in range(num_layers)
        ]
    self.free_block_mask = torch.ones(num_blocks, dtype=torch.int32, device="cpu")
    self.slots = torch.arange(
        0, num_blocks * self.block_size, dtype=torch.int64
    ).view(num_blocks, self.block_size)

def __repr__(self):
    return (f"CacheManager: "
            f"num_blocks={self.num_blocks},"
            f"block_size={self.block_size},"
            f"free_block_mask={self.free_block_mask},"
            f"slots={self.slots},"
            f"k_cache shape={self.kv_cache[0][0].shape},"
            f"v_cache shape={self.kv_cache[0][1].shape}")

def allocate(
        self,
        needed_blocks_slots: List[Tuple[int, int]],
        blocks: int,
        max_blocks: int,
        device: torch.device,
):
    # Get free blocks indices by finding values in mask that are not set to 0
    free_block_indices = self.free_block_mask.nonzero()
    if blocks > len(free_block_indices):
        raise RuntimeError(
            f"Out of available cache blocks: asked {blocks}, only {len(free_block_indices)} free blocks"
        )

    # Slice by the number of required blocks
    block_indices = free_block_indices[:blocks]
    block_indices = block_indices.flatten()

    # Padded block tables
    block_tables_tensor = torch.zeros(
        (len(needed_blocks_slots), max_blocks), dtype=torch.int32
    )

    # Allocate paged attention blocks
    cumulative_blocks = 0
    slots = []
    block_tables = []
    for i, (needed_blocks, needed_slots) in enumerate(needed_blocks_slots):
        # Get allocated blocks for this sequence
        allocated_blocks = block_indices[
                           cumulative_blocks: cumulative_blocks + needed_blocks
```



```
                        ]
            # Get slots for the allocated blocks
            all_slots = self.slots[allocated_blocks].flatten()

            # Repeat slots in the case of context sliding window
            if needed_slots > len(all_slots) and self.repeat_slots:
                repeats = math.ceil(needed_slots / len(all_slots))
                all_slots = all_slots.repeat(repeats)

            allocated_slots = all_slots[:needed_slots]

            slots.append(allocated_slots)
            block_tables.append(allocated_blocks.tolist())
            block_tables_tensor[i, :needed_blocks] = allocated_blocks
            cumulative_blocks += needed_blocks

        block_tables = block_tables
        block_tables_tensor = block_tables_tensor.to(device)
        slots = torch.concat(slots).to(device)

        # Allocate the required number of blocks by setting the mask to 0
        self.free_block_mask[block_indices] = 0

        return block_tables, block_tables_tensor, slots

    def free(self, block_indices: Optional[List[int]]):
        if block_indices is not None and block_indices:
            # Reset mask
            self.free_block_mask[block_indices] = 1


def set_cache_manager(
        num_blocks: int,
        num_layers: int,
        num_heads: int,
        head_size: int,
        repeat_slots: bool,
        dtype: torch.dtype,
        device: torch.device,
) -> CacheManager:
    global CACHE_MANAGER
    if CACHE_MANAGER is not None:
        del CACHE_MANAGER
        torch.npu.empty_cache()
        gc.collect()

    CACHE_MANAGER = CacheManager(
        num_blocks, num_layers, num_heads, head_size, repeat_slots, dtype, device
    )
    return CACHE_MANAGER


def get_cache_manager() -> CacheManager:
    global CACHE_MANAGER
    if CACHE_MANAGER is None:
        raise RuntimeError("cache manager was not initialized")

    return CACHE_MANAGER
```

- Tgi-MindIE/tgi_npu/info.py

```
from dataclasses import dataclass
import torch_npu

@dataclass
class NPUSocInfo:
     soc_name: str = ""
     soc_version: int = -1
     need_nz: bool = False
     def __post_init__(self):
          self.soc_version = torch_npu._C._npu_get_soc_version()
```



```
        if self.soc_version in (100, 101, 102, 103, 104, 200, 201, 202, 203):
            self.need_nz = True
```

- Tgi-MindIE/tgi_npu/mind_model.py

```
# Part of codes in this file was copied from project[huggingface][text-generation-inference]


import math
import time
import os
from typing import Optional, Tuple, List, Type
from dataclasses import dataclass

import torch_npu
import torch
from loguru import logger
from opentelemetry import trace
import numpy as np

from text_generation_server.models.flash_causal_lm import FlashCausalLM, FlashCausalLMBatch
from text_generation_server.models.types import (
    Batch,
    Tokens,
    Generation,
    GeneratedText
)

from text_generation_server.utils import StoppingCriteria
from text_generation_server.pb import generate_pb2
from text_generation_server.utils.speculate import get_speculate
from text_generation_server.utils.dist import RANK, MEMORY_FRACTION
from text_generation_server.utils.tokens import batch_top_tokens
from text_generation_server.models import cache_manager as tgi_cache_manager

from transformers import PreTrainedTokenizerBase
from mindie_llm.text_generator.adapter.generator_torch import GeneratorTorch
from mindie_llm.modeling.backend_type import BackendType

from tgi_npu.tokens_mindie import MindIELLMHeterogeneousNextTokenChooser

from tgi_npu.cache_manager import (
    BLOCK_SIZE,
    get_cache_manager,
    set_cache_manager,
)

tracer = trace.get_tracer(__name__)


@dataclass
class MindFlashCausalLMBatch(FlashCausalLMBatch):
    next_token_chooser: MindIELLMHeterogeneousNextTokenChooser
    all_input_ids_tensor: torch.Tensor

    def __repr__(self):
        return (f"MindFlashCausalLMBatch: batch_id={self.batch_id},"
                f"requests_idx_mapping={self.requests_idx_mapping},"
                f"input_ids={self.input_ids},"
                f"position_ids={self.position_ids},"
                f"cu_seqlen_prefill={self.cu_seqlen_prefill},"
                f"start_slots={self.start_slots},"
                f"slot_indices={self.slot_indices},"
                f"needed_blocks_slots={self.needed_blocks_slots},"
                f"block_tables={self.block_tables},"
                f"block_tables_tensor={self.block_tables_tensor},"
                f"slots={self.slots},"
                f"max_seqlen={self.max_seqlen},"
                f"prefill_head_indices={self.prefill_head_indices},"
                f"prefill_next_token_indices={self.prefill_next_token_indices},"
                f"prefill_cu_outlens={self.prefill_cu_outlens},"
```



```
            f"input_lengths={self.input_lengths},"
            f"input_lengths_tensor={self.input_lengths_tensor},"
            f"prefix_offsets={self.prefix_offsets},"
            f"read_offsets={self.read_offsets},"
            f"all_input_ids_tensor={self.all_input_ids_tensor},"
            f"next_token_chooser={self.next_token_chooser},"
            f"stopping_criterias={self.stopping_criterias},"
            f"blocks={self.blocks},"
            f"max_blocks={self.max_blocks}")

@classmethod
def from_pb(
        cls,
        pb: generate_pb2.Batch,
        tokenizer: PreTrainedTokenizerBase,
        dtype: torch.dtype,
        device: torch.device
) -> "MindFlashCausalLMBatch":
    batch_tokenized_inputs = cls.batch_tokenized_inputs(pb.requests, tokenizer)
    return cls.from_tokenized(pb, tokenizer, batch_tokenized_inputs, dtype, device)

@classmethod
def from_tokenized(
        cls,
        pb: generate_pb2.Batch,
        tokenizer: PreTrainedTokenizerBase,
        batch_tokenized_inputs,
        dtype: torch.dtype,
        device: torch.device,
) -> "MindFlashCausalLMBatch":
    position_ids = []
    cu_seqlen_prefill = [0]
    needed_blocks_slots = []
    start_slots = []
    slot_indices = []

    input_lengths = []
    prefix_offsets = []
    read_offsets = []
    all_input_ids = []
    requests_idx_mapping = {}

    all_prefill_logprobs = True
    no_prefill_logprobs = True
    prefill_head_indices = []
    prefill_next_token_indices = []
    prefill_cu_outlens = [0]

    next_token_chooser_parameters = []
    stopping_criterias = []
    top_n_tokens = []

    # Cumulative length
    cumulative_length = 0
    cumulative_max_length = 0
    prefill_out_cumulative_length = 0

    blocks = 0
    max_seqlen = 0
    max_length = 0
    max_blocks = 0

    # Parse batch
    for i, (r, tokenized_input) in enumerate(
            zip(pb.requests, batch_tokenized_inputs)
    ):
        # request id -> idx in list mapping
        requests_idx_mapping[r.id] = i
```



```
        tokenized_input = tokenized_input[-r.truncate:]
        if (
            tokenized_input[0] == tokenizer.bos_token_id
            and tokenized_input[1] == tokenizer.bos_token_id
        ):
            tokenized_input = tokenized_input[1:]

        input_length = len(tokenized_input)
        input_lengths.append(input_length)

        prefix_offsets.append(input_length - 5)
        read_offsets.append(input_length)

        all_input_ids.append(tokenized_input)

        # Position ids
        request_position_ids = torch.arange(0, input_length, dtype=torch.int32)
        position_ids.append(request_position_ids)

        # Add cumulative lengths of all previous inputs
        cu_seqlen_prefill.append(cumulative_length + input_length)

        next_token_chooser_parameters.append(r.parameters)

        stopping_criteria = StoppingCriteria.from_pb(
            r.stopping_parameters, tokenizer
        )
        max_new_tokens = stopping_criteria.max_new_tokens
        stopping_criterias.append(stopping_criteria)
        top_n_tokens.append(r.top_n_tokens)

        # Paged attention
        # Remove one as the first token des not have a past
        speculative_length = get_speculate()
        speculative_length = 0 if speculative_length is None else speculative_length
        total_tokens = input_length + max_new_tokens - 1 + speculative_length
        needed_blocks = math.ceil(total_tokens / BLOCK_SIZE)
        blocks += needed_blocks
        needed_blocks_slots.append((needed_blocks, total_tokens))
        start_slots.append(cumulative_max_length)

        request_slot_indices = torch.arange(
            cumulative_max_length,
            cumulative_max_length + input_length,
            dtype=torch.int64,
        )
        slot_indices.append(request_slot_indices)

        all_prefill_logprobs = all_prefill_logprobs and r.prefill_logprobs
        no_prefill_logprobs = no_prefill_logprobs and not r.prefill_logprobs

        if r.prefill_logprobs:
            prefill_head_indices.append(request_position_ids + cumulative_length)
            prefill_next_token_indices.append(
                prefill_out_cumulative_length + input_length - 1
            )
            prefill_cu_outlens.append(prefill_out_cumulative_length + input_length)
            prefill_out_cumulative_length += input_length
        else:
            prefill_head_indices.append(
                torch.tensor(
                    [cumulative_length + input_length - 1], dtype=torch.int64
                )
            )
            prefill_next_token_indices.append(prefill_out_cumulative_length)
            prefill_cu_outlens.append(prefill_out_cumulative_length + 1)
            prefill_out_cumulative_length += 1

        # Update
```



```
        cumulative_length += input_length
        cumulative_max_length += total_tokens
        max_seqlen = max(max_seqlen, input_length)
        max_blocks = max(max_blocks, needed_blocks)
        max_length = max(
            max_length, input_length + max_new_tokens + speculative_length
        )

    next_token_chooser = MindIELLMHeterogeneousNextTokenChooser.from_pb(
        pb=next_token_chooser_parameters, dtype=dtype, device=device
    )

    start_slots = torch.tensor(start_slots, dtype=torch.int64)

    # Padded all_input_ids_tensor
    all_input_ids_tensor = np.zeros(
        (len(all_input_ids), max_length), dtype=np.int64
    )
    for i, input_ids in enumerate(all_input_ids):
        all_input_ids_tensor[i, : len(input_ids)] = input_ids

    all_input_ids_tensor = torch.tensor(
        all_input_ids_tensor, dtype=torch.int64, device=device
    )

    if len(pb.requests) > 1:
        input_ids = np.concatenate(all_input_ids, dtype=np.int64)
        position_ids = torch.cat(position_ids)
        slot_indices = torch.cat(slot_indices)
    else:
        input_ids = all_input_ids[0]
        position_ids = position_ids[0]
        slot_indices = slot_indices[0]

    cu_seqlen_prefill = torch.tensor(
        cu_seqlen_prefill, device=device, dtype=torch.int64
    )
    position_ids = position_ids.to(device)
    slot_indices = slot_indices.to(device)
    input_ids = torch.tensor(input_ids, dtype=torch.int64, device=device)
    input_lengths_tensor = torch.tensor(
        input_lengths, dtype=torch.int64, device=device
    )

    if all_prefill_logprobs:
        prefill_head_indices = None
        prefill_next_token_indices = cu_seqlen_prefill[1:] - 1
    elif no_prefill_logprobs:
        prefill_head_indices = cu_seqlen_prefill[1:] - 1
        prefill_next_token_indices = None
    else:
        prefill_head_indices = torch.tensor(
            torch.cat(prefill_head_indices), dtype=torch.int64, device=device
        )
        prefill_next_token_indices = torch.tensor(
            prefill_next_token_indices, dtype=torch.int64, device=device
        )
    top_n_tokens_tensor = torch.tensor(
        top_n_tokens, device=device, dtype=torch.int64
    )

    return cls(
        batch_id=pb.id,
        requests=pb.requests,
        requests_idx_mapping=requests_idx_mapping,
        input_ids=input_ids,
        position_ids=position_ids,
        cu_seqlen_prefill=cu_seqlen_prefill,
        start_slots=start_slots,
```



```
            slot_indices=slot_indices,
            needed_blocks_slots=needed_blocks_slots,
            block_tables=None,
            block_tables_tensor=None,
            slots=None,
            max_seqlen=max_seqlen,
            prefill_head_indices=prefill_head_indices,
            prefill_next_token_indices=prefill_next_token_indices,
            prefill_cu_outlens=prefill_cu_outlens,
            input_lengths=input_lengths,
            input_lengths_tensor=input_lengths_tensor,
            prefix_offsets=prefix_offsets,
            read_offsets=read_offsets,
            all_input_ids=all_input_ids,
            all_input_ids_tensor=all_input_ids_tensor,
            next_token_chooser=next_token_chooser,
            stopping_criterias=stopping_criterias,
            top_n_tokens=top_n_tokens,
            top_n_tokens_tensor=top_n_tokens_tensor,
            blocks=blocks,
            max_blocks=max_blocks,
            speculative_ids=None,
        )

    @classmethod
    @tracer.start_as_current_span("concatenate")
    def concatenate(cls, batches: List["MindFlashCausalLMBatch"]) -> "MindFlashCausalLMBatch":
        # Batch attributes
        requests = []
        requests_idx_mapping = {}

        blocks = 0
        total_batch_size = 0
        total_slots = 0
        max_blocks = 0
        max_length = 0
        max_seqlen = 0
        for b in batches:
            total_batch_size += len(b)
            total_slots += len(b.slots)
            blocks += b.blocks
            speculative_length = (
                b.speculative_ids.shape[1] if b.speculative_ids is not None else 0
            )
            max_blocks = max(max_blocks, b.max_blocks)
            max_seqlen = max(max_seqlen, b.max_seqlen)
            max_length = max(
                max_length,
                max(
                    input_length
                    + stopping_criteria.max_new_tokens
                    + speculative_length
                    - stopping_criteria.current_tokens
                    for input_length, stopping_criteria in zip(
                        b.input_lengths, b.stopping_criterias
                    )
                ),
            )

        input_ids = batches[0].input_ids.new_empty(total_batch_size)
        position_ids = batches[0].position_ids.new_empty(total_batch_size)
        slots = batches[0].slots.new_empty(total_slots)
        slot_indices = batches[0].slot_indices.new_empty(total_batch_size)
        input_lengths_tensor = batches[0].input_lengths_tensor.new_empty(
            total_batch_size
        )
        block_tables_tensor = batches[0].block_tables_tensor.new_zeros(
            (total_batch_size, max_blocks)
        )
```



```
all_input_ids_tensor = batches[0].all_input_ids_tensor.new_zeros(
    (total_batch_size, max_length)
)
top_n_tokens_tensor = batches[0].top_n_tokens_tensor.new_zeros(
    total_batch_size,
)

start_slots = []
block_tables = []
all_input_ids = []

input_lengths = []
prefix_offsets = []
read_offsets = []

next_token_chooser_parameters = []
fsm_grammar_states = []
stopping_criterias = []
top_n_tokens = []

# Cumulative length
cumulative_batch_size = 0
cumulative_slots = 0

for i, batch in enumerate(batches):
    requests.extend(batch.requests)

    if i == 0:
        requests_idx_mapping = batch.requests_idx_mapping
    else:
        # We need to offset the mapping for each batch by the cumulative batch size
        for k, v in batch.requests_idx_mapping.items():
            requests_idx_mapping[k] = v + cumulative_batch_size

    start_index = cumulative_batch_size
    end_index = cumulative_batch_size + len(batch)
    slots_start_index = cumulative_slots
    slots_end_index = cumulative_slots + len(batch.slots)

    # Copy tensors (GPU)
    input_ids[start_index:end_index] = batch.input_ids
    position_ids[start_index:end_index] = batch.position_ids
    slot_indices[start_index:end_index] = batch.slot_indices + cumulative_slots
    input_lengths_tensor[start_index:end_index] = batch.input_lengths_tensor
    top_n_tokens_tensor[start_index:end_index] = batch.top_n_tokens_tensor
    slots[slots_start_index:slots_end_index] = batch.slots

    all_input_ids_tensor[
    start_index:end_index, : batch.all_input_ids_tensor.shape[1]
    ] = batch.all_input_ids_tensor[:, :max_length]

    block_tables_tensor[
    start_index:end_index, : batch.block_tables_tensor.shape[1]
    ] = batch.block_tables_tensor[:, :max_blocks]

    start_slots.append(batch.start_slots + cumulative_slots)

    block_tables.extend(batch.block_tables)
    all_input_ids.extend(batch.all_input_ids)

    input_lengths.extend(batch.input_lengths)
    prefix_offsets.extend(batch.prefix_offsets)
    read_offsets.extend(batch.read_offsets)

    next_token_chooser_parameters.extend([r.parameters for r in batch.requests])
    fsm_grammar_states.extend(batch.next_token_chooser.fsm_grammar_states)
    stopping_criterias.extend(batch.stopping_criterias)
```



```
            top_n_tokens.extend(batch.top_n_tokens)

            # Update
            cumulative_batch_size += len(batch)
            cumulative_slots += len(batch.slots)

        start_slots = torch.concat(start_slots)

        next_token_chooser = MindIELLMHeterogeneousNextTokenChooser.from_pb(
            pb=next_token_chooser_parameters,
            dtype=batches[0].next_token_chooser.dtype,
            device=batches[0].next_token_chooser.device
        )

        speculative_ids = (
            torch.cat([b.speculative_ids for b in batches], dim=0)
            if batches[0].speculative_ids is not None
            else None
        )

        # Needed to avoid dropping blocks when the batches will go out of scope
        for b in batches:
            b.block_tables = None
            del b

        return cls(
            batch_id=batches[0].batch_id,
            requests=requests,
            requests_idx_mapping=requests_idx_mapping,
            input_ids=input_ids,
            position_ids=position_ids,
            cu_seqlen_prefill=None,
            start_slots=start_slots,
            slot_indices=slot_indices,
            needed_blocks_slots=None,
            block_tables=block_tables,
            block_tables_tensor=block_tables_tensor,
            slots=slots,
            max_seqlen=max_seqlen,
            prefill_head_indices=None,
            prefill_next_token_indices=None,
            prefill_cu_outlens=None,
            input_lengths=input_lengths,
            input_lengths_tensor=input_lengths_tensor,
            prefix_offsets=prefix_offsets,
            read_offsets=read_offsets,
            all_input_ids=all_input_ids,
            all_input_ids_tensor=all_input_ids_tensor,
            next_token_chooser=next_token_chooser,
            stopping_criterias=stopping_criterias,
            top_n_tokens=top_n_tokens,
            top_n_tokens_tensor=top_n_tokens_tensor,
            blocks=blocks,
            max_blocks=max_blocks,
            speculative_ids=speculative_ids,
        )

    def to_pb(self) -> generate_pb2.CachedBatch:
        return generate_pb2.CachedBatch(
            id=self.batch_id,
            request_ids=[r.id for r in self.requests],
            size=len(self),
            max_tokens=self.blocks * BLOCK_SIZE,
        )

class MindModel(FlashCausalLM):
    def __init__(
            self,
            model_id: str
```



```
):
    logger.warning("Initialize mindie-llm model.")
    rank = int(os.getenv("RANK", "0"))
    world_size = int(os.getenv("WORLD_SIZE", "1"))
    model_config = {
        'backend_type': BackendType.ATB,
        'rank': rank,
        'world_size': world_size,
        'model_id': model_id,
        'num_threads': 8,
        'local_rank': rank,
        'npu_device_id': rank
    }
    self.model_runner = GeneratorTorch(model_config)
    super(MindModel, self).__init__(
        model=self.model_runner.model_wrapper.model_runner.model,
        tokenizer=self.model_runner.tokenizer,
        num_layers=self.model_runner.model_info.num_layers,
        num_kv_heads=self.model_runner.model_info.num_kv_heads,
        head_size=self.model_runner.model_info.head_size,
        dtype=self.model_runner.model_info.dtype,
        device=self.model_runner.model_info.device,
        rank=self.model_runner.rank,
        world_size=self.model_runner.world_size,
    )
    logger.warning("MindModel from tgi_npu initialized.")

def __del__(self):
    del self.model_runner.model_wrapper

@property
def batch_type(self) -> Type[MindFlashCausalLMBatch]:
    return MindFlashCausalLMBatch

def forward(
        self, batch: MindFlashCausalLMBatch
) -> Tuple[torch.Tensor, Optional[torch.Tensor]]:
    """Assume return logits, speculative_logits"""
    input_ids = batch.input_ids
    position_ids = batch.position_ids
    cu_seqlen_prefill = batch.cu_seqlen_prefill
    kv_cache = get_cache_manager().kv_cache
    block_tables = batch.block_tables_tensor
    slots = batch.slots[batch.slot_indices]
    input_lengths = batch.input_lengths_tensor
    max_s = batch.max_seqlen
    lm_head_indices = batch.prefill_head_indices
    speculative_logits = None

    return self.model_runner.forward_tensor(
        input_ids=input_ids,
        position_ids=position_ids,
        is_prefill=cu_seqlen_prefill is not None,
        kv_cache=kv_cache,
        block_tables=block_tables,
        slots=slots,
        input_lengths=input_lengths,
        max_seq_len=max_s,
        lm_head_indices=lm_head_indices,
    ), speculative_logits

@tracer.start_as_current_span("generate_token")
def generate_token(
        self, batch: MindFlashCausalLMBatch
) -> Tuple[List[Generation], Optional[MindFlashCausalLMBatch], Tuple[int, int]]:
    start = time.time_ns()
    prefill = batch.cu_seqlen_prefill is not None
    prefill_logprobs = batch.prefill_next_token_indices is not None
    # check if need slots
```



```
if batch.needed_blocks_slots:
    # Allocate blocks to this batch
    block_tables, block_tables_tensor, slots = get_cache_manager().allocate(
        batch.needed_blocks_slots,
        batch.blocks,
        batch.max_blocks,
        batch.input_ids.device,
    )
    batch.needed_blocks_slots = None
    batch.block_tables = block_tables
    batch.block_tables_tensor = block_tables_tensor
    batch.slots = slots

try:
    out, speculative_logits = self.forward(batch)
except Exception as e:
    del batch
    raise e

if prefill:
    next_token_logits = (
        out[batch.prefill_next_token_indices] if prefill_logprobs else out
    )
    if speculative_logits is not None:
        speculative_logits = (
            speculative_logits[batch.prefill_next_token_indices]
            if prefill_logprobs
            else speculative_logits
        )
else:
    logger.debug(f"Decode batch size {batch.input_ids.shape[0]}")
    next_token_logits = out

speculate = get_speculate()

request_ids = [req.id for req in batch.requests]
(
    next_input_ids,
    next_token_logprobs,
    logprobs,
    accepted_ids,
    speculative_ids,
) = batch.next_token_chooser(
    request_ids,
    prefill,
    batch.all_input_ids_tensor[:, : batch.max_seqlen],
    next_token_logits,
    speculate
)

batch_top_token_ids, batch_top_token_logprobs = batch_top_tokens(
    batch.top_n_tokens, batch.top_n_tokens_tensor, logprobs, accepted_ids
)

if prefill:
    if len(batch) > 1 and prefill_logprobs:
        # We create the prefill_tokens_indices tensor that will be used to gather prefill logprobs
        # When batch == 1, we will just use the batch.input_ids values directly
        prefill_tokens_indices = batch.input_ids.new_zeros(len(out))

    next_position_ids = batch.position_ids.new_empty(len(batch))
    batch.slot_indices = batch.slot_indices[batch.cu_seqlen_prefill[1:] - 1]
    # We do not need cu_seqlen_prefill anymore
    batch.cu_seqlen_prefill = None
else:
    prefill_logprobs = None
    next_position_ids = batch.position_ids

# Cumulative length
```



```
        cumulative_length = 0

        # Results
        generations: List[Generation] = []
        stopped = True

        # Zipped iterator
        iterator = zip(batch.input_lengths, batch.all_input_ids, accepted_ids)

        # We do two for loops as the first one can run completely asynchronously from the GPU while
for the second
        # one, we need to first do a GPU <-> CPU sync
        # It is faster if we delay this sync for the maximum amount of time

        # For each member of the batch
        index = 0
        for i, (input_length, all_input_ids, n_accepted_ids) in enumerate(iterator):
            # Indexing metadata
            start_index = cumulative_length
            end_index = cumulative_length + input_length

            if prefill:
                # Indexing metadata
                out_start_index = batch.prefill_cu_outlens[i]
                out_end_index = batch.prefill_cu_outlens[i + 1]
                out_length = out_end_index - out_start_index

                # Initialize position_ids
                # In decode, we do not need this as we can just increment position ids
                next_position_ids[i] = batch.position_ids[end_index - 1]

                # Used to gather prefill logprobs
                # Copy batch.input_ids to prefill_token_indices
                if prefill_logprobs:
                    if len(batch) > 1:
                        prefill_tokens_indices[out_start_index: out_end_index - 1] = (
                            batch.input_ids[start_index + 1: start_index + out_length]
                        )
                    else:
                        # Set prefill_tokens_indices to the correct slice
                        prefill_tokens_indices = batch.input_ids[
                                                 start_index + 1: start_index + out_length
                                                 ]

            for _ in range(n_accepted_ids):
                index += 1

            cumulative_length += input_length

        batch.all_input_ids_tensor.scatter_(1,
batch.input_lengths_tensor.view(batch.input_lengths_tensor.shape[0], 1),
                                            next_input_ids.view(next_input_ids.shape[0], 1))
        # Update values
        batch.input_ids = next_input_ids[accepted_ids.cumsum(dim=-1) - 1]
        batch.speculative_ids = speculative_ids
        batch.position_ids = next_position_ids + accepted_ids
        batch.input_lengths_tensor += accepted_ids
        batch.slot_indices += accepted_ids

        if prefill and prefill_logprobs:
            # Get prefill logprobs
            prefill_logprobs_tensor = torch.log_softmax(out, -1)
            prefill_logprobs = torch.gather(
                prefill_logprobs_tensor, 1, prefill_tokens_indices.view(-1, 1)
            )
            # GPU <-> CPU sync
            prefill_logprobs = prefill_logprobs.view(-1).tolist()

        # GPU <-> CPU sync
```



```
        next_token_logprobs = next_token_logprobs.tolist()
        next_token_ids = next_input_ids.tolist()
        accepted_ids = accepted_ids.tolist()
        start_decode = time.time_ns()

        # Zipped iterator
        iterator = zip(
            batch.requests,
            batch.input_lengths,
            batch.prefix_offsets,
            batch.read_offsets,
            batch.stopping_criterias,
            batch.all_input_ids,
            batch.next_token_chooser.do_sample,
            batch.next_token_chooser.seeds,
            batch.top_n_tokens,
            accepted_ids,
            batch_top_token_ids,
            batch_top_token_logprobs,
        )

        # For each member of the batch
        index = 0
        for i, (
                request,
                input_length,
                prefix_offset,
                read_offset,
                stopping_criteria,
                all_input_ids,
                do_sample,
                seed,
                top_n_tokens,
                n_accepted_ids,
                top_token_ids,
                top_token_logprobs,
        ) in enumerate(iterator):
            # Append next token to all tokens
            next_token_texts = []
            left = 0

            if n_accepted_ids > 1:
                if RANK == 0:
                    logger.debug(f"Speculated ids {n_accepted_ids - 1}")

            current_stopped = False
            for j in range(index, index + n_accepted_ids):
                # Generated token
                next_token_id = next_token_ids[j]
                all_input_ids.append(next_token_id)
                # Generated token
                next_token_text, prefix_offset, read_offset = self.decode_token(
                    all_input_ids,
                    prefix_offset,
                    read_offset,
                )

                next_token_texts.append(next_token_text)

                stop, reason = stopping_criteria(
                    next_token_id,
                    next_token_text,
                )

                if stop:
                    left = index + n_accepted_ids - j - 1
                    current_stopped = True
                    break
                else:
```



```
            current_stopped = False
        stopped = stopped and current_stopped

        _next_token_ids = next_token_ids[index: index + n_accepted_ids - left]
        _next_token_logprobs = next_token_logprobs[
                            index: index + n_accepted_ids - left
                            ]
        index += n_accepted_ids

        # Shard generations
        # All generations will be appended in the rust sharded client
        if i % self.world_size == self.rank:
            if stop:
                # Decode generated tokens
                output_text, _, _ = self.decode_token(
                    all_input_ids,
                    prefix_offset=len(all_input_ids)
                                - stopping_criteria.current_tokens
                                - 1,
                    read_offset=len(all_input_ids)
                                - stopping_criteria.current_tokens,
                    skip_special_tokens=True,
                )
                generated_text = GeneratedText(
                    output_text,
                    stopping_criteria.current_tokens,
                    reason,
                    seed if do_sample else None,
                )
            else:
                generated_text = None

            # Prefill
            if prefill and request.prefill_logprobs:
                out_start_index = batch.prefill_cu_outlens[i]
                out_end_index = batch.prefill_cu_outlens[i + 1]

                # Remove generated token to only have prefill and add nan for first prompt token
                request_prefill_logprobs = [float("nan")] + prefill_logprobs[
                                                    out_start_index: out_end_index - 1
                                                    ]
                prefill_token_ids = all_input_ids[:-1]
                prefill_texts = self.tokenizer.batch_decode(
                    prefill_token_ids,
                    clean_up_tokenization_spaces=False,
                    skip_special_tokens=False,
                )

                prefill_tokens = Tokens(
                    prefill_token_ids,
                    request_prefill_logprobs,
                    prefill_texts,
                    is_special=[],
                )
            else:
                prefill_tokens = None

            if top_n_tokens > 0:
                all_top_tokens = []
                for top_token_ids, top_token_logprobs in zip(
                        top_token_ids, top_token_logprobs
                ):
                    toptoken_texts = self.tokenizer.batch_decode(
                        top_token_ids,
                        clean_up_tokenization_spaces=False,
                        skip_special_tokens=False,
                    )
                    special_toptokens = [
                        token_id in self.all_special_ids
```



```
                        for token_id in top_token_ids
                    ]
                    top_tokens = Tokens(
                        top_token_ids,
                        top_token_logprobs,
                        toptoken_texts,
                        special_toptokens,
                    )
                    all_top_tokens.append(top_tokens)
                top_tokens = all_top_tokens
            else:
                top_tokens = None

            generation = Generation(
                request.id,
                prefill_tokens,
                Tokens(
                    _next_token_ids,
                    _next_token_logprobs,
                    next_token_texts,
                    [nid in self.all_special_ids for nid in _next_token_ids],
                ),
                generated_text,
                top_tokens,
            )

            generations.append(generation)

        # Update values
        batch.input_lengths[i] = input_length + n_accepted_ids
        if batch.input_lengths[i] > batch.max_seqlen:
            batch.max_seqlen = batch.input_lengths[i]
        batch.prefix_offsets[i] = prefix_offset
        batch.read_offsets[i] = read_offset
        batch.all_input_ids[i] = all_input_ids

    if stopped:
        del batch
        # No need to return a batch if we know that all requests stopped
        forward_ns = start_decode - start
        decode_ns = time.time_ns() - start_decode
        none_batch = None
        return generations, none_batch, (forward_ns, decode_ns)

    batch.prefill_cu_outlens = None
    batch.prefill_head_indices = None
    batch.prefill_next_token_indices = None

    forward_ns = start_decode - start
    decode_ns = time.time_ns() - start_decode
    return generations, batch, (forward_ns, decode_ns)

def warmup(self, batch: MindFlashCausalLMBatch):
    # The warmup batch is the biggest batch we could ever receive
    torch.npu.empty_cache()

    peak_memory = torch_npu.npu.max_memory_allocated()
    logger.info(f">>>>before warmup peak_memory {peak_memory}")
    try:
        cache_manager = set_cache_manager(
            batch.blocks,
            self.num_layers,
            self.num_kv_heads,
            self.head_size,
            False,
            self.dtype,
            self.device,
        )
        _, batch, _ = self.generate_token(batch)
```



```
        except torch.cuda.OutOfMemoryError as e:
            raise RuntimeError(
                f"Not enough memory to handle {len(batch.input_ids)} prefill tokens. "
                f"You need to decrease `--max-batch-prefill-tokens`"
            ) from e

        # Inspired by the original implementation in [vllm](https://github.com/vllm-project/vllm)
        # Calculate the number of blocks that can be allocated with the free memory
        dtype_size = torch.tensor([], dtype=self.dtype).element_size()
        cache_block_size = BLOCK_SIZE * self.num_kv_heads * self.head_size
        total_cache_size = self.num_layers * cache_block_size * 2 * dtype_size
        torch_npu.npu.synchronize()

        total_gpu_memory = torch_npu.npu.get_device_properties(self.device).total_memory

        peak_memory = torch_npu.npu.max_memory_allocated()
        logger.info(
            f">>>>dtype_size {dtype_size}, cache_block_size {cache_block_size}, num_kv_heads
{self.num_kv_heads}, "
            f"total_cache_size {total_cache_size}, peak_memory {peak_memory}")
        total_free_memory = total_gpu_memory - peak_memory
        logger.info(f">>>>total_free_memory {total_free_memory}, total_gpu_memory
{total_gpu_memory}, "
                    f"MEMORY_FRACTION {MEMORY_FRACTION}")
        free_memory = max(0, total_free_memory - (1 - MEMORY_FRACTION) * total_gpu_memory)

        num_blocks = (
            # Leave 5% for some wiggle room
            int((free_memory * 0.95) // total_cache_size)
            # Add batch.blocks as we allocated it above, so it is included in the peak memory.
            + cache_manager.num_blocks
        )

        del batch
        del cache_manager

        real_manager = set_cache_manager(
            num_blocks,
            self.num_layers,
            self.num_kv_heads,
            self.head_size,
            self.sliding_window is not None,
            self.dtype,
            self.device,
        )
        tgi_cache_manager.CACHE_MANAGER = real_manager
        logger.warning(f">>>>real CacheManger {get_cache_manager()}")
        peak_memory = torch_npu.npu.max_memory_allocated()
        logger.warning(f">>>>end warmup peak_memory {peak_memory}")
        logger.warning(f"Warmup return {int(num_blocks * BLOCK_SIZE)}")
        return int(num_blocks * BLOCK_SIZE)
```

- Tgi-MindIE/tgi_npu/token_mindie.py

```
# Part of codes in this file was copied from project[huggingface][text-generation-inference]

import os
from typing import List, Optional
from loguru import logger
import numpy as np
import torch
from text_generation_server.pb import generate_pb2

from mindie_llm.text_generator.utils.sampling_metadata import SamplingData, SamplingParam
from mindie_llm.text_generator.utils.config import SamplerConfig
from mindie_llm.text_generator.samplers.sampler import Sampler

from mindie_llm.modeling.backend_type import BackendType

WRAPPER_KEY = "tensor_wrapper"
try:
```



```
    from mindie_llm.text_generator.utils.sampling_metadata import TensorWrapper
except ImportError:
    class TensorWrapper:
        def __init__(self, backend, device):
            self.device = device
            self.backend = backend

        def __call__(self, data):
            if data.dtype == np.int32:
                dtype = torch.int32
            elif data.dtype == np.bool_:
                dtype = torch.bool
            else:
                dtype = None
            return torch.tensor(data, dtype=dtype, device=self.device)


    WRAPPER_KEY = "to_tensor"


def do_filter(sample_param: List, indices):
    if any(sample_param):
        return [sample_param[i] for i in indices]
    return sample_param


class MindIELLMHeterogeneousNextTokenChooser:
    def __init__(
            self,
            dtype: torch.dtype,
            device: torch.device,
            watermark: List[bool],
            temperature: List[float],
            repetition_penalty: List[float],
            frequency_penalty: List[float],
            top_k: List[int],
            top_p: List[float],
            typical_p: List[float],
            do_sample: List[bool],
            seeds: List[int],
            grammars: List[str],
            grammar_types: List[int],
            fsm_grammar_states: List[int],
            sample_method,  # mindie-llm sampling method
    ):
        if any(watermark):
            logger.warning(f"Watermark not supported now in mindie-llm")
        if any([x < 1.0 for x in typical_p]):
            logger.warning(f"Typical_p not supported now in mindie-llm")
        if any(grammar_types) or any(grammars) or any(fsm_grammar_states):
            logger.warning(f"Grammar not supported now in mindie-llm")

        self.tensor_wrapper = TensorWrapper(BackendType.ATB, device)
        self.wrapper_dict = {WRAPPER_KEY: TensorWrapper(BackendType.ATB, device)}
        self.sample_params = SamplingParam.from_numpy(
            repetition_penalty=np.array(repetition_penalty, dtype=np.float16),
            presence_penalty=None,
            frequency_penalty=np.array(frequency_penalty, dtype=np.float16),
            temperature=np.array(temperature, dtype=np.float16),
            top_k=np.array(top_k),
            top_p=np.array(top_p),
            seed=np.array(seeds).astype(np.int32),
            do_sample=np.array(do_sample),
            **self.wrapper_dict
        )

        # Temp store for filter
        self.temperature = temperature
        self.repetition_penalty = repetition_penalty
```



```
        self.frequency_penalty = frequency_penalty
        self.top_k = top_k
        self.top_p = top_p
        self.seeds = seeds
        self.do_sample = do_sample

        self.choice = sample_method
        self.dtype = dtype
        self.device = device
        self.seeds = seeds
        self.do_sample = self.sample_params.do_sample_meta.do_sample_array
        self.fsm_grammar_states = fsm_grammar_states

    def __call__(self,
                 request_ids: List,
                 is_prefill: bool,
                 input_ids: torch.Tensor,
                 scores: torch.Tensor,
                 speculate: int
                 ):
        batch_size = scores.shape[0]
        speculate_size = 1
        scores = scores.view(batch_size, speculate_size, -1)

        # Don't use SamplingData。from_numpy to avoid tensor.cpu() to transfer large data
        input_ids_int32 = input_ids.to(torch.int32)
        sample_data = SamplingData(all_input_ids=input_ids_int32, output_ids=input_ids_int32,
is_prefill=is_prefill,
                                   request_ids=np.array(request_ids))
        next_ids = torch.zeros((batch_size, speculate_size), device=scores.device, dtype=torch.long)
        for j in range(speculate_size):
            _scores = scores[:, j]

            batch_logits, _next_ids = self.choice(batch_logits=_scores, batch_sampling_data=sample_data,
                                                  batch_sampling_params=self.sample_params)
            scores[:, j] = _scores
            next_ids[:, j] = torch.from_numpy(_next_ids)
        next_ids = next_ids.view(batch_size * speculate_size)
        allscores = scores.view(batch_size * speculate_size, -1)
        alllogprobs = torch.log_softmax(allscores, -1)

        accepted_ids = torch.ones_like(next_ids)
        logprobs = alllogprobs

        next_logprobs = torch.gather(logprobs, 1, next_ids.view(-1, 1)).view(-1)

        speculative_ids = None
        return next_ids, next_logprobs, alllogprobs, accepted_ids, speculative_ids

    @classmethod
    def from_pb(
            cls,
            pb: List[generate_pb2.NextTokenChooserParameters],
            dtype: torch.dtype,
            device: torch.device,
            fsm_grammar_states: Optional[List[int]] = None,
    ) -> "MindIELLMHeterogeneousNextTokenChooser":
        curr_rank = int(os.getenv("RANK", "0"))
        sample_method = Sampler(SamplerConfig(rank=curr_rank, backend_type=BackendType.ATB,
npu_id=curr_rank))
        return MindIELLMHeterogeneousNextTokenChooser(
            watermark=[pb_.watermark for pb_ in pb],
            temperature=[pb_.temperature for pb_ in pb],
            repetition_penalty=[pb_.repetition_penalty for pb_ in pb],
            frequency_penalty=[pb_.frequency_penalty for pb_ in pb],
            top_k=[pb_.top_k for pb_ in pb],
            top_p=[pb_.top_p for pb_ in pb],
            typical_p=[pb_.typical_p for pb_ in pb],
            do_sample=[pb_.do_sample for pb_ in pb],
```



```
            seeds=[pb_.seed for pb_ in pb],
            device=device,
            dtype=dtype,
            grammars=[pb_.grammar for pb_ in pb],
            grammar_types=[pb_.grammar_type for pb_ in pb],
            fsm_grammar_states=(
                fsm_grammar_states if fsm_grammar_states else [0] * len(pb)
            ),
            sample_method=sample_method
        )

    def filter(self, indices):
        self.repetition_penalty = do_filter(self.repetition_penalty, indices)
        self.frequency_penalty = do_filter(self.frequency_penalty, indices)
        self.temperature = do_filter(self.temperature, indices)
        self.top_k = do_filter(self.top_k, indices)
        self.top_p = do_filter(self.top_p, indices)
        self.seeds = do_filter(self.seeds, indices)
        self.do_sample = do_filter(self.do_sample, indices)

        self.sample_params = SamplingParam.from_numpy(
            repetition_penalty=np.array(self.repetition_penalty, dtype=np.float16),
            presence_penalty=None,
            frequency_penalty=np.array(self.frequency_penalty, dtype=np.float16),
            temperature=np.array(self.temperature, dtype=np.float16),
            top_k=np.array(self.top_k),
            top_p=np.array(self.top_p),
            seed=np.array(self.seeds).astype(np.int32),
            do_sample=np.array(self.do_sample),
            **self.wrapper_dict
        )
        return self
```

- Tgi-MindIE/tgi_npu/vlm_mind_models.py

```
import re
from typing import List, Type, Dict
from dataclasses import dataclass
import torch
from loguru import logger
from atb_llm.runner.tokenizer_wrapper import TokenizerWrapper
from text_generation_server.pb import generate_pb2
from transformers import PreTrainedTokenizerBase
from tgi_npu.mind_models import MindFlashCausalLMBatch, MindModel
IMAGES = re.compile(r'!\[[^\]]*\]\((.*?)\s*(\"(?:.*[^\"])\")?\s*\)')
def split(string) -> List[Dict[str, str]]:
    parts = []
    cursor = 0
    for pattern in IMAGES.finditer(string):
        start = pattern.start()
        if start != cursor:
            parts.append({"text": string[cursor:start]})
        parts.append({"image": pattern.group(1)})
        cursor = pattern.end()
    if cursor != len(string):
        parts.append({"text": string[cursor:]})
    return parts
@dataclass
class VlmMindFlashCausalLMBatch(MindFlashCausalLMBatch):
    @classmethod
    def batch_tokenized_inputs(cls, requests, tokenize):
        inputs = []
        for r in requests:
            splits = split(r.inputs)
            single_input = tokenize(splits).tolist()
            inputs.append(single_input)
        return inputs
    @classmethod
    def from_pb_processor(
            cls,
            pb: generate_pb2.Batch,
```

```
            tokenizer: PreTrainedTokenizerBase,
            tokenize,
            dtype: torch.dtype,
            device: torch.device,
    ) -> "VlmMindFlashCausalLMBatch":
        batch_tokenized_inputs = cls.batch_tokenized_inputs(pb.requests, tokenize)
        batch = cls.from_tokenized(pb, tokenizer, batch_tokenized_inputs, dtype, device)
        return batch
class VlmMindModel(MindModel):
    def __init__(
            self,
            model_id: str
    ):
        logger.warning("Initialize mindie-llm model for vlm.")
        super(VlmMindModel, self).__init__(model_id)
        self.tokenize = TokenizerWrapper(model_id).tokenize
        logger.warning("VlmMindModel from tgi_npu initialized.")
    @property
    def batch_type(self) -> Type[VlmMindFlashCausalLMBatch]:
        return VlmMindFlashCausalLMBatch
```

- Tgi-MindIE/pyproject.toml

```
[tool.poetry]
name = "tgi-npu"
version = "0.1.0"
description = "NPU MindIE Adapter for TGI v2.0.4"
authors = ["Your Name <you@example.com>"]
readme = "README.md"
exclude = ["router", "cover"]



[tool.poetry.dependencies]
python = "^3.10"

[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"
```

7.3.3.2 TGI v0.9.4 版本参考适配代码

文件目录与TGI v2.0.4基本一致，由于支持的特性不同，部分文件有差异：

```
Tgi-MindIE
 |_____cover
       |_____models
             |________init__.py
 |_____tgi_npu
       |_____init__.py
       |____cache_manager.py
       |____info.py
       |____mind_model.py
       |____token_mindie.py
|_____pyproject.toml
|_____install.sh
|_____README.m
```

各源文件的含义和作用如下表所示：

源文件	含义及作用
cover/models/__init__.py	替换原仓中server/text_generation_server/models/__init__.py文件，将推理模型引导至MindIE LLM。
tgi_npu/__init__.py	针对NPU硬件环境进行必要的初始化。

源文件	含义及作用
tgi_npu/cache_manager.py	KV Cache管理器，主要针对NPU进行KV Cache初始化。
tgi_npu/info.py	NPU信息。
tgi_npu/mind_model.py	定义了推理模型入口类MindModel以及对应的数据通信格式MindFlashCasualLMBatch，分别继承自原仓的FlashCasualLM以及FlashCasualLMBatch。在MindModel中，generate_token方法沿用了原版大部分代码，并结合MindIE LLM调用过程进行了修改。其中，Forward方法改为调用MindIE LLM提供的forward_tensor方法。warmup 结合NPU访存特点进行修改。
tgi_npu/token_mindie.py	后采样代码。
pyproject.toml	适配安装包配置文件。

样例代码：

- Tgi-MindIE/install.sh

```
#!/usr/bin/env bash
# install-origin
if [ -d "./tgi_origin" ]; then
    echo "./tgi_origin directory has already exist!"
    exit 1
fi

git clone -b v0.9.4 https://github.com/huggingface/text-generation-inference.git tgi_origin

cp cover/cli.py tgi_origin/server/text_generation_server/
cp cover/models/__init__.py tgi_origin/server/text_generation_server/models
cp cover/utils/__init__.py tgi_origin/server/text_generation_server/utils
cp cover/utils/hub.py tgi_origin/server/text_generation_server/utils
cp cover/utils/import_utils.py tgi_origin/server/text_generation_server/utils
cp cover/utils/peft.py tgi_origin/server/text_generation_server/utils

sed -i "s/requires_padding, 16/requires_padding, 128/g" tgi_origin/router/src/infer.rs
sed -i "s/prefill_logprobs: true/prefill_logprobs: false/g" tgi_origin/router/client/src/client.rs
sed -i "s/Vec::from(encoding.get_ids())/encoding.get_ids()/g" tgi_origin/router/src/validation.rs
sed -i "s/bnb, accelerate/accelerate/g" tgi_origin/server/Makefile

cd tgi_origin && make install-server && make install-router && make install-launcher

cd .. && pip install -e .
pip install -r requirements.txt
```

- Tgi-MindIE/cover/models/__init__.py

```
# This file was copied from project[huggingface][text-generation-inference]
from typing import Optional

import torch
from loguru import logger

from text_generation_server.models.model import Model
```



```
# The flag below controls whether to allow TF32 on matmul. This flag defaults to False
# in PyTorch 1.12 and later.
torch.backends.cuda.matmul.allow_tf32 = True

# The flag below controls whether to allow TF32 on cuDNN. This flag defaults to True.
torch.backends.cudnn.allow_tf32 = True

# Disable gradients
torch.set_grad_enabled(False)

__all__ = [
    "Model",
    "get_model",
]


def get_model(
    model_id: str,
    revision: Optional[str],
    sharded: bool,
    quantize: Optional[str],
    dtype: Optional[str],
    trust_remote_code: bool,
) -> Model:
    if dtype is None:
        dtype = torch.float16
    elif dtype == "float16":
        dtype = torch.float16
    elif dtype == "bfloat16":
        dtype = torch.bfloat16
    else:
        raise RuntimeError(f"Unknown dtype {dtype}")

    try:
        import torch_npu
        from tgi_npu import MindModel
        npu_module_imported = True
    except (ImportError, NotImplementedError) as excp:
        npu_module_imported = False
        logger.error(f"Error catched: {str(excp)}")

    if npu_module_imported and torch.npu.is_available():
        return MindModel(model_id)
    else:
        logger.error("NPU enviroment error!!!!!!!!!")
        raise ValueError(f"NPU enviroment error!!!!!!!!!")
```

- Tgi-MindIE/tgi_npu/__init__.py

```
#!/usr/bin/env python3
# coding=utf-8
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.


import torch
import torch_npu
from loguru import logger

from tgi_npu.mind_models import MindModel


def init():
    torch._C._InferenceMode(True)
    soc_version = torch_npu._C._npu_get_soc_version()
    if soc_version not in [104, 220, 221, 222, 223, 224]:
        logger.info("Some op does not support for this soc!")
        option = {"NPU_FUZZY_COMPILE_BLACKLIST": "ReduceNansum"}
    else:
        option = {"NPU_FUZZY_COMPILE_BLACKLIST": "GatherElements"}
    try:
```



```
        torch.npu.set_option(option)
        logger.warning("Finish init for NPU device!")
    except Exception as e:
        logger.error(f"Failed to init for NPU device: {e}!")


init()
```

- Tgi-MindIE/tgi_npu/cache_manager.py

```
# This file was copied from project[huggingface][text-generation-inference]
import math
from typing import Optional, List, Tuple
import gc
import torch
from tgi_npu.info import NPUSocInfo

BLOCK_SIZE: int = 128
# Will be set in warmup
CACHE_MANAGER: Optional["CacheManager"] = None


class CacheManager:
    def __init__(
            self,
            num_blocks: int,
            num_layers: int,
            num_heads: int,
            head_size: int,
            repeat_slots: bool,
            dtype: torch.dtype,
            device: torch.device,
    ):
        self.block_size = BLOCK_SIZE
        self.num_blocks = num_blocks
        self.repeat_slots = repeat_slots
        self.need_nz = NPUSocInfo().need_nz

        if self.need_nz:
            self.kv_cache = [
                (
                    torch.empty(
                        (num_blocks, num_heads * head_size // 16, self.block_size, 16),
                        dtype=dtype,
                        device=device,
                    ),
                    torch.empty(
                        (num_blocks, num_heads * head_size // 16, self.block_size, 16),
                        dtype=dtype,
                        device=device,
                    ),
                )
                for _ in range(num_layers)
            ]
        else:
            self.kv_cache = [
                (
                    torch.empty(
                        (num_blocks, self.block_size, num_heads, head_size),
                        dtype=dtype,
                        device=device,
                    ),
                    torch.empty(
                        (num_blocks, self.block_size, num_heads, head_size),
                        dtype=dtype,
                        device=device,
                    ),
                )
                for _ in range(num_layers)
            ]
        self.free_block_mask = torch.ones(num_blocks, dtype=torch.int32, device="cpu")
```



```
    self.slots = torch.arange(
        0, num_blocks * self.block_size, dtype=torch.int64
    ).view(num_blocks, self.block_size)

def __repr__(self):
    return (f"CacheManager: "
            f"num_blocks={self.num_blocks},"
            f"block_size={self.block_size},"
            f"free_block_mask={self.free_block_mask},"
            f"slots={self.slots},"
            f"k_cache shape={self.kv_cache[0][0].shape},"
            f"v_cache shape={self.kv_cache[0][1].shape}")

def allocate(
        self,
        needed_blocks_slots: List[Tuple[int, int]],
        blocks: int,
        max_blocks: int,
        device: torch.device,
):
    # Get free blocks indices by finding values in mask that are not set to 0
    free_block_indices = self.free_block_mask.nonzero()
    if blocks > len(free_block_indices):
        raise RuntimeError(
            f"Out of available cache blocks: asked {blocks}, only {len(free_block_indices)} free blocks"
        )

    # Slice by the number of required blocks
    block_indices = free_block_indices[:blocks]
    block_indices = block_indices.flatten()

    # Padded block tables
    block_tables_tensor = torch.zeros(
        (len(needed_blocks_slots), max_blocks), dtype=torch.int32
    )

    # Allocate paged attention blocks
    cumulative_blocks = 0
    slots = []
    block_tables = []
    for i, (needed_blocks, needed_slots) in enumerate(needed_blocks_slots):
        # Get allocated blocks for this sequence
        allocated_blocks = block_indices[
                           cumulative_blocks: cumulative_blocks + needed_blocks
                           ]
        # Get slots for the allocated blocks
        all_slots = self.slots[allocated_blocks].flatten()

        # Repeat slots in the case of context sliding window
        if needed_slots > len(all_slots) and self.repeat_slots:
            repeats = math.ceil(needed_slots / len(all_slots))
            all_slots = all_slots.repeat(repeats)

        allocated_slots = all_slots[:needed_slots]

        slots.append(allocated_slots)
        block_tables.append(allocated_blocks.tolist())
        block_tables_tensor[i, :needed_blocks] = allocated_blocks
        cumulative_blocks += needed_blocks

    block_tables = block_tables
    block_tables_tensor = block_tables_tensor.to(device)
    slots = torch.concat(slots).to(device)

    # Allocate the required number of blocks by setting the mask to 0
    self.free_block_mask[block_indices] = 0

    return block_tables, block_tables_tensor, slots
```



```
    def free(self, block_indices: Optional[List[int]]):
        if block_indices is not None and block_indices:
            # Reset mask
            self.free_block_mask[block_indices] = 1


def set_cache_manager(
        num_blocks: int,
        num_layers: int,
        num_heads: int,
        head_size: int,
        repeat_slots: bool,
        dtype: torch.dtype,
        device: torch.device,
) -> CacheManager:
    global CACHE_MANAGER
    if CACHE_MANAGER is not None:
        del CACHE_MANAGER
        torch.npu.empty_cache()
        gc.collect()

    CACHE_MANAGER = CacheManager(
        num_blocks, num_layers, num_heads, head_size, repeat_slots, dtype, device
    )
    return CACHE_MANAGER


def get_cache_manager() -> CacheManager:
    global CACHE_MANAGER
    if CACHE_MANAGER is None:
        raise RuntimeError("cache manager was not initialized")

    return CACHE_MANAGER
```

- Tgi-MindIE/tgi_npu/info.py

```
#!/usr/bin/env python3
# coding=utf-8
# Copyright (c) Huawei Technologies Co., Ltd. 2024-2024. All rights reserved.

from dataclasses import dataclass
import torch_npu


@dataclass
class NPUSocInfo:
    soc_version: int = -1
    need_nz: bool = False

    def __post_init__(self):
        self.soc_version = torch_npu._C._npu_get_soc_version()
        if self.soc_version in (100, 101, 102, 103, 104, 200, 201, 202, 203):
            self.need_nz = True
```

- Tgi-MindIE/tgi_npu/mind_model.py

```
# Part of codes in this file was copied from project[huggingface][text-generation-inference]
#!/usr/bin/env python3
# coding=utf-8

import math
import itertools
import os
from typing import Optional, Tuple, List, Type
from dataclasses import dataclass

import torch_npu
import torch
from loguru import logger
from opentelemetry import trace
import numpy as np
from transformers import PreTrainedTokenizerBase
```

```
from mindie_llm.text_generator.adapter.generator_torch import GeneratorTorch
from mindie_llm.modeling.backend_type import BackendType

from text_generation_server.models.flash_causal_lm import FlashCausalLM, FlashCausalLMBatch
from text_generation_server.models.types import (
    Batch,
    PrefillTokens,
    Generation,
    GeneratedText
)
from text_generation_server.utils import StoppingCriteria
from text_generation_server.pb import generate_pb2
from text_generation_server.utils.dist import RANK, MEMORY_FRACTION
from text_generation_server.utils.import_utils import (
    empty_cache,
    synchronize,
    get_free_memory,
)
from tgi_npu.tokens_mindie import MindIELLMHeterogeneousNextTokenChooser
from tgi_npu.cache_manager import (
    BLOCK_SIZE,
    get_cache_manager,
    set_cache_manager,
)

tracer = trace.get_tracer(__name__)


@dataclass
class MindFlashCausalLMBatch(FlashCausalLMBatch):
    next_token_chooser: MindIELLMHeterogeneousNextTokenChooser
    all_input_ids_tensor: torch.Tensor
    block_tables: Optional[List[List[int]]]
    block_tables_tensor: Optional[torch.Tensor]

    def __repr__(self):
        return (f"MindFlashCausalLMBatch: batch_id={self.batch_id},"
                f"requests_idx_mapping={self.requests_idx_mapping},"
                f"input_ids={self.input_ids},"
                f"position_ids={self.position_ids},"
                f"cu_seqlen_prefill={self.cu_seqlen_prefill},"
                f"start_slots={self.start_slots},"
                f"slot_indices={self.slot_indices},"
                f"needed_blocks_slots={self.needed_blocks_slots},"
                f"block_tables={self.block_tables},"
                f"block_tables_tensor={self.block_tables_tensor},"
                f"slots={self.slots},"
                f"max_seqlen={self.max_seqlen},"
                f"prefill_head_indices={self.prefill_head_indices},"
                f"prefill_next_token_indices={self.prefill_next_token_indices},"
                f"prefill_cu_outlens={self.prefill_cu_outlens},"
                f"input_lengths={self.input_lengths},"
                f"input_lengths_tensor={self.input_lengths_tensor},"
                f"prefix_offsets={self.prefix_offsets},"
                f"read_offsets={self.read_offsets},"
                f"all_input_ids_tensor={self.all_input_ids_tensor},"
                f"next_token_chooser={self.next_token_chooser},"
                f"stopping_criterias={self.stopping_criterias},"
                f"blocks={self.blocks},"
                f"max_blocks={self.max_blocks}")

    def __len__(self):
        return len(self.requests)

    def __del__(self):
        if self.block_tables is not None and self.block_tables:
            global CACHE_MANAGER
            # Free blocks
            get_cache_manager().free(
```



```
            list(itertools.chain.from_iterable(self.block_tables))
        )

    @classmethod
    def batch_tokenized_inputs(cls, requests, tokenizer):
        batch_inputs = []
        max_truncation = 0
        for r in requests:
            batch_inputs.append(r.inputs)
            max_truncation = max(max_truncation, r.truncate)
        batch_tokenized_inputs = tokenizer(batch_inputs, truncation=True, max_length=max_truncation)
["input_ids"]
        return batch_tokenized_inputs


    @classmethod
    def from_pb(
            cls,
            pb: generate_pb2.Batch,
            tokenizer: PreTrainedTokenizerBase,
            dtype: torch.dtype,
            device: torch.device,
    ) -> "MindFlashCausalLMBatch":

        batch_tokenized_inputs = cls.batch_tokenized_inputs(pb.requests, tokenizer)

        position_ids = []
        cu_seqlen_prefill = [0]
        needed_blocks_slots = []
        start_slots = []
        slot_indices = []

        input_lengths = []
        prefix_offsets = []
        read_offsets = []
        all_input_ids = []
        requests_idx_mapping = {}

        all_prefill_logprobs = True
        no_prefill_logprobs = True
        prefill_head_indices = []
        prefill_next_token_indices = []
        prefill_cu_outlens = [0]

        next_token_chooser_parameters = []
        stopping_criterias = []

        # Cumulative length
        cumulative_length = 0
        cumulative_max_length = 0
        prefill_out_cumulative_length = 0

        blocks = 0
        max_seqlen = 0
        max_length = 0
        max_blocks = 0

        # Parse batch
        for i, (r, tokenized_input) in enumerate(
                zip(pb.requests, batch_tokenized_inputs)
        ):
            # request id -> idx in list mapping
            requests_idx_mapping[r.id] = i

            tokenized_input = tokenized_input[-r.truncate:]
            if (
                    tokenized_input[0] == tokenizer.bos_token_id
                    and tokenized_input[1] == tokenizer.bos_token_id
            ):
```



```
            tokenized_input = tokenized_input[1:]

        input_length = len(tokenized_input)
        input_lengths.append(input_length)

        prefix_offsets.append(input_length - 5)
        read_offsets.append(input_length)

        all_input_ids.append(tokenized_input)

        # Position ids
        request_position_ids = torch.arange(0, input_length, dtype=torch.int32)
        position_ids.append(request_position_ids)

        # Add cumulative lengths of all previous inputs
        cu_seqlen_prefill.append(cumulative_length + input_length)

        next_token_chooser_parameters.append(r.parameters)

        stopping_criteria = StoppingCriteria.from_pb(
            r.stopping_parameters, tokenizer
        )
        max_new_tokens = stopping_criteria.max_new_tokens
        stopping_criterias.append(stopping_criteria)

        # Paged attention
        # Remove one as the first token des not have a past
        total_tokens = input_length + max_new_tokens - 1
        needed_blocks = math.ceil(total_tokens / BLOCK_SIZE)
        blocks += needed_blocks
        needed_blocks_slots.append((needed_blocks, total_tokens))
        start_slots.append(cumulative_max_length)

        request_slot_indices = torch.arange(
            cumulative_max_length,
            cumulative_max_length + input_length,
            dtype=torch.int64,
        )
        slot_indices.append(request_slot_indices)

        all_prefill_logprobs = all_prefill_logprobs and r.prefill_logprobs
        no_prefill_logprobs = no_prefill_logprobs and not r.prefill_logprobs

        if r.prefill_logprobs:
            prefill_head_indices.append(request_position_ids + cumulative_length)
            prefill_next_token_indices.append(
                prefill_out_cumulative_length + input_length - 1
            )
            prefill_cu_outlens.append(prefill_out_cumulative_length + input_length)
            prefill_out_cumulative_length += input_length
        else:
            prefill_head_indices.append(
                torch.tensor(
                    [cumulative_length + input_length - 1], dtype=torch.int64
                )
            )
            prefill_next_token_indices.append(prefill_out_cumulative_length)
            prefill_cu_outlens.append(prefill_out_cumulative_length + 1)
            prefill_out_cumulative_length += 1

        # Update
        cumulative_length += input_length
        cumulative_max_length += total_tokens
        max_seqlen = max(max_seqlen, input_length)
        max_blocks = max(max_blocks, needed_blocks)
        max_length = max(max_length, input_length + max_new_tokens)

    next_token_chooser = MindIELLMHeterogeneousNextTokenChooser.from_pb(
        pb=next_token_chooser_parameters, dtype=dtype, device=device
```



```
    )
    start_slots = torch.tensor(start_slots, dtype=torch.int64)

    # Padded all_input_ids_tensor
    all_input_ids_tensor = np.zeros(
        (len(all_input_ids), max_length), dtype=np.int64
    )
    for i, input_ids in enumerate(all_input_ids):
        all_input_ids_tensor[i, : len(input_ids)] = input_ids

    # Create tensors on device
    all_input_ids_tensor = torch.tensor(
        all_input_ids_tensor, dtype=torch.int64, device=device
    )

    if len(pb.requests) > 1:
        input_ids = np.concatenate(all_input_ids, dtype=np.int64)
        position_ids = torch.cat(position_ids)
        slot_indices = torch.cat(slot_indices)
    else:
        input_ids = all_input_ids[0]
        position_ids = position_ids[0]
        slot_indices = slot_indices[0]

    cu_seqlen_prefill = torch.tensor(
        cu_seqlen_prefill, device=device, dtype=torch.int64
    )

    position_ids = position_ids.to(device)
    slot_indices = slot_indices.to(device)
    input_ids = torch.tensor(input_ids, dtype=torch.int64, device=device)
    input_lengths_tensor = torch.tensor(
        input_lengths, dtype=torch.int64, device=device
    )

    if all_prefill_logprobs:
        prefill_head_indices = None
        prefill_next_token_indices = cu_seqlen_prefill[1:] - 1
    elif no_prefill_logprobs:
        prefill_head_indices = cu_seqlen_prefill[1:] - 1
        prefill_next_token_indices = None
    else:
        prefill_head_indices = torch.tensor(
            torch.cat(prefill_head_indices), dtype=torch.int64, device=device
        )
        prefill_next_token_indices = torch.tensor(
            prefill_next_token_indices, dtype=torch.int64, device=device
        )

    return cls(
        batch_id=pb.id,
        requests=pb.requests,
        requests_idx_mapping=requests_idx_mapping,
        input_ids=input_ids,
        position_ids=position_ids,
        cu_seqlen_prefill=cu_seqlen_prefill,
        start_slots=start_slots,
        slot_indices=slot_indices,
        needed_blocks_slots=needed_blocks_slots,
        block_tables=None,
        block_tables_tensor=None,
        slots=None,
        max_seqlen=max_seqlen,
        prefill_head_indices=prefill_head_indices,
        prefill_next_token_indices=prefill_next_token_indices,
        prefill_cu_outlens=prefill_cu_outlens,
        input_lengths=input_lengths,
        input_lengths_tensor=input_lengths_tensor,
        prefix_offsets=prefix_offsets,
```



```
            read_offsets=read_offsets,
            all_input_ids=all_input_ids,
            all_input_ids_tensor=all_input_ids_tensor,
            next_token_chooser=next_token_chooser,
            stopping_criterias=stopping_criterias,
            blocks=blocks,
            max_blocks=max_blocks,
        )

    @classmethod
    @tracer.start_as_current_span("concatenate")
    def concatenate(cls, batches: List["MindFlashCausalLMBatch"]) -> "MindFlashCausalLMBatch":
        # Batch attributes
        requests = []
        requests_idx_mapping = {}

        blocks = 0
        total_batch_size = 0
        total_slots = 0
        max_blocks = 0
        max_length = 0
        max_seqlen = 0
        for b in batches:
            total_batch_size += len(b)
            total_slots += len(b.slots)
            blocks += b.blocks
            max_blocks = max(max_blocks, b.max_blocks)
            max_seqlen = max(max_seqlen, b.max_seqlen)
            max_length = max(
                max_length,
                max(
                    input_length
                    + stopping_criteria.max_new_tokens
                    - stopping_criteria.current_tokens
                    for input_length, stopping_criteria in zip(
                        b.input_lengths, b.stopping_criterias
                    )
                ),
            )

        input_ids = batches[0].input_ids.new_empty(total_batch_size)
        position_ids = batches[0].position_ids.new_empty(total_batch_size)
        slots = batches[0].slots.new_empty(total_slots)
        slot_indices = batches[0].slot_indices.new_empty(total_batch_size)
        input_lengths_tensor = batches[0].input_lengths_tensor.new_empty(
            total_batch_size
        )
        block_tables_tensor = batches[0].block_tables_tensor.new_zeros(
            (total_batch_size, max_blocks)
        )
        all_input_ids_tensor = batches[0].all_input_ids_tensor.new_zeros(
            (total_batch_size, max_length)
        )

        start_slots = []
        block_tables = []
        all_input_ids = []

        input_lengths = []
        prefix_offsets = []
        read_offsets = []

        next_token_chooser_parameters = []
        stopping_criterias = []

        # Cumulative length
        cumulative_batch_size = 0
        cumulative_slots = 0
```

```
    for i, batch in enumerate(batches):
        requests.extend(batch.requests)

        if i == 0:
            requests_idx_mapping = batch.requests_idx_mapping
        else:
            # We need to offset the mapping for each batch by the cumulative batch size
            for k, v in batch.requests_idx_mapping.items():
                requests_idx_mapping[k] = v + cumulative_batch_size

        start_index = cumulative_batch_size
        end_index = cumulative_batch_size + len(batch)
        slots_start_index = cumulative_slots
        slots_end_index = cumulative_slots + len(batch.slots)

        # Copy tensors (GPU)
        input_ids[start_index:end_index] = batch.input_ids
        position_ids[start_index:end_index] = batch.position_ids
        slot_indices[start_index:end_index] = batch.slot_indices + cumulative_slots
        input_lengths_tensor[start_index:end_index] = batch.input_lengths_tensor
        slots[slots_start_index:slots_end_index] = batch.slots

        all_input_ids_tensor[
            start_index:end_index, : batch.all_input_ids_tensor.shape[1]
        ] = batch.all_input_ids_tensor[:, :max_length]

        block_tables_tensor[
            start_index:end_index, : batch.block_tables_tensor.shape[1]
        ] = batch.block_tables_tensor[:, :max_blocks]

        start_slots.append(batch.start_slots + cumulative_slots)

        block_tables.extend(batch.block_tables)
        all_input_ids.extend(batch.all_input_ids)

        input_lengths.extend(batch.input_lengths)
        prefix_offsets.extend(batch.prefix_offsets)
        read_offsets.extend(batch.read_offsets)

        next_token_chooser_parameters.extend([r.parameters for r in batch.requests])
        stopping_criterias.extend(batch.stopping_criterias)

        # Update
        cumulative_batch_size += len(batch)
        cumulative_slots += len(batch.slots)

    start_slots = torch.concat(start_slots)

    next_token_chooser = MindIELLMHeterogeneousNextTokenChooser.from_pb(
        pb=next_token_chooser_parameters,
        dtype=batches[0].next_token_chooser.dtype,
        device=batches[0].next_token_chooser.device,
    )

    # Needed to avoid dropping blocks when the batches will go out of scope
    for b in batches:
        b.block_tables = None
        del b

    return cls(
        batch_id=batches[0].batch_id,
        requests=requests,
        requests_idx_mapping=requests_idx_mapping,
        input_ids=input_ids,
        position_ids=position_ids,
        cu_seqlen_prefill=None,
        start_slots=start_slots,
        slot_indices=slot_indices,
        needed_blocks_slots=None,
```



```
            block_tables=block_tables,
            block_tables_tensor=block_tables_tensor,
            slots=slots,
            max_seqlen=max_seqlen,
            prefill_head_indices=None,
            prefill_next_token_indices=None,
            prefill_cu_outlens=None,
            input_lengths=input_lengths,
            input_lengths_tensor=input_lengths_tensor,
            prefix_offsets=prefix_offsets,
            read_offsets=read_offsets,
            all_input_ids=all_input_ids,
            all_input_ids_tensor=all_input_ids_tensor,
            next_token_chooser=next_token_chooser,
            stopping_criterias=stopping_criterias,
            blocks=blocks,
            max_blocks=max_blocks,
        )

    @tracer.start_as_current_span("filter")
    def filter(self, request_ids: List[int]) -> "MindFlashCausalLMBatch":
        if len(request_ids) == 0:
            raise ValueError("Batch must have at least one request")
        # We assume that if len(requests) == len(self) then the requests are the same
        if len(request_ids) == len(self):
            return self

        device = self.input_ids.device

        # New values after filtering
        requests_idx_mapping = {}

        # Used to index into tensors
        indices = []

        # slots to keep after filtering
        slot_filtering_indices = torch.zeros(
            self.slots.shape[0], dtype=torch.bool, device=device
        )

        # Create on CPU to only move to GPU once instead of at every copy
        slot_indices = torch.empty(len(request_ids), dtype=torch.int64)
        max_seqlen = 0

        requests = []
        start_slots = []
        block_tables = []
        all_input_ids = []

        input_lengths = []
        prefix_offsets = []
        read_offsets = []

        stopping_criterias = []

        blocks = 0
        max_blocks = 0
        # Cumulative length
        cumulative_max_length = 0

        for i, request_id in enumerate(request_ids):
            idx = self.requests_idx_mapping[request_id]
            indices.append(idx)
            requests_idx_mapping[request_id] = i

            requests.append(self.requests[idx])

            # Get length
            request_input_length = self.input_lengths[idx]
```



```
        max_seqlen = max(max_seqlen, request_input_length)

        all_input_ids.append(self.all_input_ids[idx])

        input_lengths.append(request_input_length)
        prefix_offsets.append(self.prefix_offsets[idx])
        read_offsets.append(self.read_offsets[idx])

        stopping_criteria = self.stopping_criterias[idx]
        stopping_criterias.append(stopping_criteria)

        remaining_tokens = (
            stopping_criteria.max_new_tokens - stopping_criteria.current_tokens
        )

        request_block_table = self.block_tables[idx]
        blocks += len(request_block_table)
        block_tables.append(request_block_table)
        start_slots.append(cumulative_max_length)

        # Copy to tensor (CPU)
        slot_indices[i] = cumulative_max_length + request_input_length - 1

        # Set slice
        slot_filtering_indices[
            self.start_slots[idx] : self.start_slots[idx]
            + request_input_length
            + remaining_tokens
            - 1
        ] = True

        cumulative_max_length += request_input_length + remaining_tokens - 1

        max_blocks = max(max_blocks, len(request_block_table))

    block_indices_to_free = []
    # Iterate on all requests
    for i, r in enumerate(self.requests):
        # Filter requests that are not part of the new batch
        if r.id not in requests_idx_mapping.keys():
            block_indices_to_free.extend(self.block_tables[i])
    # Free blocks
    get_cache_manager().free(block_indices_to_free)
    # Needed to avoid dropping blocks when the batches will go out of scope
    self.block_tables = None

    # Index into tensors
    input_ids = self.input_ids[indices]
    position_ids = self.position_ids[indices]
    all_input_ids_tensor = self.all_input_ids_tensor[indices]
    block_tables_tensor = self.block_tables_tensor[indices]
    input_lengths_tensor = self.input_lengths_tensor[indices]
    slots = self.slots[slot_filtering_indices]
    next_token_chooser = self.next_token_chooser.filter(indices)

    start_slots = torch.tensor(start_slots, dtype=torch.int64)

    # Move to GPU now that we have the whole tensor
    slot_indices = slot_indices.to(device)

    return MindFlashCausalLMBatch(
        batch_id=self.batch_id,
        requests=requests,
        requests_idx_mapping=requests_idx_mapping,
        input_ids=input_ids,
        position_ids=position_ids,
        cu_seqlen_prefill=None,
        start_slots=start_slots,
        slot_indices=slot_indices,
```



```
            needed_blocks_slots=None,
            block_tables=block_tables,
            block_tables_tensor=block_tables_tensor,
            slots=slots,
            max_seqlen=max_seqlen,
            prefill_head_indices=None,
            prefill_next_token_indices=None,
            prefill_cu_outlens=None,
            input_lengths=input_lengths,
            input_lengths_tensor=input_lengths_tensor,
            prefix_offsets=prefix_offsets,
            read_offsets=read_offsets,
            all_input_ids=all_input_ids,
            all_input_ids_tensor=all_input_ids_tensor,
            next_token_chooser=next_token_chooser,
            stopping_criterias=stopping_criterias,
            blocks=blocks,
            max_blocks=max_blocks,
        )

    def to_pb(self) -> generate_pb2.CachedBatch:
        return generate_pb2.CachedBatch(
            id=self.batch_id,
            request_ids=[r.id for r in self.requests],
            size=len(self),
            max_tokens=self.blocks * BLOCK_SIZE,
        )


class MindModel(FlashCausalLM):
    def __init__(
            self,
            model_id: str
    ):
        logger.warning("Initialize mindie-llm model.")
        rank = int(os.getenv("RANK", "0"))
        world_size = int(os.getenv("WORLD_SIZE", "1"))
        model_config = {
            'backend_type': BackendType.ATB,
            'rank': rank,
            'world_size': world_size,
            'model_id': model_id,
            'num_threads': 8,
            'local_rank': rank,
            'npu_device_id': rank
        }
        self.model_runner = GeneratorTorch(model_config)
        super(MindModel, self).__init__(
            model=self.model_runner.model_wrapper.model_runner.model,
            tokenizer=self.model_runner.tokenizer,
            num_layers=self.model_runner.model_info.num_layers,
            num_kv_heads=self.model_runner.model_info.num_kv_heads,
            head_size=self.model_runner.model_info.head_size,
            dtype=self.model_runner.model_info.dtype,
            device=self.model_runner.model_info.device,
            rank=self.model_runner.rank,
            world_size=self.model_runner.world_size,
        )
        logger.warning("MindModel from tgi_npu initialized.")

    def __del__(self):
        del self.model_runner.model_wrapper

    @property
    def batch_type(self) -> Type[MindFlashCausalLMBatch]:
        return MindFlashCausalLMBatch

    def forward(
            self, batch: MindFlashCausalLMBatch
```



```
) -> Tuple[torch.Tensor, Optional[torch.Tensor]]:
    input_ids = batch.input_ids
    position_ids = batch.position_ids
    cu_seqlen_prefill = batch.cu_seqlen_prefill
    kv_cache = get_cache_manager().kv_cache
    block_tables = batch.block_tables_tensor
    slots = batch.slots[batch.slot_indices]
    input_lengths = batch.input_lengths_tensor
    max_s = batch.max_seqlen
    lm_head_indices = batch.prefill_head_indices

    return self.model_runner.forward_tensor(
        input_ids=input_ids,
        position_ids=position_ids,
        is_prefill=cu_seqlen_prefill is not None,
        kv_cache=kv_cache,
        block_tables=block_tables,
        slots=slots,
        input_lengths=input_lengths,
        max_seq_len=max_s,
        lm_head_indices=lm_head_indices,
    )

@tracer.start_as_current_span("generate_token")
def generate_token(
        self, batch: MindFlashCausalLMBatch
) -> Tuple[List[Generation], Optional[MindFlashCausalLMBatch]]:
    prefill = batch.cu_seqlen_prefill is not None
    prefill_logprobs = batch.prefill_next_token_indices is not None

    if batch.needed_blocks_slots:
        # Allocate blocks to this batch
        block_tables, block_tables_tensor, slots = get_cache_manager().allocate(
            batch.needed_blocks_slots,
            batch.blocks,
            batch.max_blocks,
            batch.input_ids.device,
        )
        batch.needed_blocks_slots = None
        batch.block_tables = block_tables
        batch.block_tables_tensor = block_tables_tensor
        batch.slots = slots

    try:
        out = self.forward(batch)
    except Exception as e:
        del batch
        raise e

    if prefill:
        next_token_logits = (
            out[batch.prefill_next_token_indices] if prefill_logprobs else out
        )
    else:
        logger.debug(f"Decode batch size {batch.input_ids.shape[0]}")
        next_token_logits = out

    request_ids = [req.id for req in batch.requests]
    next_input_ids, next_token_logprobs = batch.next_token_chooser(
        request_ids, prefill,
        batch.all_input_ids_tensor[:, : batch.max_seqlen], next_token_logits
    )

    if prefill:
        if len(batch) > 1 and prefill_logprobs:
            # We create the prefill_tokens_indices tensor that will be used to gather prefill logprobs
            # When batch == 1, we will just use the batch.input_ids values directly
            prefill_tokens_indices = batch.input_ids.new_zeros(len(out))
```



```
            next_position_ids = batch.position_ids.new_empty(len(batch))
            batch.slot_indices = batch.slot_indices[batch.cu_seqlen_prefill[1:] - 1]
            # We do not need cu_seqlen_prefill anymore
            batch.cu_seqlen_prefill = None
        else:
            prefill_logprobs = None
            next_position_ids = batch.position_ids

        # Cumulative length
        cumulative_length = 0

        # Results
        generations: List[Generation] = []
        stopped = True

        # Zipped iterator
        iterator = zip(
            batch.input_lengths,
            batch.all_input_ids,
        )

        # We do two for loops as the first one can run completely asynchronously from the GPU while
for the second
        # one, we need to first do a GPU <-> CPU sync
        # It is faster if we delay this sync for the maximum amount of time

        # For each member of the batch
        for i, (
                input_length,
                all_input_ids,
        ) in enumerate(iterator):
            # Indexing metadata
            start_index = cumulative_length
            end_index = cumulative_length + input_length

            if prefill:
                # Indexing metadata
                out_start_index = batch.prefill_cu_outlens[i]
                out_end_index = batch.prefill_cu_outlens[i + 1]
                out_length = out_end_index - out_start_index

                # Initialize position_ids
                # In decode, we do not need this as we can just increment position ids
                next_position_ids[i] = batch.position_ids[end_index - 1]

                # Used to gather prefill logprobs
                # Copy batch.input_ids to prefill_token_indices
                if prefill_logprobs:
                    if len(batch) > 1:
                        prefill_tokens_indices[
                        out_start_index: out_end_index - 1
                        ] = batch.input_ids[start_index + 1: start_index + out_length]
                    else:
                        # Set prefill_tokens_indices to the correct slice
                        prefill_tokens_indices = batch.input_ids[
                                                 start_index + 1: start_index + out_length
                                                 ]

            batch.all_input_ids_tensor[i, input_length] = next_input_ids[i]

            cumulative_length += input_length


        # Set values in batch
        batch.input_ids = next_input_ids
        batch.position_ids = next_position_ids + 1
        batch.input_lengths_tensor += 1
        batch.slot_indices += 1
```



```
    if prefill and prefill_logprobs:
        # Get prefill logprobs
        prefill_logprobs_tensor = torch.log_softmax(out, -1)
        prefill_logprobs = torch.gather(
            prefill_logprobs_tensor, 1, prefill_tokens_indices.view(-1, 1)
        )
        # GPU <-> CPU sync
        prefill_logprobs = prefill_logprobs.view(-1).tolist()

    # GPU <-> CPU sync
    next_token_logprobs = next_token_logprobs
    next_token_ids = batch.input_ids.tolist()

    # Zipped iterator
    iterator = zip(
        batch.requests,
        batch.input_lengths,
        batch.prefix_offsets,
        batch.read_offsets,
        batch.stopping_criterias,
        batch.all_input_ids,
        batch.next_token_chooser.do_sample,
        batch.next_token_chooser.seeds,
        next_token_ids,
        next_token_logprobs,
    )

    # For each member of the batch
    for i, (
            request,
            input_length,
            prefix_offset,
            read_offset,
            stopping_criteria,
            all_input_ids,
            do_sample,
            seed,
            next_token_id,
            next_token_logprob,
    ) in enumerate(iterator):
        # Append next token to all tokens
        all_input_ids.append(next_token_id)
        # Generated token
        next_token_text, prefix_offset, read_offset = self.decode_token(
            all_input_ids,
            prefix_offset,
            read_offset,
        )

        # Evaluate stopping criteria
        stop, reason = stopping_criteria(
            next_token_id,
            next_token_text,
        )

        if not stop:
            stopped = False

        # Shard generations
        # All generations will be appended in the rust sharded client
        if i % self.world_size == self.rank:
            if stop:
                # Decode generated tokens
                # decode & decode_token
                output_text = self.decode(
                    all_input_ids[-stopping_criteria.current_tokens:]
                )
                generated_text = GeneratedText(
                    output_text,
```



```
                    stopping_criteria.current_tokens,
                    reason,
                    seed if do_sample else None,
                )
            else:
                generated_text = None

            # Prefill
            if prefill and request.prefill_logprobs:
                out_start_index = batch.prefill_cu_outlens[i]
                out_end_index = batch.prefill_cu_outlens[i + 1]

                # Remove generated token to only have prefill and add nan for first prompt token
                request_prefill_logprobs = [float("nan")] + prefill_logprobs[
                                                    out_start_index: out_end_index - 1
                                                    ]
                prefill_token_ids = all_input_ids[:-1]
                prefill_texts = self.tokenizer.batch_decode(
                    prefill_token_ids,
                    clean_up_tokenization_spaces=False,
                    skip_special_tokens=False,
                )
                # PrefillTokens & Tokens
                prefill_tokens = PrefillTokens(
                    prefill_token_ids, request_prefill_logprobs, prefill_texts
                )
            else:
                prefill_tokens = None

            generation = Generation(
                request.id,
                prefill_tokens,
                next_token_id,
                next_token_logprob,
                next_token_text,
                next_token_id in self.all_special_ids,
                generated_text,
            )

            generations.append(generation)

        # Update values
        batch.input_lengths[i] = input_length + 1
        batch.prefix_offsets[i] = prefix_offset
        batch.read_offsets[i] = read_offset
        batch.all_input_ids[i] = all_input_ids

    if stopped:
        del batch
        # No need to return a batch if we know that all requests stopped
        return_value = None
        return generations, return_value

    batch.prefill_cu_outlens = None
    batch.prefill_head_indices = None
    batch.prefill_next_token_indices = None
    batch.max_seqlen = batch.max_seqlen + 1

    return generations, batch

def warmup(self, batch: MindFlashCausalLMBatch):
    # The warmup batch is the biggest batch we could ever receive
    empty_cache()

    peak_memory = torch_npu.npu.max_memory_allocated()
    logger.warning(f">>>>before warmup peak_memory {peak_memory}")
    try:
        cache_manager = set_cache_manager(
            batch.blocks,
```



```
                self.num_layers,
                self.num_kv_heads,
                self.head_size,
                False,
                self.dtype,
                self.device,
            )
            _, batch = self.generate_token(batch)
        except torch.cuda.OutOfMemoryError as e:
            raise RuntimeError(
                f"Not enough memory to handle {len(batch.input_ids)} prefill tokens. "
                f"You need to decrease `--max-batch-prefill-tokens`"
            ) from e

        # Inspired by the original implementation in [vllm](https://github.com/vllm-project/vllm)
        # Calculate the number of blocks that can be allocated with the free memory
        dtype_size = torch.tensor([], dtype=self.dtype).element_size()
        cache_block_size = BLOCK_SIZE * self.num_kv_heads * self.head_size
        total_cache_size = self.num_layers * cache_block_size * 2 * dtype_size
        torch_npu.npu.synchronize()
        total_gpu_memory = torch_npu.npu.get_device_properties(self.device).total_memory
        peak_memory = torch_npu.npu.max_memory_allocated()
        logger.info(
            f">>>>dtype_size {dtype_size}, cache_block_size {cache_block_size}, num_kv_heads
{self.num_kv_heads}, "
            f"total_cache_size {total_cache_size}, peak_memory {peak_memory}")
        total_free_memory = total_gpu_memory - peak_memory
        logger.info(f">>>>total_free_memory {total_free_memory}, total_gpu_memory
{total_gpu_memory}, "
                    f"MEMORY_FRACTION {MEMORY_FRACTION}")

        free_memory = max(
            0, total_free_memory - (1 - MEMORY_FRACTION) * total_gpu_memory
        )

        num_blocks = (
            # Leave 5% for some wiggle room
            int((free_memory * 0.95) // total_cache_size)
            # Add batch.blocks as we allocated it above, so it is included in the peak memory.
            + cache_manager.num_blocks
        )

        del batch
        del cache_manager

        set_cache_manager(
            num_blocks,
            self.num_layers,
            self.num_kv_heads,
            self.head_size,
            # self.sliding_window is not None,
            False,
            self.dtype,
            self.device,
        )

        logger.warning(f">>>>real CacheManger {get_cache_manager()}")
        peak_memory = torch_npu.npu.max_memory_allocated()
        logger.warning(f">>>>end warmup peak_memory {peak_memory}")
        logger.warning(f"Warmup return {int(num_blocks * BLOCK_SIZE)}")
        return int(num_blocks * BLOCK_SIZE)
```

- Tgi-MindIE/tgi_npu/token_mindie.py

```
# Part of codes in this file was copied from project[huggingface][text-generation-inference]
#!/usr/bin/env python3
# coding=utf-8

import os
from typing import List
```



```
import torch
from loguru import logger
from text_generation_server.pb import generate_pb2

from mindie_llm.text_generator.utils.sampling_metadata import SamplingData, SamplingParam
from mindie_llm.modeling.backend_type import BackendType
from mindie_llm.text_generator.utils.config import SamplerConfig
from mindie_llm.text_generator.samplers.sampler import Sampler

import numpy as np


WRAPPER_KEY = "tensor_wrapper"

try:
    from mindie_llm.text_generator.utils.sampling_metadata import TensorWrapper
except ImportError:
    class TensorWrapper:
        def __init__(self, backend, device):
            self.device = device
            self.backend = backend

        def __call__(self, data):
            if data.dtype == np.int32:
                dtype = torch.int32
            elif data.dtype == np.bool_:
                dtype = torch.bool
            else:
                dtype = None
            return torch.tensor(data, dtype=dtype, device=self.device)
    WRAPPER_KEY = "to_tensor"


def do_filter(sample_param: List, indices):
    if any(sample_param):
        return [sample_param[i] for i in indices]
    return sample_param


class MindIELLMHeterogeneousNextTokenChooser:
    def __init__(self,
                 dtype: torch.dtype,
                 device: torch.device,
                 watermark: List[bool],
                 temperature: List[float],
                 repetition_penalty: List[float],
                 frequency_penalty: List[float],
                 top_k: List[int],
                 top_p: List[float],
                 typical_p: List[float],
                 do_sample: List[bool],
                 seeds: List[int],
                 grammars: List[str],
                 grammar_types: List[int],
                 fsm_grammar_states: List[int],
                 sample_method,  # mindie-llm sampling method
                 ):

        if any([x != 0.0 for x in frequency_penalty]):
            logger.warning(f"Frequency_penalty is supported now in mindie-llm but not supported in TGI
v0.9.4")
        if any(watermark):
            logger.warning(f"Watermark not supported now in mindie-llm")
        if any([x < 1.0 for x in typical_p]):
            logger.warning(f"Typical_p not supported now in mindie-llm")
        if any(grammar_types) or any(grammars) or any(fsm_grammar_states):
            logger.warning(f"Grammar not supported now in mindie-llm")

        self.tensor_wrapper = TensorWrapper(BackendType.ATB, device)
```



```
        wrapper_dict = {WRAPPER_KEY: TensorWrapper(BackendType.ATB, device)}
        self.sample_params = SamplingParam.from_numpy(
            repetition_penalty=np.array(repetition_penalty, dtype=np.float16),
            presence_penalty=None,
            temperature=np.array(temperature, dtype=np.float16),
            top_k=np.array(top_k),
            top_p=np.array(top_p),
            seed=np.array(seeds).astype(np.int32),
            do_sample=np.array(do_sample),
            **wrapper_dict
        )

        # Temp store for filter
        self.temperature = temperature
        self.repetition_penalty = repetition_penalty
        self.frequency_penalty = frequency_penalty
        self.top_k = top_k
        self.top_p = top_p
        self.seeds = seeds
        self.do_sample = do_sample

        self.choice = sample_method
        self.dtype = dtype
        self.device = device
        self.seeds = seeds
        self.do_sample = self.sample_params.do_sample_meta.do_sample_array
        self.fsm_grammar_states = fsm_grammar_states

    def __call__(self,
                 request_ids: List,
                 is_prefill: bool,
                 input_ids: torch.Tensor,
                 scores: torch.Tensor,
                 ):
        batch_size = scores.shape[0]
        speculate_size = 1
        scores = scores.view(batch_size, speculate_size, -1)

        # Don't use SamplingData.from_numpy to avoid tensor.cpu() to transfer large data
        input_ids_int32 = input_ids.to(torch.int32)
        sample_data = SamplingData(all_input_ids=input_ids_int32, output_ids=input_ids_int32,
is_prefill=is_prefill,
                                   request_ids=np.array(request_ids))
        next_ids = torch.zeros((batch_size, speculate_size), device=scores.device, dtype=torch.long)
        for j in range(speculate_size):
            _scores = scores[:, j]
            batch_logits, _next_ids = self.choice(batch_logits=_scores, batch_sampling_data=sample_data,
                                                  batch_sampling_params=self.sample_params)
            scores[:, j] = _scores
            next_ids[:, j] = torch.from_numpy(_next_ids)

        next_ids = next_ids.view(batch_size * speculate_size)
        allscores = scores.view(batch_size * speculate_size, -1)
        alllogprobs = torch.log_softmax(allscores, -1)

        logprobs = alllogprobs
        next_logprobs = torch.gather(logprobs, 1, next_ids.view(-1, 1)).view(-1)

        return next_ids, next_logprobs


    @classmethod
    def from_pb(
            cls,
            pb: List[generate_pb2.NextTokenChooserParameters],
            dtype: torch.dtype,
            device: torch.device,
    ) -> "MindIELLMHeterogeneousNextTokenChooser":
        curr_rank = int(os.getenv("RANK", "0"))
```



```
        sample_method = Sampler(SamplerConfig(
            rank=curr_rank,
            backend_type=BackendType.ATB,
            npu_id=curr_rank
        ))
        return MindIELLMHeterogeneousNextTokenChooser(
            watermark=[pb_.watermark for pb_ in pb],
            temperature=[pb_.temperature for pb_ in pb],
            repetition_penalty=[pb_.repetition_penalty for pb_ in pb],
            frequency_penalty=[],
            top_k=[pb_.top_k for pb_ in pb],
            top_p=[pb_.top_p for pb_ in pb],
            typical_p=[pb_.typical_p for pb_ in pb],
            do_sample=[pb_.do_sample for pb_ in pb],
            seeds=[pb_.seed for pb_ in pb],
            device=device,
            dtype=dtype,
            grammars=[],
            grammar_types=[],
            fsm_grammar_states=[],
            sample_method=sample_method
        )

    def filter(self, indices):
        self.repetition_penalty = do_filter(self.repetition_penalty, indices)
        self.frequency_penalty = do_filter(self.frequency_penalty, indices)
        self.temperature = do_filter(self.temperature, indices)
        self.top_k = do_filter(self.top_k, indices)
        self.top_p = do_filter(self.top_p, indices)
        self.seeds = do_filter(self.seeds, indices)
        self.do_sample = do_filter(self.do_sample, indices)

        self.sample_params = SamplingParam.from_numpy(
            repetition_penalty=np.array(self.repetition_penalty, dtype=np.float16),
            presence_penalty=None,
            frequency_penalty=np.array(self.frequency_penalty, dtype=np.float16),
            temperature=np.array(self.temperature, dtype=np.float16),
            top_k=np.array(self.top_k),
            top_p=np.array(self.top_p),
            seed=np.array(self.seeds).astype(np.int32),
            do_sample=np.array(self.do_sample),
            **self.wrapper_dict
        )
        return self
```

- Tgi-MindIE/pyproject.toml

```
[tool.poetry]
name = "tgi-npu"
version = "0.1.0"
description = "NPU MindIE Adapter for TGI v0.9.4"
authors = ["Your Name <you@example.com>"]
readme = "README.md"
exclude = ["router", "cover"]



[tool.poetry.dependencies]
python = "^3.10"

[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"
```


8 调度工具

llm_engine_test

8.1 llm_engine_test

说明

llm_engine_test命令自本版本起停止演进，且在2024年12月30日后退出并删除。

命令功能

支持用户使用自定义的数据集进行性能验证。

命令格式

./llm_engine_test *可选参数1 可选参数2 可选参数3 可选参数4*

参数说明

参数	说明
可选参数1	自定义数据集名称，若没有输入此参数则选用数据集默认名称：token_input_gsm.csv。
可选参数2	是否记录output id。若传一个大于0的参数，则output id会被写进llm_engine_test同级目录下的token_output.csv。 设置可选参数2时，必须要设置可选参数1。
可选参数3	warmup的次数。 设置可选参数3时，必须要设置可选参数1和可选参数2。
可选参数4	传入config.json文件路径，若没有此参数，则默认读取Service下的config.json文件。

操作步骤

步骤1 配置config.json文件内容，常用配置参数请参见**2.2 配置参数说明**。如需配置其他参数，详情请参见《 MindIE安装指南 》中“配置MindIE > 配置MindIE Server > **单机推理**”章节中的步骤4。

步骤2 提供测试数据集，将数据集文件上传至llm_engine_test所在目录（ '${MINDIE_INSTALL_PATH}/latest/mindie-llm/bin' ）。

用户自定义数据集需满足以下格式要求：

- 每一行表示一条数据，均以1开头，每个token id之间以逗号相隔（每行最后一个token id后不加逗号）。
- 请求数建议不超过50000条。
- token id取值限制以词表为准，即配置项“modelWeightPath”下面的config.json，其中“vocab_size”字段的值。

  以“/data/atb_testdata/weights/llama1-65b-safetensors”目录下的config.json为例，即token id不能超过32000。

```
{
  "architectures": [
    "LlamaForCausalLM"
  ],
  "bos_token_id": 1,
  "eos_token_id": 2,
  "hidden_act": "silu",
  "hidden_size": 8192,
  "initializer_range": 0.02,
  "intermediate_size": 22016,
  "max_sequence_length": 2048,
  "model_type": "llama",
  "num_attention_heads": 64,
  "num_hidden_layers": 80,
  "pad_token_id": 0,
  "rms_norm_eps": 1e-05,
  "tie_word_embeddings": false,
  "torch_dtype": "float16",
  "transformers_version": "4.28.0.dev0",
  "use_cache": true,
  "vocab_size": 32000
}
```

步骤3 执行命令。

```
./bin/llm_engine_test 可选参数1 可选参数2 可选参数3 可选参数4
```

----结束

9 附录

9.1 术语&缩略语

术语/缩略语	含义
LLM	Large Language Model，大语言模型。
TGI	Text Generation Inference，文本生成推理。是一个用于部署和服务大型语言模型的工具包。TGI为最流行的开源LLM提供高性能文本生成，包括LLaMa、Falcon、Starcoder、Bloom、GPT-NeoX等。
vLLM	vLLM是一个开源的大模型推理加速框架。
Triton	Triton是一个开源的推理服务软件，全称为Triton Inference Server。通过Triton，您可以在基于GPU或CPU的各种基础架构（云、数据中心或边缘）上部署、运行和扩展来自任何框架的AI模型。
ContinuousBatching（CB）	连续批处理（ContinuousBatching），也称为动态批处理或基于迭代级的批处理，是一种针对提升LLM迭代推理性能的优化手段，可以减少调度空泡，提升业务吞吐。
PagedAttention（PA）	自回归过程中缓存的K和V张量非常大，PagedAttention灵感来自于操作系统中虚拟内存和分页的经典思想，它可以允许在非连续空间里存储连续的KV张量。
Daemon	Daemon（守护进程）是运行在后台的一种特殊进程。它独立于控制终端并且周期性地执行某种任务或等待处理某些发生的事件。它不需要用户输入就能运行，同时提供某种服务，不仅对整个系统，还可以对某个用户程序提供服务。

术语/缩略语	含义
Backend	模型执行器，推理服务化框架后端对接模型推理层模块。
HDK	Hardware Developer Kit，硬件开发工具包。
ATB	Ascend Transformer Boost 加速库，基于Transformer的神经网络推理加速引擎库。
ATB Models	基于ATB框架开发的模型。
MoE	Mixture of Experts（MoE）。
W8A8	一种量化方式：权重和激活值同时量化为Int8。
W8A16	一种量化方式：对激活值不做量化，仅将权重量化为8 bit。
W8A8SC	一种量化方式：对模型权重进行稀疏、量化和压缩。
KV Cache int8	一种量化方式：将k cache和v cache量化为8 bit。
Anti-Outlier	量化过程中的离群值处理。
MindFormers	MindFormers是基于MindSpore框架的大模型全流程套件，支持大模型训练、微调、评估、推理和部署。

9.2 FAQ

9.2.1 如何开启加速库的强制同步来定位报错

解决方案

```
export ATB_STREAM_SYNC_EVERY_KERNEL_ENABLE=1
export ATB_STREAM_SYNC_EVERY_RUNNER_ENABLE=1
export ATB_STREAM_SYNC_EVERY_OPERATION_ENABLE=1
```

开启环境变量后，运行模型推理，根据加速库日志中的第一个error进一步定位。

9.2.2 当出现 undefinedsymbols: xxx 这样的报错如何定位

解决方案

需要确认MindIE LLM和ATB，CANN，torch，torch_npu是否匹配，abi=0/1的选择是否正确 。

9.2.3 为什么跑精度数据集的时候，最后的精度结果有浮动

1. 将模型后处理部分从sampling改为greedy后，可以基本保证输出文本的稳定性。
2. 由于确定性计算的问题，输出可能存在略微差异。

9.2.4 LLM 推理结果存在乱码

解决方案

检查tokenizer在token转id时，是否使用了正确的模型路径。

9.2.5 什么是确定性计算

确定性计算，是指在输入数据集等输入条件不变时，多次运行推理应用，输出结果每次保持一致。

9.2.6 为什么相同输入送入 MindIE Server 推理，输出存在一定的不确定性

调度框架代码（block查询模式下）是可以保证确定性的，但是在cpu负载等环境因素的影响下，可能会对请求到达时间产生影响，最终影响调度的确定性。例如，客户外部服务查询引擎后得知可以提交10个请求，第一次运行时，10个请求快速达到并组成batch；但第二次运行时，受环境影响部分请求到达较晚，仅有5个请求组成了batch；那么两次运行的结果就会产生差异。

9.2.7 为什么相同输入，组 batch 顺序不同，送入 LLM 模型推理输出不同

解决方案

1. 由于matmul算子在不同行上的累加顺序不完全相同，加之浮点精度没有加法交换律的特性，导致不同行上即使输入完全相同，计算结果也会存在一定的误差。
2. 可以通过设置环境变量export ATB_MATMUL_SHUFFLE_K_ENABLE=0将加速库Matmul的shuffle k功能关闭，关闭之后可以保证所有行上算子累加顺序一致，但matmul性能会下降10%左右 。

9.2.8 在昇腾上进行 LLM 推理，如何保证确定性计算

1. 模型层面：

   通信算子：

   ```
   export LCCL_DETERMINISTIC=1
   export HCCL_DETERMINISTIC=true
   ```

   MatMul：

   ```
   export ATB_MATMUL_SHUFFLE_K_ENABLE=0
   ```

2. 推理引擎：

   MindIE：基于block进行新request的获取。

   TGI：暂不支持。

9.2.9 常见的 LLM 推理性能优化手段都有哪些

算子融合、量化、Tensor并行、ContinuousBatching等。

9.3 环境变量说明

MindIE LLM安装完成后，提供进程级环境变量设置脚本“set_env.sh”，以自动完成环境变量设置。

表 9-1 “set_env.sh”脚本环境变量说明

环境变量名	说明
MINDIE_LLM_BACKEND	MindIE LLM的BACKEND类型，可选值为"atb"、"pt"和"ms"。
PYTHONPATH	用于告诉 Python 解释器在哪个路径下查找用户定义的模块。
MINDIE_LLM_HOME_PATH	MindIE LLM主目录所在路径。默认值为set_env.sh所在的路径。
MINDIE_LLM_CONTINUOUS_BATCHING	MindIE LLM是否连续batching的开关。 ● 0：不开启 ● 1：开启（默认值）
MINDIE_LLM_PYTHON_LOG_TO_FILE	MindIE LLM中Python程序运行时产生的日志是否写入文件的开关。 ● 0：不开启 ● 1：开启
MINDIE_LLM_PYTHON_LOG_TO_STDOUT	MindIE LLM中Python程序运行时产生的日志是否写入标准输出流的开关。 ● 0：不开启（默认值） ● 1：开启
MINDIE_LLM_PYTHON_LOG_PATH	MindIE LLM中Python程序运行时所产生日志文件的存储路径。默认值为${MINDIE_LLM_HOME_PATH}/logs/pythonlog.log。
MINDIE_LLM_PYTHON_LOG_LEVEL	MindIE LLM中Python程序运行时的日志级别，包括CRITICAL, FATAL, ERROR, WARNING, INFO（默认值），DEBUG, NOTSET几种级别。
MINDIE_LLM_RECOMPUTE_THRESHOLD	MindIE LLM中重计算阈值。默认值为0.5。
ATB_SPEED_HOME_PATH	模型仓安装后文件存储路径，MindIE LLM和MindIE Service组件依赖此环境变量。
PYTORCH_INSTALL_PATH	torch三方件的安装路径，使用以下方式获取python3 -c 'import torch, os; print(os.path.dirname(os.path.abspath(torch.__file__)))'。

环境变量名	说明
PYTORCH_NPU_INSTALL_PATH	torch_npu三方件的安装路径，使用以下方式获取python3 -c 'import torch, torch_npu, os; print(os.path.dirname(os.path.abspath(torch_npu.__file__)))'。
LD_LIBRARY_PATH	在LD_LIBRARY_PATH环境变量中，添加${ATB_SPEED_HOME_PATH}/lib路径，以此关联模型仓编译出的动态库； 添加${PYTORCH_INSTALL_PATH}/lib路径，以关联torch三方件提供的动态库； 添加${PYTORCH_NPU_INSTALL_PATH}/lib路径，以关联torch_npu三方件提供的动态库。
ATB_OPERATION_EXECUTE_ASYNC	算子setup和execute异步执行开关。 ● 0：不开启 ● 1：开启（默认值）
TASK_QUEUE_ENABLE	默认模型仓拉起子线程执行execute任务，开启此环境变量后，使用Torch三方件启动的线程执行execute任务。 ● 0：模型仓拉起子线程执行execute任务 ● 1：使用Torch三方件启动的线程执行execute任务（默认值）
ATB_CONTEXT_HOSTTILING_RING	CPU侧用来存放算子Tiling data的Buffer块数量。默认为1。
ATB_CONTEXT_HOSTTILING_SIZE	CPU侧用来存放算子Tiling data的Buffer块大小。默认为102400字节。
ATB_WORKSPACE_MEM_ALLOC_GLOBAL	是否使用全局中间tensor内存分配算法。 开启后会对中间tensor内存进行大小计算与分配。 ● 0：不开启 ● 1：开启（默认值）
ATB_USE_TILING_COPY_STREAM	是否多开启一个Stream做Tiling data从CPU拷贝到NPU的动作。不推荐开启，加速库已默认开启ATB_LAUNCH_KERNEL_WITH_TILING环境变量，打开Tiling data拷贝随算子下发功能。 ● 0：不开启（默认值） ● 1：开启
ATB_OPSRUNNER_KERNEL_CACHE_LOCAL_COUNT	设置op runner的本地cache 槽位数，默认值为1个。
ATB_OPSRUNNER_KERNEL_CACHE_GLOABL_COUNT	设置op runner的全局cache 槽位数，默认值为16个。

可选环境变量说明请参考表9-2。

表 9-2 其他环境变量说明

环境变量名	说明
INF_NAN_MODE_ENABLE	可以开启以下环境变量，开启后若出现溢出，溢出值会被置为NaN；若不开启此变量，则会对溢出值进行截断。 • 0：不开启 • 1：开启（默认值）
INT8_FORMAT_NZ_ENABLE	INT8类型的Tensor转为NZ格式时需要开启此环境变量。 • 0：不开启（默认值） • 1：开启
ATB_ACLNN_CACHE_GLOABL_COUNT	设置AclNN算子所需aclExecutor的全局Cache个数。取值范围：[0, 100)，默认值为16。
ATB_LLM_LCOC_ENABLE	通信计算掩盖功能开关。 • 0：不开启 • 1：开启（默认值）
BIND_CPU	是否将NPU上运行的进程基于CPU亲和度绑核。 • 0：不开启 • 1：开启（默认值）
CPU_BINDING_NUM	设置每个进程绑的核数。该值不得小于0，默认值为None，即将NUMA节点一半的CPU核心数平分给直连在这个NUMA节点上的NPU device。 NUMA（Non-UnifNUMA（Non-Uniform Memory Access，非统一内存访问）架构是一种针对多处理器系统的内存组织方式。
WORLD_SIZE	张量并行数量。
ASCEND_RT_VISIBLE_DEVICES	指定当前机器上可用的逻辑NPU核心，多个核心间使用逗号相连。默认值为None，代表可用的逻辑NPU核心为[1，2，3，...，${WORLD_SIZE} - 1]。
ATB_LLM_HCCL_ENABLE	是否使能HCCL通信后端。 • 0：不开启（默认值） • 1：开启
PROFILING_LEVEL	设置ProfilerLevel。可选项为"Level0"（默认值）、"Level1"、"Level2"、"Level_none"。
ATB_PROFILING_ENABLE	是否采集性能profiling数据。 • 0：否（默认值） • 1：是

环境变量名	说明
PROFILING_FILEPATH	设置profiling文件路径，默认保存在当前路径下profiling文件夹中。
RESERVED_MEMORY_GB	模型运行时动态申请显存池的大小。取值范围：[0, 64)，默认值为3。
NPU_MEMORY_FRACTION	NPU显存利用率。取值范围：(0.0, 1.0]，在ATB Models中默认值为1.0，在MindIE LLM 中默认值为0.8。
ATB_LLM_BENCHMARK_ENABLE	性能数据获取是否打开。 ● 0：不开启（默认值） ● 1：开启
ATB_LLM_BENCHMARK_FILEPATH	性能数据保存路径，默认为None。
ATB_LLM_TOKEN_IDS_SAVE_ENABLE	ModelTest场景下，是否保存token信息。 ● 0：否（默认值） ● 1：是 使用此功能时，“MODELTEST_DATASET_SPECIFIED”环境变量不能为空。
ATB_LLM_TOKEN_IDS_SAVE_FOLDER	ModelTest场景下，token信息的存放路径，默认为当前路径。
MODELTEST_DATASET_SPECIFIED	ModelTest场景下，采用的数据集后处理类型，默认为None。
ATB_LLM_LOGITS_SAVE_ENABLE	ModelTest场景下，是否保存logits信息。 ● 0：否（默认值） ● 1：是 使用此功能时，“MODELTEST_DATASET_SPECIFIED”环境变量不能为空。
ATB_LLM_LOGITS_SAVE_FOLDER	Logits信息的保存路径，默认为当前路径。
ATB_LLM_ENABLE_AUTO_TRANSPOSE	是否开启权重右矩阵自动转置寻优。 ● 0：不开启 ● 1：开启（默认值）
ATB_LOG_TO_STDOUT	ATB的C++日志写入标准输出流的开关。 ● 0：不开启（默认值） ● 1：开启
ATB_LOG_TO_FILE	ATB的C++日志写入文件的开关。 ● 0：不开启（默认值） ● 1：开启

环境变量名	说明
ATB_LOG_LEVEL	ATB C++日志级别。可选范围："TRACE"、"DEBUG"、"INFO"、"WARN"（默认值）、"ERROR","FATAL"。
ATB_HOME_PATH	ATB加速库所在路径。
ASCEND_LAUNCH_BLOCKING	算子同步下发功能开关，用于debug场景。 ● 0：不开启（默认值） ● 1：开启
IS_ALIBI_MASK_FREE	是否支持Speculate。 ● 0：否（默认值） ● 1：是
RANK	指示device的全局ID。取值范围为[0, ${WORLD_SIZE} - 1)，默认值为0。
LONG_SEQ_ENABLE	判断是否为长序列。 ● 0：否（默认值） ● 1：是
LOG_LEVEL	ATB Python日志级别。取值范围："NOTSET"、"DEBUG"、"INFO"（默认值）、"WARN"、"WARNING"、"ERROR"、"FATAL"、"CRITICAL"。
LOG_TO_FILE	ATB Python日志文件的存储路径。默认为空（即不写入文件）。
PYTHON_LOG_MAXSIZE	ATB Python日志单个文件的最大容量（单位：字节），默认值：1073741824（1GB）。
LOCAL_RANK	指示Device的本地ID。取值范围为0到每台机器拥有的NPU卡数-1, 默认值为0。
TIMEIT	默认值为0，要开启性能测试需设置为1，部分场景需开启用来测试性能，详细场景使用情况参考各模型readme。
MAX_SEQ_LEN	RotaryEmbedding入参。
MINDIE_LLM_LOG_TO_STDOUT	MindIE LLM C++日志写入标准输出流开关。 ● 0：不开启（默认值） ● 1：开启
MINDIE_LLM_LOG_TO_FILE	MindIE LLM C++日志写入文件开关。 ● 0：不开启（默认值） ● 1：开启
MINDIE_LLM_LOG_LEVEL	MindIE LLM C++日志级别。取值范围："TRACE"、"DEBUG"、"INFO"、"WARN"（默认值）、"ERROR"、"FATAL"。
SOURCE_DATE_EPOCH	构建whl包所需的时间戳。

环境变量名	说明
RANKTABLEFILE	Ranktable文件存放的路径。 ● 多机推理必须配置。 ● 单机推理建议取消该环境变量（取消命令：unset RANKTABLEFILE）。如果设置该环境变量，文件内容必须正确有效（节点IP地址和device_ip必须正确），否则会导致模型初始化失败。
ATB_LLM_RAZOR_ATTENTION_ENABLE	RA压缩开关。 ● 0：不开启（默认值） ● 1：开启
MIES_USE_MB_SWAPPER	高性能Swap开关。 ● 0：不开启（默认值） ● 1：开启
MIES_PYTHON_LOG_LEVEL	设定MindIE Server Python日志级别。取值范围："CRITICAL"、"ERROR"、"WARNING"、"INFO"(default)、"DEBUG"、"NOTSET"。
MIES_PYTHON_LOG_TO_FILE	MindIE Server Python日志写入文件开关。 ● 0：不开启（默认值） ● 1：开启
MIES_PYTHON_LOG_PATH	MindIE Server Python日志文件存储路径。
MIES_PYTHON_LOG_TO_STDOUT	MindIE Server Python日志写入标准输出流开关。 ● 0：不开启（默认值） ● 1：开启
MINDIE_LLM_PYTHON_LOG_MAXSIZE	MindIE LLM Python单个日志文件的最大容量（单位：字节）。默认值：1073741824。
MINDIE_LLM_PYTHON_LOG_MAXNUM	MindIE LLM Python日志存储的最大数量（单位：件），默认值：10。
MINDIE_LLM_FRAMEWORK_BACKEND	MindIE LLM框架后端类型，当前可选值为"atb"（ATB，默认值）和"ms"（MindSpore）。
MIES_INSTALL_PATH	MindIE Server的安装路径。
MIES_CONFIG_JSON_PATH	MindIE Server配置文件所在路径。
NPU_DEVICE_IDS	使用的NPU卡号。取值范围：[0, ${WORLD_SIZE} - 1]。
MIES_CONTAINER_IP	MindIE Server容器IP地址。
HOST_IP	宿主机IP地址。

环境变量名	说明
MINDIE_LLM_BENCHMARK_ENABLE	是否开启MindIE LLM模块的benchmark功能，开启后将会输出性能数据到指定文件路径。默认为0，改为1可开启。
MINDIE_LLM_BENCHMARK_FILEPATH	指定MindIE LLM模块的benchmark功能输出的性能数据文件路径。默认为"{MINDIE_LLM_HOME_PATH}/logs/benchmark.jsonl"。
MINDIE_LLM_BENCHMARK_RESERVING_RATIO	当性能数据文件超过最大文件大小限制时，旧数据会被新数据覆盖。此环境变量指定保留旧数据的比例，默认为0.1
POST_PROCESSING_SPEED_MODE_TYPE	● 指定后处理加速模式，默认为0，即不开启加速。 ● 指定为1时，开启top_p近似计算，该功能加速较为明显，但会略微影响后处理精度。 ● 指定为2时，开启索引加速，该功能加速效果微弱。 ● 指定为3时，将同时开启上述两个功能。
ATB_LLM_RAZOR_ATTENTION_ROPE	Rope编码下的RA压缩开关。 ● 0：不开启（默认值） ● 1：开启
MINDIE_LLM_USE_MB_SWAPPER	MindIE LLM高性能Swap开关。 ● 0：不开启（默认值） ● 1：开启
MODELTEST_LOG_FILENAME	MindIE LLM Modeltest日志文件名。
MODELTEST_LOG_LEVEL	MindIE LLM Modeltest日志级别。取值范围："DEBUG"、"INFO"（默认值）、"WARN"、"ERROR"、"CRITICAL"。
MODELTEST_LOG_TO_FILE	MindIE LLM Modeltest日志写入文件开关。 ● 0：不开启 ● 1：开启（默认值）
PERFORMANCE_PREFIX_TREE_ENABLE	memory_decoding并行解码高性能前缀树实现开关。 ● 0：不开启（默认值） ● 1：开启
ENABLE_ACLNN_ATTN_DECODE_BACKEND	Decode阶段attention计算是否使用aclnn后端，默认使用atb后端。 ● 0：使用atb后端（当权重格式为float16时，建议使用atb后端） ● 1：使用aclnn后端（当权重格式为bfloat16时，建议使用aclnn后端）

说明

当BIND_CPU环境变量开启时，会调用execute_command方法执行以下命令：

execute_command(["npu-smi", "info", "-i", f"{npu_id}", "-t", "memory"]).split("\n")
[1:]execute_command(["npu-smi", "info", "-i", f"{npu_id}", "-t", "usages"]).split("\n")
[1:]execute_command(["npu-smi", "info", "-m"]).strip().split("\n")
[1:]execute_command(["npu-smi", "info", "-t", "board", "-i", f"{device_info.npu_id}", -c", f"{device_info.chip_id}"]).strip().split("\n")execute_command(["lspci", "-s", f"{pcie_no}", "-vvv"]).split("\n")execute_command(["lscpu"]).split("\n")

