MindIE
1.0.RC1

MindIE-Service 开发指南

文档版本	01
发布日期	2025-03-25

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 产品简介

产品介绍

MindIE-Service是面向通用模型的推理服务化场景，实现开放、可扩展的推理服务化平台架构，支持对接业界主流推理框架接口，满足大语言模型、文生图等多类型模型的高性能推理需求。它的组件包括MindIE-Server和MindIE-Client，一方面通过对接昇腾的推理加速引擎带来大模型在昇腾环境中的性能提升，另一方面，通过接入现有的主流推理框架生态，逐渐以性能和易用性牵引存量生态的用户向全自研推理服务化平台迁移。

支持的特性

- 支持大模型服务化快速部署，详情请参见**2 快速开始**。
- 提供了标准的昇腾服务化接口，兼容Triton/OpenAI/TGI/vLLM等第三方框架接口，详情请参见**3.3.2 EndPoint提供的RESTful接口**。
- 支持Continuous Batching，PagedAttention。
- 支持基于Transformer推理加速库（Ascend Transformer Boost）的模型接入，继承其加速能力，包括融合加速算子、量化等特性。

2 快速开始

2.1 概述

本章节介绍了如何使用MindIE-Server大语言模型推理服务端，启动兼容多种第三方接口与自研接口的服务。

2.2 安装与部署

请参考《**MindIE安装指南**》中“物理机部署MindIE”章节进行环境的安装与部署，并参考**2.3 参数说明**根据用户需要配置参数和最优性能配置。

2.3 参数说明

2.3.1 配置参数说明

说明

配置项取值请参考《**MindIE安装指南**》中的“物理机部署MindIE > 配置MindIE-Server”章节中的步骤4。

表 2-1 ResourceParam 参数

配置项	取值类型	取值范围	是否必填	配置说明
cacheBlockSize	uint32_t	[1, 128]	必填	kvcache block的size大小。 建议取值：128，其他值建议取为2的n次幂。
preAllocBlocks	uint32_t	[0, maxIterTimes / cacheBlockSize]，maxIterTimes / cacheBlockSize向上取整。	必填	预分配策略：给请求分配好block数量。 如果设置了AIE_LLM_CONTINUOUS_BATCHING环境变量： ● AIE_LLM_CONTINUOUS_BATCHING=1时打开CB，建议配置为maxIterTimes/cacheBlockSize。 ● AIE_LLM_CONTINUOUS_BATCHING=0时关闭CB，建议配置为maxSeqLen/cacheBlockSize。 ● 未设置环境变量时，则默认开启CB。

表 2-2 LogParam 参数

配置项	取值类型	取值范围	默认值	配置说明
logLevel	std::string	● "Verbose" ● "Info" ● "Warning" ● "Error" ● "None"	必填	● "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"：只会打印error级别的日志。 ● "None"：不打印日志。

配置项	取值类型	取值范围	默认值	配置说明
logPath	std::string	文件绝对路径长度<=4096。	必填	日志保存路径。相对路径，代码中会取到工程的绝对路径，最后拼接而成。目前只支持修改日志文件名，即logPath必须配置为"/logs/xxxx"，其中xxxx为日志名称。 例如，假设MindIE-Service的安装路径为“/opt/Ascend-mindie-service_{version}_linux-x86_64/”，则默认的日志绝对路径为“/opt/Ascend-mindie-service_{version}_linux-x86_64/logs/mindservice.log”。

表 2-3 ServeParam 参数

配置项	取值类型	取值范围	默认值	配置说明
ipAddress	std::string	IPv4地址。	必填	EndPoint提供的RESTfull接口绑定的IP地址。不建议绑定为0.0.0.0。
port	int32_t	[1024, 65535]	必填	EndPoint提供的RESTfull接口绑定的端口号。 如果采用物理机/宿主机IP地址通信，请自行保证端口号无冲突。
maxLinkNum	uint32_t	[1, 300]	必填	RESTfull接口请求并发数，EndPoint支持的最大并发请求数。
httpsEnabled	bool	• true • false	必填	是否开启https通信。建议值true，为false时，后续参数忽略。
tlsCaPath	std::string	建议tlsCaPath+tlsCaFile路径长度<=2048。	“httpsEnabled”=“true”，必填	根证书路径。“httpsEnabled”=“true”生效。
tlsCaFile	std::set<std::string>	建议tlsCaPath+tlsCaFile路径长度<=2048。	“httpsEnabled”=“true”，必填	根证书名称列表。["ca.pem"]，“httpsEnabled”=“true”生效。

配置项	取值类型	取值范围	默认值	配置说明
tlsCert	std::string	建议文件路径长度<=2048。	“httpsEnabled”=“true”，必填	服务证书文件路径。“httpsEnabled”=“true”生效。
tlsPk	std::string	建议文件路径长度<=2048。	“httpsEnabled”=“true”，必填	服务证书私钥文件路径。“httpsEnabled”=“true”生效。
tlsPkPwd	std::string	文件路径长度<=2048。	“httpsEnabled”=“true”，必填	服务证书私钥加密密钥文件路径。“httpsEnabled”=“true”生效。
kmcKsMaster	std::string	建议文件路径长度<=2048。	“httpsEnabled”=“true”，必填	KMC密钥库文件路径。“httpsEnabled”=“true”生效。
kmcKsStandby	std::string	建议文件路径长度<=2048。	“httpsEnabled”=“true”，必填	KMC密钥库备份文件路径。“httpsEnabled”=“true”生效。
tlsCrl	std::string	建议文件路径长度<=2048。	“httpsEnabled”=“true”，必填	服务证书吊销列表文件路径。“httpsEnabled”=“true”生效。

说明

如果推理服务所在的计算节点的网络为跨公网和局域网，绑定0.0.0.0的IP地址可能导致网络隔离失效，存在较大安全风险。故该场景下不建议EndPoint的IP地址绑定为0.0.0.0。

表 2-4 TemplateParam 参数

配置项	取值类型	取值范围	默认值	配置说明
templateType	std::string	"Standard"	必填	普通推理。
templateName	std::string	-	必填	工作流名称。
pipelineNumber	uint32_t	[1, 10]	必填	工作流数量。

其中，对于每一种模型，其全局参数配置参见表2-5。

表 2-5 ModelDeployParam 参数

参数	取值类型	取值范围	是否必填	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。	必填	最大序列长度。输入的长度+输出的长度<=maxSeqLen，用户根据自己的推理场景选择maxSeqLen。
npuDeviceIds	std::set<size_t>	根据模型和环境的实际情况来决定。	必填	启用哪几张卡。对于每个模型实例分配的npuIds。
ModelParam				
modelInstanceType	std::string	• "Standard" • "StandardMock"	选填，默认值为"Standard"	模型类型。 • 普通推理：modelInstanceType="Standard" • 假模型：modelInstanceType="StandardMock"
modelName	std::string	-	必填	模型名称。
modelWeightPath	std::string	文件绝对路径长度<=2048。	必填	模型权重路径。路径必需存在。
worldSize	uint32_t	根据模型实际情况来决定。	必填	启用几张卡推理。目前llama-65b至少启用四张NPU卡。
cpuMemSize	uint32_t	上限根据显存和用户需求来决定。	必填	CPU中可以用来申请kv cache的size上限。单位：GB。建议值：5。当“supportSwapPolicy”=“false”，可以配置为0。
npuMemSize	uint32_t	上限根据显存和用户需求来决定。	必填	NPU中可以用来申请kv cache的size上限。单位：GB。建议值：8。 快速计算公式：npuMemSize=(总空闲-权重/NPU卡数-后处理占用)*系数，其中系数取0.8。

参数	取值类型	取值范围	是否必填	配置说明
backendType	std::string	• "atb" • "mindformer" • "torchair" • "default" • ""	必填	当前对接后端类型，大小写不敏感。

表 2-6 ScheduleParam 参数

配置项	取值类型	取值范围	默认值	配置说明
maxPrefillBatchSize	uint32_t	[1, maxBatchSize]	必填	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。
maxPrefillTokens	uint32_t	[4096, 409600]	必填	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。
prefillTimeMsPerReq	uint32_t	[0, 1000]	必填	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。
prefillPolicyType	uint32_t	• 0 • 1 • 3	必填	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。 • 3：MLFQ，多级反馈队列。 其中，3是0/1的组合。

配置项	取值类型	取值范围	默认值	配置说明
decodeTimeMsPerReq	uint32_t	[0, 1000]	必填	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。
decodePolicyType	uint32_t	• 0 • 1 • 3	必填	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。 • 3：MLFQ，多级反馈队列。 其中，3是0/1的组合。
maxBatchSize	uint32_t	[1, 5000]	必填	最大decode batch size。 1. 首先计算block_num：Total Block Num = Floor(NPU显存/(模型网络层数*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。 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。 3. maxBatchSize=Total Block Num/Block Num。
maxIterTimes	uint32_t	[1, maxSeqLen-1]	必填	最大迭代次数，即一句话最大可生成长度。请求级别里面有一个max_output_length参数，maxIterTimes是一个全局设置，与max_output_length取小作为最终output的最长length。

配置项	取值类型	取值范围	默认值	配置说明
maxPreemptCount	int32_t	[0, maxBatchSize]	必填	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。
supportSelectBatch	bool	• true • false	必填	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。
maxQueueDelayMicroseconds	uint32_t	[500, 1000000]	必填	队列等待时间，单位：us。

说明

- “npuDeviceIds”参数使用规则：
  1. 先执行下列命令，设置设备上加速库可见的卡。
     ```
     export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
     ```
  2. 然后“npuDeviceIds”表示在上述可见的卡中，按ASCEND_RT_VISIBLE_DEVICES设置的顺序，选取对应的卡。

  例如：
  - ASCEND_RT_VISIBLE_DEVICES=7,0,1,2,3,4,5,6
  - npuDeviceIds：[[0,1,3,4]]

    则最终分配的卡为7,0,2,3。
- “npuMemSize”参数使用规则：
  - 为快速确定最佳显存参数取值范围，提供了快速计算公式。该公式计算的结果仅作为取值参考，为了达到最佳性能，可以适当向上调整该值并进行性能压力测试。
  - 如果设置的“npuMemSize”参数超过系统可分配最大显存值，会出现推理服务启动失败、推理服务启动卡死等异常现象，需要减小该值并重试。

2.3.2 性能调优

最优性能参数配置

配置类型	配置项	最优配置
调度配置	maxPrefillBatchSize	• 使用maxPrefillTokens/数据集平均输入长度，得到maxPrefillBatchSize。 • 替代方案：maxBatchSize的一半（可微调）。
	maxPrefillTokens	• 卡的理想输入token数（吞吐最大）。 • 替代方案：maxPrefillBatchSize*70（可微调。如果允许数据泄漏，70可以设置成数据集平均输入长度）。
	maxBatchSize	表2-6中根据公式计算出的最大值。
	maxPreemptCount	10。
	supportSelectBatch	true
模型配置	npuDeviceIds	[0, 1, 2, ..., worldSize-1]
	worldSize	节点的最大可用卡数。
	npuMemSize	表2-5中根据公式计算出的最大值，单位GB。
资源配置	preAllocBlocks	表2-1范围内适当调小。理论上为ceil（数据集平均输出长度/cacheBlockSize）最佳，但当前不开满可能会卡死。

maxBatchSize 调优指导

在mindservice.log日志中，过滤测试时间段对应的“COMPLETED REQ ID”关键字如下：

```
...COMPLETED REQ ID:  363 , 5   , 593 , 81  , 512 , 12  , 60  , 1
...COMPLETED REQ ID:  377 , 5   , 237 , 87  , 150 , 12  , 60  , 1
```

- 前五列代表当前batch在本次decode推理完成的第一个请求的基本信息。
- 后三列代表本次decode推理的batch的基本信息。

以第一行日志为例，每一列数据的含义如表2-7所示。

表 2-7 第一行日志“COMPLETED REQ ID”关键字各列数据含义

363	5	593	81	512	12	60	1
请求ID。	请求使用的Block Num。	总序列长度。	输入序列长度。	输出序列长度。	当前BatchSize大小。	当前batch使用的Block Num。	本次推理完成的请求个数。

性能的最优值就是根据第7列的NpuBlock使用个数和Total Block Num个数之间的关系调整BatchSize，具体调整策略如下：

- 如果在整个过程中，Block Num一直低于Total Block Num，那么说明BatchSize不足，适当调大。
- 如果在整个过程中，Block Num在部分请求段大于或等于Total Block Num，那么说明BatchSize过大，直接调小到该段对应的BatchSize附近。

操作步骤

这里以llama-65b模型为例进行最优性能的配置，环境信息举例如下：

- 本机显存大小为64G的卡，NPU数量为8卡，以已占用3G为例。
- 由于后处理在当前版本默认开启，以占用2G为例；如需关闭后处理，请参见如下方法，关闭后则不需要参与计算。

  使用vim命令打开/usr/local/Ascend/mindie/latest/mindie-service/latest/bin/ibis目录中的model_inputs.py文件，注释第61-64行即可关闭后处理：

  ```
  # elif input_tensor.name() == "SAMPLING":
  #     has_sampling = True
  #     if is_prefill:
  #         sampling_param = np.array(input_tensor, copy=False)[0]
  ```

步骤1 请参考表2-5中的“npuMemSize”参数计算出npuMemSize的值并根据计算结果调整配置中的该值，计算过程如下所示。

1. 计算模型的权重大小，计算公式为：参数量*tensor字节数（这两个值请用户根据模型自行获取）。

   llama-65b模型的参数量为650亿，tensor字节数为2，通过计算公式得出llama-65b的权重约为130G。

2. 计算npuMemSize的值，计算公式为：Floor[(总空闲-权重/NPU卡数-后处理占用)* 系数]，系数取值为0.8。

   npuMemSize = Floor[(64 - 3 - 130/8 - 2)] * 0.8 = 34G。

   说明

   “Floor”表示计算结果向下取整。

步骤2 请参考表2-6中的“maxBatchSize”参数计算出maxBatchSize的值并根据计算结果调整配置中的该值，计算过程如下所示。

根据计算公式：maxBatchSize = Total Block Num/Block Num，需要先计算出Total Block Num和Block Num的值。

1. 计算Total Block Num的值。

根据计算公式：Total Block Num = Floor[NPU显存/(模型网络层数*Block Size*模型注意力头数*注意力头大小*Cache类型字节数*Cache数)]，公式中各参数的取值信息如表2-8所示。

表 2-8 Total Block Num 公式中的参数值

参数	取值
NPU显存	步骤1中npuMemSize的值。
模型网络层数	模型网络层数是模型权重文件config.json中的hidden_layers值，取值为：80。
Block Size	默认值：128。
模型注意力头数	“模型注意力头数*注意力头大小”的值为模型权重文件中config.json中的hidden_size值。 **说明** 对于llama2模型，由于是GQA类模型（分组查询注意力类模型），需使用模型权重文件config.json中num_key_value_heads的值作为注意力头数，而不是num_attention_heads。所以每张卡上的“模型注意力头数*注意力头大小”的值不一定是用hidden_size/卡数。例如llama2-70b的“模型注意力头数*注意力头大小”的值应该为8*128=1024，而不是hidden_size中的值：8192。
注意力头大小	
Cache类型字节数	由模型config.json文件中的torch.dtype决定，一般为float16类型，取值为：2。
Cache数	默认值：2。

将以上参数值代入公式，得到Total Block Num=Floor[34*1024*1024*1024/(80 * 128 * 128 * 64/8 * 2 * 2)] = 870。

注：以上算式中64/8是由于有8张NPU卡，所以注意头数需均分在8张卡上。

说明

Total Block Num的值也可以通过跑一次性能后在python日志中的npuBlockNum获取，详情请参考•吞吐量最大化。

2. 计算Block Num的值。

根据计算公式：Block Num = Ceil(输入Token数/Block Size)+Ceil(最大输出Token数/Block Size)，公式中各参数的取值信息如表2-9所示。

表 2-9 Block Num 公式中的参数值

参数	取值
输入Token数	首次一般参考数据集的平均输入，取值100。 第二次则可以直接取返回结果中的average_input_length的值。
最大输出Token数	从config.json文件中的maxIterTimes参数获取，取值：512。

参数	取值
Block Size	默认值：128。

将以上参数值代入公式，得到Block Num = Ceil(100/128)+Ceil(512/128) = 5。

说明

“Ceil”表示计算结果向上取整。

3. 根据**步骤2.1**和**步骤2.2**计算出的Total Block Num和Block Num值，然后使用公式maxBatchSize=Total Block Num/Block Num得到maxBatchSize的值。

   maxBatchSize = 870/5 = 174。

步骤3 根据**步骤2**中maxBatchSize的值调整maxPrefillBatchSize和maxPrefillTokens的值，并将supportSelectBatch参数设置为“true”，详情请参考**最优性能参数配置**。

preAllocBlocks的值根据ceil(数据集平均输出长度/cacheBlockSize) 进行计算，约为1。

步骤4 根据本机环境和选用的模型需要调整npuDeviceIds和worldSize的值，详情请参考**最优性能参数配置**。进行第一次调试，调试方法请参见**3.4.3 llm_engine_test**。

说明

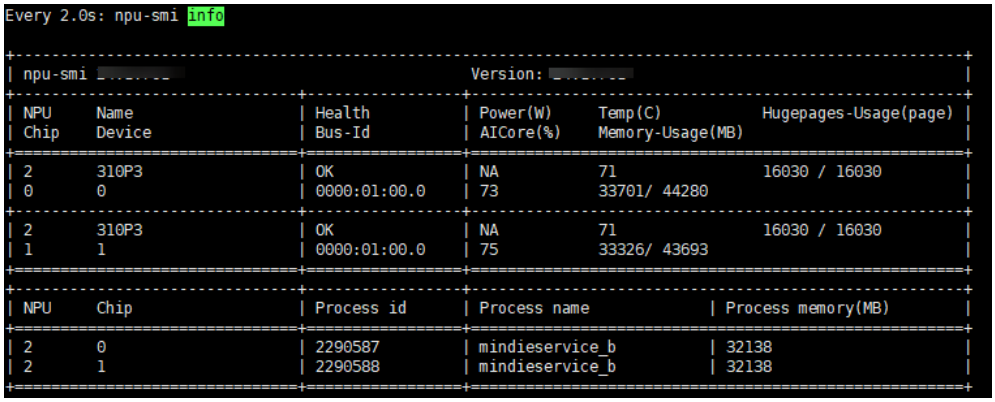
数据集如需要转换请参见**A.2 数据集使用**章节。

步骤5 调整npuMemSize，在调试过程中，重开一个窗口使用以下命令查看占卡情况，如果剩余空间还很大，可以调大npuMemSize的值，重复**步骤2**~**步骤4**后再次进行调试。

```
watch npu-smi info
```

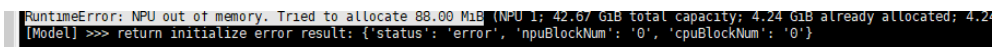
- 当npuMemSize的值不够大时，可以继续调大空闲值，如**图2-1**。

图 2-1 npuMemSize 值不够大



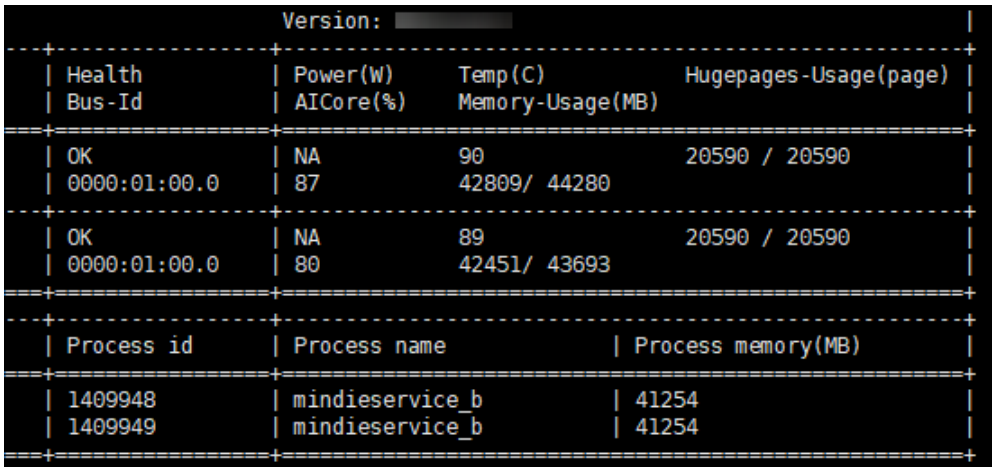
- 当npuMemSize的值过大时，则会报“Npu out of memory”错误，如**图2-2**所示。

图 2-2 npuMemSize 过大的情况



- 根据“Npu out of memory”报错信息，将npuMemSize的值调小（比如在原来的值上减2，避免卡死），则可以达到最优npuMemSize的值。如图2-3，大概是占据95%卡的状态。

图 2-3 npuMemSize 最优值



这时从性能返回结果可以看出来lpot会优于配置之前的值，并且generate speed也有很大的提升。

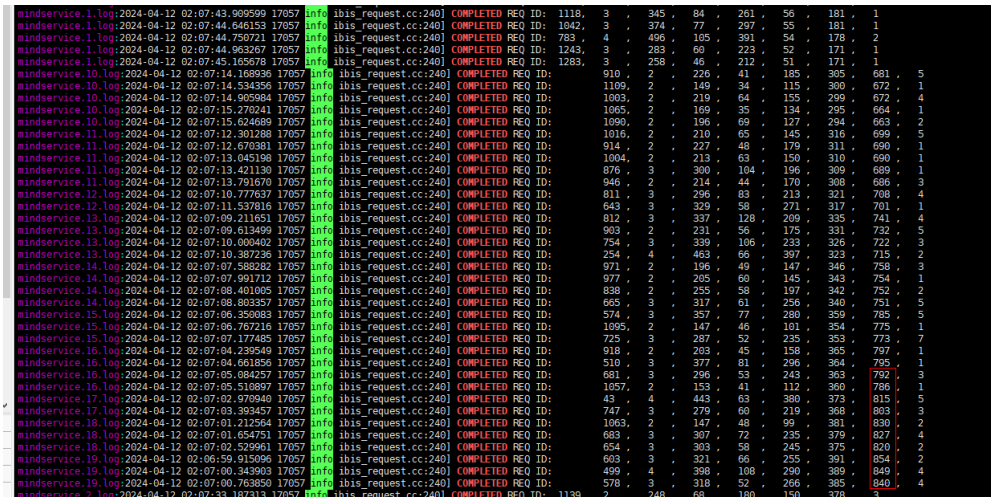
步骤6 根据用户需求通过以下方式进行吞吐量最大化的配置或者非首token时延（lpot）最小化的配置。

- 吞吐量最大化

  通过调整maxBatchSize的值使吞吐量最大化。首先根据结果日志（logs/mindservice.logs）使用以下命令查看最大Total Block Num的值，如图2-4所示。

```
grep "COMPLETED" mindservice*.log
```

图 2-4 最大 Total Block Num 值

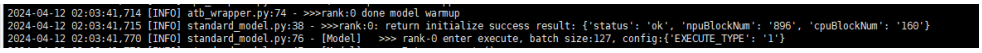


从上图中查看Total Block Num的峰值为854，比计算得到的Total Block Num值小。即可适当调大maxBatchSize的值（同步调整对应的maxPrefillBatchSize、maxPrefillTokens和maxPreemptCount的值）再进行一次性能操作。

说明

也可以通过将set_env.sh环境变量中打开Python日志（export MIES_PYTHON_LOG_TO_FILE=1），并在/workspace/log/pythonlog.log.*xxxxx*下对应日志中通过搜索npuBlockNum找到，如图2-5所示，值为896。

图 2-5 python 日志中的 npuBlockNum 显示



- 非首token时延（lpot）最小化

  为了让非首token时延（lpot）最小化，则需要调小maxBatchSize来达到最优性能，暂时只能通过阶梯调整。

  根据步骤3对maxBatchSize进行调整，并调整对应的maxPrefillBatchSize、maxPrefillTokens和maxPreemptCount的值，对比哪个值可以达到要求。

  说明

  maxPrefillTokens的值不能小于4096。

  比如baichuan2-7b模型在Atlas 300I Duo 推理卡上进行的测试，lpot要求是小于80ms，这时maxBatchSize的值选择87时可以得到最小化的时延要求以及该时延下最优吞吐量。

表 2-10 maxBatchSize 对应的 lpot 和 generate speed

maxBatchSize	lpot（时延）	generate speed（吞吐量）
200	160	1145
100	88	1074
90	81	1058
89	80.4	1058
88	80.4	1060
87	79.7	1032

----结束

2.4 启动服务与接口使用

2.4.1 启动服务

MindIE-Server可以部署为兼容Triton/OpenAI/TGI/vLLM等第三方框架接口的服务应用。推荐用户开启HTTPS通信，并按照《**MindIE安装指南**》中的“物理机部署MindIE > 配置MindIE Server”章节配置开启https通信所需服务证书、私钥等设置。服务器默认启动在https://127.0.0.1:1025，用户可通过在config.json文件下修改ipAddress和port参数来自定义启动IP地址与端口号。目前MindIE-Server可实现服务状态查询，模型信息查询，文本/流式推理等功能。

步骤1 使用以下命令启动服务，以当前所在Ascend-mindie-service_*{version}*_linux-*{arch}*目录为例。

```
./bin/mindieservice_daemon
```

回显如下则说明启动成功。

```
Daemon start success!
```

步骤2 用户可使用HTTP客户端（Linux curl命令，Postman工具等）发送HTTP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。例如列出当前模型列表：

```
curl -H "Accept: application/json" -H "Content-type: application/json" --cacert ca.pem --cert client.pem  --key client.key.pem -X GET https://127.0.0.1:1025/v1/models
```

说明

- --cacert：验签证书文件路径。
- ca.pem为MindIE-Server服务端证书的验签证书/根证书。
- --cert： 客户端证书文件路径。
- client.pem为客户端证书。
- --key：客户端私钥文件路径。
- client.key.pem为客户端证书私钥（未加密，建议采用加密密钥）。

请用户根据实际情况对相应参数进行修改。

----结束

2.4.2 使用接口说明

2.4.2.1 使用兼容 TGI 0.9.4 版本的接口

- 文本推理接口：

```
curl -H "Accept: application/json" -H "Content-type: application/json" --cacert ca.pem --cert client.pem  --key client.key.pem -X POST -d '{
  "inputs": "My name is Olivier and I",
  "parameters": {
    "decoder_input_details": true,
    "details": true,
    "do_sample": true,
    "max_new_tokens": 20,
    "repetition_penalty": 1.03,
    "return_full_text": false,
    "seed": null,
    "temperature": 0.5,
    "top_k": 10,
    "top_p": 0.95,
    "truncate": null
  }
}' https://127.0.0.1:1025/generate
```

- 流式推理接口：

```
curl -H "Accept: application/json" -H "Content-type: application/json" --cacert ca.pem --cert client.pem  --key client.key.pem -X POST -d '{
  "inputs": "My name is Olivier and I",
  "parameters": {
    "decoder_input_details": true,
    "details": true,
    "do_sample": true,
    "max_new_tokens": 20,
    "repetition_penalty": 1.03,
```



```
    "return_full_text": false,
    "seed": null,
    "temperature": 0.5,
    "top_k": 10,
    "top_p": 0.95,
    "truncate": null
  }
}' https://127.0.0.1:1025/generate_stream
```

其他接口请参见**3.3.2.1 兼容TGI 0.9.4版本接口**章节。

2.4.2.2 使用兼容 vLLM 0.2.6 版本接口

文本/流式推理接口，将请求体中的stream参数改为false即为文本推理，改为true即为流式推理：

```
curl -H "Accept: application/json" -H "Content-type: application/json" --cacert ca.pem --cert client.pem  --
key client.key.pem -X POST -d '{
    "prompt": "My name is Olivier and I",
    "max_tokens": 20,
    "repetition_penalty": 1.03,
    "presence_penalty": 1.2,
    "frequency_penalty": 1.2,
    "temperature": 0.5,
    "top_k": 10,
    "top_p": 0.95,
    "stream": false
}' https://127.0.0.1:1025/generate
```

其他接口请参见**3.3.2.2 兼容vLLM 0.2.6版本接口**章节。

2.4.2.3 使用兼容 OpenAI 接口

推理接口，将请求体中的stream参数改为false即为文本推理，改为true即为流式推理：

```
curl -H "Accept: application/json" -H "Content-type: application/json" --cacert ca.pem --cert client.pem  --
key client.key.pem -X POST -d '{
 "model": "gpt-3.5-turbo",
 "messages": [{
  "role": "system",
  "content": "You are a student who is good at math."
    },
    {
     "role": "user",
     "content": "what is your hobby?"
    }
],
 "max_tokens": 20,
 "presence_penalty": 1.03,
 "frequency_penalty": 1.0,
 "seed": null,
 "temperature": 0.5,
 "top_p": 0.95,
 "stream": false
}' https://127.0.0.1:1025/v1/chat/completions
```

其他接口请参见**3.3.2.3 兼容OpenAI接口**章节。

2.4.2.4 使用兼容 Triton 接口

- Token推理接口：

  ```
  curl -H "Accept: application/json" -H "Content-type: application/json" --cacert ca.pem --cert
  client.pem  --key client.key.pem -X POST -d '{
   "id": "42",
   "inputs": [{
    "name": "input0",
  ```



```
 "shape": [
  1,
  10
 ],
 "datatype": "UINT32",
 "data": [
  396, 319, 13996, 29877, 29901, 29907, 3333, 20718, 316, 23924
 ],
 "parameters": {
  "temperature": 0.5,
  "top_k": 10,
  "top_p": 0.95,
  "do_sample": true,
  "seed": null,
  "repetition_penalty": 1.03,
  "max_new_tokens": 512
 }
}],
"outputs": [{
 "name": "output0"
}]
}' https://127.0.0.1:1025/v2/models/llama_65b/infer
```

- 文本推理接口：

```
curl -H "Accept: application/json" -H "Content-type: application/json" --cacert ca.pem --cert
client.pem  --key client.key.pem -X POST -d '{
  "id":"a123",
  "text_input": "My name is Olivier and I",
  "parameters": {
    "details": true,
    "do_sample": true,
    "max_new_tokens":200,
    "repetition_penalty": 1.1,
    "seed": 123,
    "temperature": 1,
    "top_k": 2147483647,
    "top_p": 0.99,
    "batch_size":100
  }
}' https://127.0.0.1:1025/v2/models/llama_65b/generate
```

- 流式推理接口：

```
curl -H "Accept: application/json" -H "Content-type: application/json" --cacert ca.pem --cert
client.pem  --key client.key.pem -X POST -d '{
  "id":"a123",
  "text_input": "My name is Olivier and I",
  "parameters": {
    "details": true,
    "do_sample": true,
    "max_new_tokens":200,
    "repetition_penalty": 1.1,
    "seed": 123,
    "temperature": 1,
    "top_k": 2147483647,
    "top_p": 0.99,
    "batch_size":100
  }
}' https://127.0.0.1:1025/v2/models/llama_65b/generate_stream
```

其他接口请参见**3.3.2.3 兼容OpenAI接口**章节。

也可以使用MindIE Client的Python接口来进行推理，例如文本推理。

首先需要用户创建一个MindIE Client，将该文件命名为utils.py，后续可持续使用该方法。

```
import argparse
from mindieclient.python.httpclient import MindIEHTTPClient
```



```
def create_client(request_count=1):
    # get argument
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-u",
        "--url",
        required=False,
        default="https://127.0.0.1:1025",
        help="MindIE-Server URL.",
    )
    parser.add_argument(
        "-v",
        "--verbose",
        action="store_true",
        required=False,
        default=True,
        help="Enable detailed information output.",
    )
    parser.add_argument(
        "-s",
        "--ssl",
        action="store_true",
        required=False,
        default=False,
        help="Enable encrypted link with https",
    )
    parser.add_argument(
        "-ca",
        "--ca_certs",
        required=False,
        default="ca.pem",
        help="Provide https ca certificate.",
    )
    parser.add_argument(
        "-key",
        "--key_file",
        required=False,
        default="client.key.pem",
        help="Provide https client certificate.",
    )
    parser.add_argument(
        "-cert",
        "--cert_file",
        required=False,
        default="client.pem",
        help="Provide https client keyfile.",
    )
    args = parser.parse_args()
    # create client
    try:
        if args.ssl:
            ssl_options = {}
            if args.ca_certs is not None:
                ssl_options["ca_certs"] = args.ca_certs
            if args.key_file is not None:
                ssl_options["keyfile"] = args.key_file
            if args.cert_file is not None:
                ssl_options["certfile"] = args.cert_file
            mindie_client = MindIEHTTPClient(
                url=args.url,
                verbose=args.verbose,
                enable_ssl=True,
                ssl_options=ssl_options,
                concurrency=request_count,
            )
        else:
            mindie_client = MindIEHTTPClient(
                url=args.url, verbose=args.verbose, concurrency=request_count
            )
```



```
    except Exception as e:
        raise e
    return mindie_client
```

之后创建文件，调用上述create_client方法，即可调用文本推理接口。

```
from utils import create_client
if __name__ == "__main__":
    # get argument and create client
    try:
        mindie_client = create_client()
    except Exception as e:
        print("Client Creation falied!")
        sys.exit(1)
    # create input
    prompt = "My name is Olivier and I"
    model_name = "llama_65b"
    parameters = {
        "do_sample": True,
        "temperature": 0.5,
        "top_k": 10,
        "top_p": 0.9,
        "truncate": 5,
        "typical_p": 0.9,
        "seed": 1,
        "repetition_penalty": 1,
        "watermark": True,
        "details": True,
    }
    # apply model inference
    result = mindie_client.generate(
        model_name,
        prompt,
        request_id="1",
        parameters=parameters,
    )
    print(result.get_response())
```

其他MindIE Client接口请参见**4.3.1.2 class MindIEHTTPClient**章节。

2.4.2.5 使用自研接口

文本/流式推理接口，将请求体中的stream参数改为false即为文本推理，改为true即为流式推理：

```
curl -H "Accept: application/json" -H "Content-type: application/json" --cacert ca.pem --cert client.pem  --
key client.key.pem -X POST -d '{
 "inputs": "My name is Olivier and I",
 "stream": true,
 "parameters": {
  "temperature": 0.5,
  "top_k": 10,
  "top_p": 0.95,
  "do_sample": true,
  "seed": null,
  "repetition_penalty": 1.03,
  "details": true
 }
}' https://127.0.0.1:1025/infer
```

其他接口请参见**3.3.2.5 自研接口**章节

创建MindIE Client的方法请参见**2.4.2.4 使用兼容Triton接口**，之后可使用MindIE Client的Python接口来提前终止请求。

```
from utils import create_client
if __name__ == "__main__":
```

```
# get argument and create client
mindie_client = create_client()
# create input
prompt = "My name is Olivier and I"
model_name = "llama_65b"
parameters = {
    "do_sample": True,
    "temperature": 0.5,
    "top_k": 10,
    "top_p": 0.9,
    "truncate": 5,
    "typical_p": 0.9,
    "seed": 1,
    "repetition_penalty": 1,
    "watermark": True,
    "details": True,
}
# apply model inference
results = mindie_client.generate_stream(
    model_name,
    prompt,
    request_id="1",
    parameters=parameters,
)
# stop early
generated_text = ""
index = 0
for cur_res in results:
    index += 1
    if index == 10:
        flag = mindie_client.cancel(model_name, "1")
        if flag:
            print("Test cancel api succeed!")
            sys.exit(0)
        else:
            print("Test cancel api failed!")
            sys.exit(1)
    print("current result: %s", cur_res)
```

其他MindIE Client接口请参见**4.3.1.2 class MindIEHTTPClient**章节。

3 MindIE Server

3.1 功能介绍

面向通用模型的推理服务化场景，实现开放、可扩展的推理服务化平台架构，支持对接业界主流推理框架接口，满足大语言模型、文生图等多类型模型的高性能推理需求。主要包括以下特性：

- 服务启动：Daemon负责推理服务启动，加载配置文件，初始化其他模块。
- 北向接口：ServerEndpoint面向推理服务开发者（包括华为研发和合作伙伴及有开发能力的用户）提供极简易用的API接口，支持TGI、vLLM、OpenAI等主流推理框架请求接口。
- 统一推理API：统一推理API提供模型初始化、推理请求处理、服务/模型信息查询接口，ServerEndpoint和其他三方框架调用统一推理API使能推理服务。
- 推理服务化平台：GMIS模块支持推理流程的工作流定义扩展，以工作流为驱动，实现从推理任务调度到任务执行的可扩展架构，适应各类推理方法如投机推理、LoRA推理、LLMA、Prompt增强等的快速落地。
- 南向接口：ModelWrapper模块面向不同推理引擎，不同模型，提供统一抽象接口，便于扩展，减少推理引擎、模型变化带来的修改。

3.2 使用指导

使用限制

须知

使用Atlas 300I Duo卡的推理服务场景，推理请求的后处理参数不支持top_k。如果并发发送推理（包括EndPoint提供的RESTfull接口和Engine提供的Forward接口），并且部分推理请求的后处理参数设置了top_k，部分请求后处理参数不设置top_k，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。

场景说明

1. MindIE Server提供EndPoint模块对推理服务化协议和接口封装，兼容Triton/OpenAI/TGI/vLLM等第三方框架接口。使用单节点安装模式安装MindIE Server之后，用户使用http/https客户端（Linux curl命令，Postman工具等）发送http/https请求，即可调用EndPoint提供的接口。
2. MindIE Server通过Engine模块提供C++接口给客户，客户可以基于C++接口做二次集成开发。

Endpoint RESTfull 接口使用说明

http/https请求的URL的IP地址和端口号在config.json中进行配置，详情请参见**表2-3**。

- 以Linux curl工具发送generate请求，URL请求格式如下：
  - 操作类型：**POST**
  - **URL：http*[s]*://*{ip}*:*{port}*/generate**
- 未开启https，发送推理请求：

```
curl -H "Accept: application/json" -H "Content-type: application/json" -X POST -d '{
  "inputs": "My name is Olivier and I",
  "parameters": {
    "details": true,
    "do_sample": true,
    "repetition_penalty": 1.1,
    "return_full_text": false,
    "seed": null,
    "temperature": 1,
    "top_n_tokens": 5,
    "top_p": 0.99
  },
  "stream": false
}' http://{ip}:{port}/generate
```

- https双向认证的请求方式示例：

```
curl --location --request POST 'https://{ip}:{port}/generate' \
--header 'Content-Type: application/json' \
--cacert /home/runs/static_conf/ca/ca.pem \
--cert /home/runs/static_conf/cert/client.pem \
--key /home/runs/static_conf/cert/client.key.pem \
--data-raw '{
    "inputs": "My name is Olivier and I",
    "parameters": {
        "best_of": 1,
        "decoder_input_details": false,
```



```
        "details": false,
        "do_sample": true,
        "max_new_tokens": 20,
        "repetition_penalty": 2,
        "return_full_text": false,
        "seed": 12,
        "stop": [
            "photographer"
        ],
        "temperature": 0.1,
        "top_k": 1,
        "top_n_tokens": 5,
        "top_p": 0.9,
        "truncate": 1024,
        "typical_p": 0.95,
        "watermark": true
    },
    "stream": true
}'
```

说明

- --cacert：验签证书文件路径。
- ca.pem为MindIE-Server服务端证书的验签证书/根证书。
- --cert： 客户端证书文件路径。
- client.pem为客户端证书。
- --key：客户端私钥文件路径。
- client.key.pem为客户端证书私钥（未加密，建议采用加密密钥）。

请用户根据实际情况对相应参数进行修改。

提供的RESTful API列表如下：

表 3-1 服务状态查询 API

API	接口类型	URL	说明	支持框架
Server Live	GET	/v2/health/live	检查服务器是否在线。	Triton
Server Ready	GET	/v2/health/ready	检查服务器是否准备。	Triton
Model Ready	GET	/v2/models/${MODEL_NAME}[/versions/${MODEL_VERSION}]/ready	检查模型是否准备。	Triton
health	GET	/health	服务健康检查。	TGI/vLLM
查询TGI EndPoint信息	GET	/info	查询TGI EndPoint信息。	TGI

API	接口类型	URL	说明	支持框架
slot统计	GET	/v2/models/${MODEL_NAME}[/versions/${MODEL_VERSION}]/getSlotCount	参考Triton格式，自定义的slot统计信息查询接口。	华为自研

表 3-2 管理 API

API	接口类型	URL	说明	支持框架
models列表	GET	/v1/models	列举当前可用模型列表。	OpenAI
model详情	GET	/v1/models/{model}	查询模型信息。	OpenAI
服务元数据查询	GET	/v2	获取服务元数据。	Triton
模型元数据查询	GET	/v2/models/${MODEL_NAME}[/versions/${MODEL_VERSION}]	查询模型元数据信息。	Triton
查询模型配置	GET	/v2/models/${MODEL_NAME}[/versions/${MODEL_VERSION}]/config	查询模型配置。	Triton

表 3-3 推理 API

API	接口类型	URL	说明	支持框架
推理任务	POST	/	TGI推理接口，stream==false返回文本推理结果，stream==true返回流式推理结果。	TGI
	POST	/generate	TGI和vLLM的推理接口，通过请求参数来区分是哪种服务的接口。	TGI/vLLM

API	接口类型	URL	说明	支持框架
	POST	/generate_stream	TGI流式推理接口，使用Server-Sent Events格式返回结果。	TGI
	POST	/v1/chat/completions	OpenAI文本推理接口。	OpenAI
	POST	/infer	华为自研推理接口，支持文本/流式返回结果。	华为自研
	POST	/v2/models/${MODEL_NAME}[/versions/${MODEL_VERSION}]/infer	Triton的token推理接口。	Triton
	POST	/v2/models/${MODEL_NAME}[/versions/${MODEL_VERSION}]/stopInfer	参考Triton接口定义，提供提前终止请求接口。	华为自研
	POST	/v2/models/${MODEL_NAME}[/versions/${MODEL_VERSION}]/generate	Triton文本推理接口。	Triton
	POST	/v2/models/${MODEL_NAME}[/versions/${MODEL_VERSION}]/generate_stream	Triton流式推理接口。	Triton

Engine 模块提供 C++接口

使用Engine模块提供C++接口，需要开发代码来集成，以下代码提供接口使用样例，仅供参考。

```
/*
 * Copyright (c) Huawei Technologies Co., Ltd. 2023-2023. All rights reserved.
 */

#include <iostream>
#include <memory>
#include <map>
#include <thread>
#include <algorithm>
#include <wait.h>

#include "infer_engine.h"
#include "infer_request.h"

#include "metrics.h"
```



```
#include "data_loader.h"
#include "util.h"
#include "io_manager.h"

using namespace SimpleLLMInference;
using SC = std::chrono::steady_clock;

IOManager g_Manager;
Statistics g_Statistics;
std::map<std::string, Metrics> g_Metrics;
volatile int g_CompleteNum = false;

std::mutex g_Mutex;
std::mutex g_MetricsMutex;
static std::mutex g_ExitMtx;
static bool g_ProcessExit = false;

bool g_RecordOutput = false;

namespace SimpleLLMInference {
/**
 *
 * @param response
 */
Status ParseEosAttr(std::shared_ptr<InferenceResponse> &response, int64_t *flag, int64_t *outputLen)
{
    InferenceResponse::Output *output;
    auto status = response->ImmutableOutput("IBIS_EOS_ATTR", &output);
    if (!status.IsOk()) {
        return status;
    }

    auto *eosData = static_cast<int64_t *>(output->Buffer());
    *flag = eosData[0];
    *outputLen = eosData[1];
    return Status(infrastructure::Error::Code::OK, "Success");
}

/**
 * 解析返回的token id
 * @param response
 */
Status ParseOutputId(std::shared_ptr<InferenceResponse> &response, std::vector<int64_t> &outputIds)
{
    InferenceResponse::Output *output;
    auto status = response->ImmutableOutput("OUTPUT_IDS", &output);
    if (!status.IsOk()) {
        return status;
    }
    if (outputIds.empty()) {
        outputIds.reserve(128);
    }
    // 获取输出长度
    auto len = output->Shape()[0];
    auto *data = static_cast<int64_t *>(output->Buffer());
    for (int i = 0; i < len; ++i) {
        outputIds.push_back(data[i]);
    }
    return Status(infrastructure::Error::Code::OK, "Success");
}

/**
 * 请求回调
 * @param response
 */
void ResponseCallback(std::shared_ptr<InferenceResponse> &response)
{
    auto reqId = response->GetRequestId().StringValue();
    size_t decodeTime;
```



```
    auto now = SC::now();
    g_Manager.SetOutputData(reqId);

    {
        std::unique_lock lock(g_MetricsMutex);

        // 生成token数
        int64_t flag;
        int64_t outputLen;
        auto ret = ParseEosAttr(response, &flag, &outputLen);
        if (!ret.IsOk()) {
            std::cout << "ReqId:" << reqId << ", Error:" << ret.StatusMsg() << std::endl;
            return;
        }
        g_Metrics[reqId].tokensOutput += outputLen;

        if (g_Metrics[reqId].firstTokenCost == 0) {
            // prefill 记录首token时间
            decodeTime = GetDuration(now, g_Metrics[reqId].startingTime);
            g_Metrics[reqId].firstTokenCost = decodeTime;
        } else {
            // decode 记录每次decode的时间
            decodeTime = GetDuration(now, g_Metrics[reqId].lastTokenTime);
            // 针对投机场景适配，decode返回小于等于gamma个token，四舍五入
            auto avgDecodeTime = (decodeTime + outputLen / 2) / outputLen;
            for (int i = 0; i < outputLen; ++i) {
                g_Metrics[reqId].decodeTime.push_back(avgDecodeTime);
            }
        }
        g_Metrics[reqId].lastTokenTime = now;

        // 生成token id
        if (g_RecordOutput) {
            ret = ParseOutputId(response, g_Metrics[reqId].outputTokenIds);
            if (!ret.IsOk()) {
                std::cout << "ReqId:" << reqId << ", Error:" << ret.StatusMsg() << std::endl;
                return;
            }
        }

        if (response->IsEOS()) {
            g_Metrics[reqId].endingTime = now;
            // 最后一个Token耗时
            g_Metrics[reqId].lastTokenCost = decodeTime;
        }
    }

    if (response->IsEOS()) {
        std::unique_lock lock(g_Mutex);
        g_CompleteNum++;
        std::cout << "ReqId:" << reqId << " Finished" << std::endl;
    }
}

void SendRequest(InferenceEngine &engine, uint64_t maxBatchSize)
{
    uint64_t processingNum = 0;
    engine.GetProcessingRequest(&processingNum);
    std::cout << "the processing request num is " << processingNum << " at first." << std::endl;

    uint64_t slotNum = 0;
    uint64_t remainBlocks = 0;
    uint64_t remainPrefillSlots = 0;
    uint64_t remainPrefillTokens = 0;
    while (!g_Manager.Empty()) {
        // 2. 获取可用的slot数目
        engine.GetRequestBlockQuotas(&remainBlocks, &remainPrefillSlots, &remainPrefillTokens);
        engine.GetProcessingRequest(&processingNum);
        slotNum = maxBatchSize - processingNum;
```



```
        if (remainBlocks > 0 && remainPrefillSlots > 0 && remainPrefillTokens > 0) {
            // 3. Set input
            std::vector<std::shared_ptr<Data>> data =
                g_Manager.GetInputDataByQuotas(remainBlocks, remainPrefillSlots, remainPrefillTokens,
slotNum);

            if (!data.empty()) {
                std::vector<std::shared_ptr<InferenceRequest>> requests = Data2Request(data);
                g_Statistics.requestNumber += requests.size(); // total num

                // 4. forward（异步）
                for (size_t i = 0; i < requests.size(); ++i) {
                    auto reqId = requests[i]->GetRequestId().StringValue();
                    {
                        std::unique_lock lock(g_MetricsMutex);
                        g_Metrics[reqId].startingTime = SC::now();
                        g_Metrics[reqId].tokensInput = data[i]->size;
                    }
                    engine.Forward(requests[i]);
                }
            }
        }
        std::this_thread::sleep_for(std::chrono::milliseconds(20L));
    }

    engine.GetProcessingRequest(&processingNum);
    std::cout << "the processing request num is " << processingNum << " when all requests dispatched." <<
std::endl;
}

void RunEngine(std::string dataset)
{
    TtimeT start;
    TtimeT end;

    g_Manager.SetInputData(dataset);

    // 初始化engine
    InferenceEngine engine;
    auto ret = engine.Init(ResponseCallback);
    if (!ret.IsOk()) {
        std::cout << "engine init failed:  " << ret.StatusMsg() << std::endl;
        return;
    }

    uint64_t maxBatchSize;
    ret = engine.GetMaxBatchSize(&maxBatchSize);
    if (!ret.IsOk()) {
        std::cout << "GetMaxBatchSize failed:  " << ret.StatusMsg() << std::endl;
        return;
    }

    start = SC::now();
    SendRequest(engine, maxBatchSize);
    while (g_CompleteNum < g_Statistics.requestNumber) {
        std::this_thread::sleep_for(std::chrono::milliseconds(10L));
    }
    end = SC::now();

    // 5. 统计打点信息
    g_Statistics.modelFullName = "";
    g_Statistics.tp = 8;
    g_Statistics.pp = 1;
    g_Statistics.latencyForAll = GetDuration(end, start);

    FormatMetrics(g_Metrics, g_Statistics);
    PrintStatistics(g_Statistics);

    if (g_RecordOutput) {
```



```
        std::map<std::string, std::vector<int64_t>> outputTokensId;
        for (auto &m : g_Metrics) {
            outputTokensId[m.first] = m.second.outputTokenIds;
        }
        WriteOutputIds(outputTokensId, "./token_output.csv");
    }

    // 6. 释放资源
    auto res = engine.Finalize();
    std::cout << "inferenceEngine finalize message is : " << res.StatusMsg() << std::endl;

    {
        std::unique_lock<std::mutex> lock(g_ExitMtx);
        g_ProcessExit = true;
    }
}
}

int main(int argc, char *argv[])
{
    // 数据集管理
    std::string dataset = argc > 1 ? argv[1] : "token_input_gsm.csv";
    g_RecordOutput = argc > 2 && std::stoi(argv[2]);

    std::thread businessThread(RunEngine, dataset);
    businessThread.detach();

    int status;
    while (true) {
        pid_t childPid = wait(&status);
        if (childPid == -1) {
            sleep(10u);
        } else {
            std::cout << "Child Process:" << childPid << " Exited" << std::endl;
        }

        std::unique_lock<std::mutex> lock(g_ExitMtx);
        if (g_ProcessExit) {
            break;
        }
    }

    return 0;
}
```

3.3 API 接口参考

3.3.1 Engine 提供的 C++接口

3.3.1.1 结构体和枚举说明

3.3.1.1.1 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;
```

```
    uint32_t topK;
    float topP;
    float typicalP;
    bool doSample;
    uint32_t seed;
    float repetitionPenalty;
    bool watermark;
    float frequencyPenalty;
    float presencePenalty;
};
```

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。	-
topK	uint32_t	控制模型生成过程中考虑的词汇范围，只从概率最高的k个候选词中选择。使用限制请参见**使用限制**。	-
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)
typicalP	float	每个单词被选择为下一个单词的概率大小。	(0, 1)
doSample	bool	是否做sampling。	-
seed	uint32_t	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。	-
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。	-
watermark	bool	水印。	-
frequencyPenalty	float	它影响模型如何根据文本中词汇（token）的现有频率惩罚新词汇（token）。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。	-
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。	-

使用样例

```
llm::engine::SamplingParams param;
param.temperature = 1.0;
```



```
param.topK = 0;
param.topP = 1.0;
```

返回值

无

3.3.1.1.2 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

使用样例

无

返回值

无

3.3.1.1.3 DataType 结构体

结构体功能

标识请求体对象序列号数据类型的枚举类。

结构体格式

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


结构体参数

参数	参数类型	说明	取值要求
UINT64	枚举值	请求体对象序列号数据类型为uint64。	0
STRING	枚举值	请求体对象序列号数据类型为string。	1

使用样例

无

返回值

无

3.3.1.1.4 LLMENGINE_memorytype_enum 枚举

内存类型枚举

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

3.3.1.1.5 LLM_ENGINE_DataType_enum 枚举

数据类型枚举

```
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;
```

3.3.1.1.6 LLM_ENGINE_parametertype_enum 枚举

参数类型枚举

```
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;
```

3.3.1.1.7 InferResponseEndFlagEnum 枚举

```
InferenceResponse GetFlags接口获取到的标志定义
typedef enum InferResponseEndFlagEnum {
    // 请求继续迭代执行
    INFER_RESPONSE_CONTINUE = 0,
    // 请求正常结束
    INFER_RESPONSE_EOS = 1,
    // 请求被主动CANCEL或STOP，用户不感知，丢弃响应
    INFER_RESPONSE_CANCEL = 2,
    // 请求执行中出错，响应输出为空，err_msg非空
    INFER_RESPONSE_EXEC_ERROR = 3,
    // 请求输入校验异常，响应输出为空，err_msg非空
    INFER_RESPONSE_ILLEGAL_INPUT = 4,
    // 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_SEQ_LEN = 5,
    // 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
    INFER_RESPONSE_REACH_MAX_OUTPUT_LEN = 6,
} InferResponseEndFlag;
```

3.3.1.1.8 Code 枚举

错误码类型枚举

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

3.3.1.1.9 Operation 枚举

控制操作类型枚举

控制操作类型枚举，目前只有STOP一种类。

```
struct Operation {
    STOP = 1,
};
```

3.3.1.2 类参考

3.3.1.2.1 InferenceEngine

类说明

InferenceEngine推理引擎功能实现类，提供服务初始化、模型加载、推理请求调度等功能入口。

InferenceEngine 接口

接口功能

默认构造函数。

接口格式

```
InferenceEngine();
```

接口参数

无

返回值

无

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);

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。

- SUCCESS：初始化成功。
- 否则初始化失败，并返回失败原因。

GetRequestBlockQuotas 接口

接口功能

获取remainBlocks、remainPrefillSlots和remainPrefill的值。

接口格式

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

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。

- SUCCESS：获取成功。
- 否则获取失败，并返回失败原因。

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。

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。

- SUCCESS：获取成功。
- 否则获取失败，并返回失败原因。

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。

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。

- SUCCESS：去初始化成功。

- 否则去初始化失败，并返回失败原因。

~InferenceEngine()

接口功能

默认析构函数

接口格式

```
~InferenceEngine();
```

接口参数

无

返回值

无

GetMaxBatchSize

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	batch size大小。	合法的数字。

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。

3.3.1.2.2 Input

Input(const std::string &name, const LLM_ENGINE_DataType datatype, const int64_t *shape,const uint64_t dim_count)

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, const LLM_ENGINE_DataType datatype, const int64_t *shape, const uint64_t dim_count)
```

接口参数

表 3-4 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，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3-5。
shape	必选	张量的维度。	合法的维度信息。
dim_count	必选	张量的dim。	合法的维度值。

表 3-5 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

LLM_ENGINE_DataType_enum类型	值	意义
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

```
std::shared_ptr<Input> input = std::make_shared<Input>(name, datatype, shape, dim_count);
```

返回值

返回推理请求输入张量对象。

Input() = delete

接口参数

无

使用样例

无

返回值

input对象。

~Input()

接口功能

析构函数。

接口格式

```
~Input();
```

接口参数

无

使用样例

无

返回值

无

Status Allocate(size_t size)

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。

const std::string &Name() const

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

无

返回值

string类型的输入张量名称。

LLM_ENGINE_DataType DType() const

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

无

返回值

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

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

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

无

返回值

输入张量的维度信息。

const uint64_t &DimCount() const

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

无

返回值

输入张量dim大小。

void *Buffer()

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。

uint64_t ByteSize() const

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

无

返回值

输入张量的输入数据总大小。

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

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

无

返回值

inputInner对象。

3.3.1.2.3 InferenceRequest

InferenceRequest(const RequestId &reqId)

接口功能

创建InferenceRequest对象。

接口格式

```
InferenceRequest(const RequestId &reqId);
```

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。

使用样例

无

返回值

InferenceRequest对象。

InferenceRequest() = delete

接口功能

InferenceRequest默认构造器。

接口格式

```
InferenceRequest() = delete;
```

接口参数

无

使用样例

无

返回值

InferenceRequest对象。

Status AddOriginalInput(const LLM_ENGINE_DataType datatype, const int64_t *shape, const uint64_t dim_count, Input **input)

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。

接口格式

```
Status AddOriginalInput(const LLM_ENGINE_DataType datatype, const int64_t *shape, const uint64_t dim_count, 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。
shape	必选	输入张量的维度。	合法的维度信息。
dim_count	必选	输入张量的dim。	合法的维度值dim。
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

无

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。

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

接口功能

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


接口参数

参数	是否必选	说明	取值要求
name	必选	输入张量的名称。	合法的张量名称。
datatype	必选	输入张量中保存的数据类型。	合法的数据类型。
shape	必选	输入张量的维度。	合法的维度信息。
dim_count	必选	输入张量的dim。	合法的维度dim值。
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

无

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。

Status RemoveOriginalInput(const std::string &name)

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。	合法的张量名称。

使用样例

无

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。

uint32_t MaxOutputLen() const

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

无

返回值

推理请求返回token的最大值，uint32_t类型。

bool SetMaxOutputLen(uint32_t max_output_len)

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 max_output_len);
```

接口参数

参数	是否必选	说明	取值要求
max_output_len	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

无

返回值

- true：设置成功。
- false：设置失败。

RequestId &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

无

返回值

返回推理请求体对象。

SamplingParams GetSamplingParams() const

接口功能

获取后处理参数。

接口格式

```
SamplingParams GetSamplingParams() const
```

接口参数

无

使用样例

无

返回值

后处理参数。

void SetSamplingParams(SamplingParams samplingParams)

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

无

返回值

无

void SetSendResponseCallback(const SendResponseCallback &callback)

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。	合法的回调函数。

使用样例

无

返回值

无

bool HasSampling()

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

无

返回值

- true：存在后处理参数。
- false：不存在后处理参数。

void SetInputText(std::string &text)

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
string	必选	推理文本。	合法字符串。

使用样例

无

返回值

无

std::string &GetInputText()

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

无

返回值

推理文本字符串。

bool IsInputText()

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

无

返回值

- true：属于文本推理。
- false：不属于文本推理。

std::shared_ptr<InferenceRequestInner> GetRequestInner() const

接口功能

获取RequestInner对象。

接口格式

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

接口参数

无

使用样例

无

返回值

RequestInner对象。

3.3.1.2.4 RequestId

类说明

推理请求对象，作为推理请求的唯一标识，在请求对象内部，唯一性标识序列id根据数据类型分为两种，一种是uint64，一种是string。推理请求对象可以被多次使用。

RequestId(const std::string &request_label)

接口功能

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

接口格式

```
RequestId(const std::string &request_label)
```

接口参数

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

使用样例

无

返回值

RequestId对象。

RequestId(uint64_t request_index)

接口功能

根据序列uint64类型request_index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

```
RequestId(uint64_t request_index);
```


接口参数

参数	是否必选	说明	取值要求
request_index	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

无

返回值

RequestId对象。

RequestId &operator = (const uint64_t rhs)

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。

接口格式

```
RequestId &operator = (const uint64_t 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。

使用样例

无

返回值

RequestId对象。

RequestId &operator = (const std::string &rhs)

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。

接口格式

```
RequestId &operator = (const std::string &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。	合法的字符串。

使用样例

无

返回值

返回RequestId对象。

RequestId &operator = (const RequestId &rhs)

接口功能

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 RequestId &rhs)
```

接口参数

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

无

返回值

返回RequestId对象。

RequestId(const llm::engine::RequestId &other)

接口功能

使用已有的requestId初始化RequestId对象。

接口格式

```
RequestId(const llm::engine::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

无

返回值

RequestId对象。

DataType Type() const

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

无

返回值

请求体对象唯一的序列号标识的数据类型。

const std::string &StringValue() const

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

无

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。

uint64_t UnsignedIntValue() const

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

无

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。

3.3.1.2.5 Output

~Output()

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

const std::string &Name() const

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

无

返回值

输出张量的名称。

LLM_ENGINE_DataType DType() const

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

无

返回值

输出张量的数据类型。

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

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

无

返回值

返回输出张量维度数组。

const uint64_t &DimCount() const

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

无

返回值

输出张量的dim值。

void *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

无

返回值

无

const void *Buffer() const noexcept = 0

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

无

返回值

无

uint64_t ByteSize() const

接口功能

获取输出张量对象数据总大小。

接口格式

```
uint64_t ByteSize() const
```

接口参数

无

使用样例

无

返回值

返回输出张量对象数据总大小。

3.3.1.2.6 InferenceResponse

~InferenceResponse() = default

接口功能

析构函数。

接口格式

```
~InferenceResponse()
```

接口参数

无

使用样例

无

返回值

无

bool IsEOS()

接口功能

判断推理使用的tokennizer是否需要使用eos。

接口格式

```
bool IsEOS()
```

接口参数

无

使用样例

无

返回值

tokennizer是否需要使用eos判断结果。

uint32_t GetFlags()

接口功能

获取推理请求当前状态标志。

接口格式

```
uint32_t GetFlags()
```


接口参数

无

使用样例

无

返回值

返回推理请求当前所处状态的标志。

Status ImmutableOutput(const std::string &name, Output **output)

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

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

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

无

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。

RequestId GetRequestId() const

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

```
RequestId GetRequestId() const
```

接口参数

无

使用样例

无

返回值

当前响应请求对象对应的推理请求体对象。

3.3.1.2.7 Error

Error(Code code = Code::OK)

接口功能

根据枚举Code创建返回信息对象。

接口格式

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

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

使用样例

```
Error(Error::Code::OK)
```

返回值

Error对象。

Error(Code code, std::string msg)

接口功能

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, 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。

使用样例

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

返回值

Error对象。

Code ErrorCode() const

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。

const std::string &Message() const

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。

bool IsOk() const

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。

std::string ToString() const

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

static const char *CodeToString(const Code code)

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:

3.3.1.2.8 Status

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

接口功能

通过错误码创建状态对象。

接口格式

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

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

使用样例

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

返回值

返回状态对象。

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

接口功能

通过错误码，错误信息创建状态对象。

接口格式

```
explicit Status(Error::Code code, const 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
msg	必选	错误信息.	合法字符串。

使用样例

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

返回值

返回状态对象。

explicit Status(const Error &error)

接口功能

通过错误信息返回对象创建状态对象。

接口格式

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

接口参数

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

返回值

返回状态对象。

explicit Status(Error *error)

接口功能

通过错误信息返回对象创建状态对象，同时创建完成后释放错误信息返回对象。

接口格式

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


接口参数

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

返回值

返回状态对象。

bool IsOk() const

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。

Error::Code StatusCode() const

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。

std::string StatusMsg() const

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。

bool operator==(const Status& other) const

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

无

3.3.2 EndPoint 提供的 RESTful 接口

3.3.2.1 兼容 TGI 0.9.4 版本接口

3.3.2.1.1 健康检查

接口功能

检查服务状态是否正常。

接口格式

操作类型：**GET**

URL：https://*{ip}*:*{port}*/health

请求参数

无

使用样例

请求样例：

```
GET https://<ip>:<port>/health
```

响应状态码：200

输出说明

- 状态码200，服务状态正常，消息体没有内容。
- 其他状态码，服务状态异常。

3.3.2.1.2 查询 TGI EndPoint 信息

接口功能

查询TGI EndPoint信息。

说明

最大程度兼容TGI接口返回格式，对于MindIE Server不支持的返回字段，返回null。

接口格式

操作类型：**GET**

URL：https://*{ip}*:*{port}*/info

请求参数

无

使用样例

请求样例：

```
GET https://<ip>:<port>/info
```

响应样例：

```
{
 "docker_label": null,
 "max_batch_total_tokens": 32000,
 "max_best_of": 1,
 "max_concurrent_requests": 300,
 "max_input_length": 1024, //maxSeqLen - maxIterTimes
 "max_stop_sequences": null,
 "max_waiting_tokens": null,
 "models": [{
   "model_device_type": "npu",
   "model_dtype": "torch.float16",
   "model_id": "bigscience/blomm-560m",//模型名称
   "model_pipeline_tag": "text-generation",
    "max_total_tokens": 2048,//取maxSeqLen的值
   "model_sha": null
  },
  {
   "model_device_type": "npu",
   "model_dtype": "torch.float16",
   "model_id": "bigscience/blomm-560m",
   "model_pipeline_tag": "text-generation",
    "max_total_tokens": 2048,//取maxSeqLen的值
   "model_sha": null
  }
 ],
 "sha": null,
 "validation_workers": null,
 "version": "*{version}*",
 "waiting_served_ratio": null
}
```

响应状态码：200

输出说明

参数	类型	说明
docker_label	string	暂不支持，默认返回null。
max_batch_total_tokens	int	建议取maxPrefillTokens。
max_best_of	int	暂不支持best_of参数，默认返回1，即每次只返回1个推理结果。
max_concurrent_requests	int	最大并发请求数，取maxBatchSize。
max_input_length	int	最大输入长度，取值maxSeqLen-maxIterTimes。
max_stop_sequences	int	暂不支持，默认返回null。
max_waiting_tokens	int	暂不支持，默认返回null。
models	list	模型配置。
model_device_type	string	模型运行设备类型，默认返回"npu"。
model_dtype	string	模型数据类型，读取权重配置文件目录config.json文件中的torch_dtype字段。
model_id	string	模型名称。
model_pipeline_tag	string	模型任务类型，默认返回"text-generation"。
max_total_token	string	最大推理token总数，读取maxSeqLen的值。
model_sha	string	暂不支持，默认返回null。
sha	string	暂不支持，默认返回null。
validation_workers	int	暂不支持，默认返回null。
version	string	"*{version}*"，版本号。
waiting_served_ratio	float	暂不支持，默认返回null。

3.3.2.1.3 文本/流式推理接口

接口功能

提供文本/流式推理处理功能。

接口格式

操作类型：**POST**

URL：https://*{ip}:{port}*/

请求参数

参数	是否必选	说明	取值要求
inputs	必选	推理请求文本。	非空，0<字符数<=16000，支持中英文。tokenizer之后的token数量<=maxSeqLen-maxIterTimes（配置文件读取）。
decoder_input_details	可选	是否返回推理请求文本的token id。如果“stream”=“true”，该参数不能为“true”。	bool类型，默认值false。
details	可选	是否返回推理详细输出结果。根据TGI 0.9.4接口行为，“decoder_input_details”和“details”字段任意一个为“true”，即返回所有的details信息。	bool类型，默认值false。
do_sample	可选	是否做sampling。	bool类型，默认值false。
max_new_tokens	可选	允许的最大新标记数目。控制从模型生成的文本中添加到最终输出中的最大词汇数量。该字段受到GIMIS配置文件maxIterTimes参数影响，推理token输出长度<=maxIterTimes。	int类型，取值范围(0, maxIterTimes]。默认值20。
repetition_penalty	可选	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，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。
return_full_text	可选	是否将推理请求文本（inputs参数）添加到推理结果前面。	bool类型，默认值false。 ● true：添加。 ● false：不添加。

参数	是否必选	说明	取值要求
seed	可选	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。	uint_64，取值范围(0, 18446744073709551615]，不传递该参数，系统会产生一个随机seed值。
temperature	可选	控制生成的随机性，较高的值会产生更多样化的输出。	float类型，大于0，默认值1.0。 • 1.0表示不进行计算。 • 大于1.0表示输出随机性提高。
top_k	可选	控制模型生成过程中考虑的词汇范围，只从概率最高的k个候选词中选择。使用限制请参见**使用限制**。	int类型，取值范围(0, vocabSize)&&(0, 2147483647]，默认值0。 vocabSize是从modelWeightPath路径下的config.json文件中读取的vocab_size值，若不存在则vocabSize取默认值0。
top_p	可选	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。	float类型，取值范围(0, 1)，默认值1.0。
truncate	可选	输入文本做tokenizer之后，将token数量截断到该长度。读取截断后的n个token。	int类型，取值范围(0, maxSeqLen-maxIterTimes]，默认值0（表示不做truncate）。
stream	可选	指定返回结果是文本推理还是流式推理。	bool类型，默认值false。

使用样例

请求样例：

```
POST https://<ip>:<port>/
```

请求消息体：

```
{
  "inputs": "My name is Olivier and I",
```



```
 "parameters": {
   "decoder_input_details": true,
   "details": true,
   "do_sample": true,
   "max_new_tokens": 20,
   "repetition_penalty": 1.03,
   "return_full_text": false,
   "seed": null,
   "temperature": 0.5,
   "top_k": 10,
   "top_p": 0.95,
   "truncate": null
  },
 "stream": false
}
```

文本推理（“stream”=“false”）响应样例：

```
{
 "details": {
  "finish_reason": "length",
  "generated_tokens": 1,
  "prefill": [{
   "id": 0,
   "logprob":null,
   "text": "test"
  }],
  "seed": 42,
  "tokens": [{
   "id": 0,
   "logprob": null,
   "special": null,
   "text": "test"
  }]
 },
 "generated_text": "am a Frenchman living in the UK. I have been working as an IT consultant for "
}
```

流式推理（“stream”=“true”）响应样例（使用sse格式返回）：

```
data: {"token":{"id":626,"text":"am","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":263,"text":" a","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":5176,"text":" French","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":1171,"text":"man","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":8471,"text":" living","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":297,"text":" in","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":278,"text":" the","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":10261,"text":" UK","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":29889,"text":".","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":306,"text":" I","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":505,"text":" have","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":1063,"text":" been","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":1985,"text":" working","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":408,"text":" as","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":385,"text":" an","logprob":null,"special":null},"generated_text":null,"details":null}
```



```
data: {"token":{"id":13315,"text":" IT","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":8799,"text":" consult","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":424,"text":"ant","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":363,"text":" for","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":29871,"text":" ","logprob":null,"special":null},"generated_text":"am a Frenchman living in the UK. I have been working as an IT consultant for ","details":null}
```

输出说明

表 3-6 文本推理结果说明

返回值	类型	说明
generated_text	string	推理返回结果。
details	object	推理details结果，请求参数“decoder_input_details”和“details”任意一个字段为“True”，即返回details结果。
finish_reason	string	结束原因。
generated_tokens	int	推理结果token数量。
seed	int	返回推理请求的seed值，如果请求参数没有指定seed参数，则返回系统随机生成的seed值。
prefill	List[token]	请求参数“decoder_input_details”=“True”，返回推理请求文本detokenizer之后的token，默认为空列表。
id	int	token id。
text	string	token对应文本。
logprob	float	概率对数，可以为空（第一个token概率值不能被计算获得）。当前不支持，默认返回null。
tokens	List[token]	返回推理结果的所有tokens。
id	int	token id。
text	string	token对应文本。
logprob	概率对数	概率对数。当前不支持，默认返回null。
special	bool	表明该token是否是special，如果是“special”=“True”，该token在做连接的时候可以被忽略。当前不支持，默认返回null。

表 3-7 流式推理结果说明

返回值	类型	说明
generated_text	string	推理文本返回结果，只在最后一次推理结果才返回。
details	object	推理details结果，只在最后一次推理结果返回，并且请求参数“details”=“True”才返回details结果。
finish_reason	string	结束原因。
generated_tokens	int	推理结果token数量。
seed	int	返回推理请求的seed值，如果请求参数没有指定seed参数，则返回系统随机生成的seed值。
token	object	每一次推理的token。
id	int	token id。
text	string	token对应文本。
logprob	概率对数	当前不支持，默认返回null。
special	bool	表明该token是否是special，如果是“special”=“True”，该token在做连接的时候可以被忽略。当前不支持，默认返回null。

3.3.2.1.4 文本推理接口

提供文本推理处理功能。

接口格式

操作类型：**POST**

URL：https://*{ip}*:*{port}*/generate

请求参数

参数	是否必选	说明	取值要求
inputs	必选	推理请求文本。	非空，0<字符数<=16000，支持中英文。tokenizer之后的token数量<=maxSeqLen-maxIterTimes（配置文件读取）。

参数	是否必选	说明	取值要求
decoder_input_details	可选	是否返回推理请求文本的token id。如果stream=true，该参数不能为true。	bool类型，默认值false。
details	可选	是否返回推理详细输出结果。根据TGI 0.9.4接口行为，decoder_input_details和details字段任意一个为true，即返回所有的details信息。	bool类型，默认值false。
do_sample	可选	是否做sampling。	bool类型，默认值false。
max_new_tokens	可选	允许的最大新标记数目。控制从模型生成的文本中添加到最终输出中的最大词汇数量。该字段受到GIMIS配置文件maxIterTimes参数影响，推理token输出长度<=maxIterTimes。	int类型，取值范围(0, maxIterTimes]。默认值20。
repetition_penalty	可选	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，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。
return_full_text	可选	是否将推理请求文本（inputs参数）添加到推理结果前面。	bool类型，默认值false。 ● true表示添加。 ● false表示不添加。
seed	可选	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。	uint_64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。
temperature	可选	控制生成的随机性，较高的值会产生更多样化的输出。	float类型，取值大于0，默认值1.0。 ● 1.0表示不进行计算。 ● 大于1.0表示输出随机性提高。

参数	是否必选	说明	取值要求
top_k	可选	控制模型生成过程中考虑的词汇范围，只从概率最高的k个候选词中选择。使用限制请参见**使用限制**。	int类型，取值范围(0, vocabSize)&&(0, 2147483647]，默认值0。 vocabSize是从modelWeightPath路径下的config.json文件中读取的vocab_size值，若不存在则vocabSize取默认值0。
top_p	可选	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。	float类型，取值范围(0, 1)，默认值1。
truncate	可选	输入文本做tokenizer之后，将token数量截断到该长度。读取截断后的n个token。	int类型，取值范围(0, maxSeqLen-maxIterTimes]，默认值0（表示不做truncate）。

使用样例

请求样例：

```
POST https://<ip>:<port>/generate
```

请求消息体：

```
{
  "inputs": "My name is Olivier and I",
  "parameters": {
    "decoder_input_details": true,
    "details": true,
    "do_sample": true,
    "max_new_tokens": 20,
    "repetition_penalty": 1.03,
    "return_full_text": false,
    "seed": null,
    "temperature": 0.5,
    "top_k": 10,
    "top_p": 0.95,
    "truncate": null
   }
}
```

响应样例：

```
{
 "details": {
  "finish_reason": "length",
```



```
 "generated_tokens": 1,
 "prefill": [{
  "id": 0,
  "logprob":null,
  "text": "test"
 }],
 "seed": 42,
 "tokens": [{
  "id": 0,
  "logprob": null,
  "special": null,
  "text": "test"
 }]
},
"generated_text": "am a Frenchman living in the UK. I have been working as an IT consultant for "
}
```

输出说明

返回值	类型	说明
generated_text	string	推理返回结果。
details	object	推理details结果，请求参数“decoder_input_details”和“details”任意一个字段为“True”，即返回details结果。
finish_reason	string	结束原因。
generated_tokens	int	推理结果token数量。
seed	int	返回推理请求的seed值，如果请求参数没有指定seed参数，则返回系统随机生成的seed值。
prefill	List[token]	请求参数“decoder_input_details”=“True”，返回推理请求文本detokenizer之后的token，默认为空列表。
id	int	token id。
text	string	token对应文本。
logprob	float	概率对数，可以为空（第一个token概率值不能被计算获得）。当前不支持，默认返回null。
tokens	List[token]	返回推理结果的所有tokens。
id	int	token id。
text	string	token对应文本。
logprob	概率对数	概率对数。当前不支持，默认返回null。

返回值	类型	说明
special	bool	表明该token是否是special，如果是“special”=“True”，该token在做连接的时候可以被忽略。当前不支持，默认返回null。

3.3.2.1.5 流式推理接口

提供流式推理处理功能。

接口格式

操作类型：**POST**

URL：https://*{ip}*:*{port}*/generate_stream

请求参数

参数	是否必选	说明	取值要求
inputs	必选	推理请求文本。	非空，0<字符数<=16000，支持中英文。tokenizer之后的token数量<=maxSeqLen-maxIterTimes（配置文件读取）。
details	可选	是否返回推理详细输出结果。根据TGI 0.9.4接口行为，“details”=“true”，即返回所有的details信息。	bool类型参数，默认值false。
do_sample	可选	是否做sampling。	bool类型，默认值false。
max_new_tokens	可选	允许的最大新标记数目。控制从模型生成的文本中添加到最终输出中的最大词汇数量。该字段受到GIMIS配置文件maxIterTimes参数影响，推理token输出长度<=maxIterTimes。	int类型，取值范围(0, maxIterTimes]。默认值20。

参数	是否必选	说明	取值要求
repetition_penalty	可选	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，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。
return_full_text	可选	是否将推理请求文本（inputs参数）添加到推理结果前面。	bool类型，默认值false。 ● true表示添加。 ● false表示不添加。
seed	可选	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。	uint_64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。
temperature	可选	控制生成的随机性，较高的值会产生更多样化的输出。	float类型，大于0，默认值1.0。 ● 1.0表示不进行计算。 ● 大于1.0表示输出随机性提高。
top_k	可选	控制模型生成过程中考虑的词汇范围，只从概率最高的k个候选词中选择。使用限制请参见**使用限制**。	int类型，取值范围(0, vocabSize)&&(0, 2147483647]，默认值0。 vocabSize是从modelWeightPath路径下的config.json文件中读取的vocab_size值，若不存在则vocabSize取默认值0。
top_p	可选	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。	float类型，取值范围(0, 1)，默认值1.0。

参数	是否必选	说明	取值要求
truncate	可选	输入文本做tokenizer之后，将token数量截断到该长度。读取截断后的n个token。	int类型，取值范围(0, maxSeqLen-maxIterTimes]，默认值0（表示不做truncate）。

使用样例

请求样例：

```
POST https://<ip>:<port>/generate_stream
```

请求消息体：

```
{
  "inputs": "My name is Olivier and I",
  "parameters": {
    "details": true,
    "do_sample": true,
    "max_new_tokens": 20,
    "repetition_penalty": 1.03,
    "return_full_text": false,
    "seed": null,
    "temperature": 0.5,
    "top_k": 10,
    "top_p": 0.95,
    "truncate": null
   }
}
```

响应样例（使用sse格式返回）：

```
data: {"token":{"id":626,"text":"am","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":263,"text":" a","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":5176,"text":" French","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":1171,"text":"man","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":8471,"text":" living","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":297,"text":" in","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":278,"text":" the","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":10261,"text":" UK","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":29889,"text":".","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":306,"text":" I","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":505,"text":" have","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":1063,"text":" been","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":1985,"text":" working","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":408,"text":" as","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":385,"text":" an","logprob":null,"special":null},"generated_text":null,"details":null}
```


data: {"token":{"id":13315,"text":" IT","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":8799,"text":" consult","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":424,"text":"ant","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":363,"text":" for","logprob":null,"special":null},"generated_text":null,"details":null}

data: {"token":{"id":29871,"text":" ","logprob":null,"special":null},"generated_text":"am a Frenchman living in the UK. I have been working as an IT consultant for ","details":null}

data: {"token":{"id":2,"text":"","logprob":null,"special":null},"generated_text":"\nJanet makes $104 a day at the farmers' market.\nThe number of eggs that Janet sells at the farmers' market each day is 16 - 3 - 4 = 7.\nShe makes $2 for each egg that she sells.\nThe total amount that she makes is $2 * 7 = $14.\nThe total amount that she makes is $14 + $104 = $118.\nThe total amount that she makes is $118.","details":{"finish_reason":"eos_token","generated_tokens":116,"seed":8756727004129248931}}

输出说明

返回值	类型	说明
generated_text	string	推理文本返回结果，只在最后一次推理结果才返回。
details	object	推理details结果，只在最后一次推理结果返回，并且请求参数“details”=“True”才返回details结果。
finish_reason	string	结束原因。
generated_tokens	int	推理结果token数量。
seed	int	返回推理请求的seed值，如果请求参数没有指定seed参数，则返回系统随机生成的seed值。
token	object	每一次推理的token。
id	int	token id。
text	string	token对应文本。
logprob	概率对数	当前不支持，默认返回null。
special	bool	表明该token是否是special，如果是“special”=True，该token在做连接的时候可以被忽略。当前不支持，默认返回null。

3.3.2.2 兼容 vLLM 0.2.6 版本接口

3.3.2.2.1 健康检查

请参见**3.3.2.1.1 健康检查**。

3.3.2.2.2 文本/流式推理接口

接口功能

提供文本/流式推理处理功能。

接口格式

操作类型：**POST**

URL：https://*{ip}*:*{port}*/generate

请求参数

参数	是否必选	说明	取值要求
prompt	必选	推理请求文本。	非空，0<字符数<=16000，支持中英文。tokenizer之后的token数量<=maxSeqLen-maxIterTimes（配置文件读取）。
presence_penalty	可选	存在惩罚介于-2.0和2.0之间，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。
frequency_penalty	可选	频率惩罚介于-2.0和2.0之间，它影响模型如何根据文本中词汇（token）的现有频率惩罚新词汇（token）。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。
repetition_penalty	可选	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，取值大于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。

参数	是否必选	说明	取值要求
max_tokens	可选	允许的最大新标记数目。控制从模型生成的文本中添加到最终输出中的最大词汇数量。该字段受到GIMIS配置文件maxIterTimes参数影响，推理token输出长度<=maxIterTimes。	int类型，取值范围(0, maxIterTimes]。默认值16。
temperature	可选	控制生成的随机性，较高的值会产生更多样化的输出。1.0表示不进行计算，大于1.0表示输出随机性提高。temperature=0，即采用greedy sampling，该参数映射到后处理的1.0。	float类型，取值>=0，默认值1.0。
top_k	可选	控制模型生成过程中考虑的词汇范围，只从概率最高的k个候选词中选择。-1表示不进行top k计算。使用限制请参见**使用限制**。	int类型，取值范围-1或者(0, vocabSize)&&(0, 2147483647]，默认值-1。 vocabSize是从modelWeightPath路径下的config.json文件中读取的vocab_size值，若不存在则vocabSize取默认值0。
top_p	可选	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。	float类型，取值范围(0, 1]，默认值1.0。
stream	可选	指定返回结果是文本推理还是流式推理。	bool类型，默认值false。

使用样例

请求样例：

```
POST https://<ip>:<port>/generate
```

请求消息体：

```
{
    "prompt": "My name is Olivier and I",
    "max_tokens": 20,
    "repetition_penalty": 1.03,
    "presence_penalty": 1.2,
    "frequency_penalty": 1.2,
```



```
    "temperature": 0.5,
    "top_k": 10,
    "top_p": 0.95,
    "stream": false
}
```

文本推理（stream=false）响应样例：

```
{"text":["My name is Olivier and I am a Frenchman living in the UK. I am a keen photographer and"]}
```

流式推理（stream=true）响应样例（使用sse格式返回）：

```
{"text":["am"]}{"text":[" a"]}{"text":[" French"]}{"text":["man"]}{"text":[" living"]}{"text":[" in"]}"text":["
the"]}{"text":[" UK"]}{"text":["."]}{"text":[" I"]} {"text":[" am"]}{"text":[" a"]}"text":[" keen"]}{"text":["
photograph"]}{"text":["er"]}{"text":[" and"]}
```

输出说明

返回值	类型	说明
text	string	推理返回结果。

须知

使用**curl**命令发送vLLM流式推理请求的样例如下：

```
curl -H "Accept: application/json" -H "Content-type: application/json" --cacert /home/runs/static_conf/ca/ca.pem  --cert /home/runs/static_conf/cert/client.pem  --key /home/runs/static_conf/cert/client.key.pem -X POST -d '{
"prompt": "My name is Olivier and I",
"stream": true,
"repetition_penalty": 1.0,
"top_p": 1.0,
"top_k": 10,
"max_tokens": 16,
"temperature": 1.0
}' https://{ip}:{port}/generate
```

3.3.2.3 兼容 OpenAI 接口

3.3.2.3.1 列举当前模型列表

接口功能

列举当前模型列表信息。

接口格式

操作类型：**GET**

URL：https://*{ip}*:*{port}*/v1/models

请求参数

无

使用样例

请求样例：

```
GET https://<ip>:<port>/v1/models
```

响应样例：

```
{
  "object": "list",
  "data": [
    {
      "id": "model-id-0",
      "object": "model",
      "created": 1686935002,
      "owned_by": "organization-owner"
    }
  ],
  "object": "list"
}
```

响应状态码：200

输出说明

参数	类型	说明
object	string	返回结果类型，默认"list"。
data	list	模型列表。
id	string	模型名称。
object	string	数据结果类型，默认"model"。
created	int	模型加载时间戳，即服务启动时间戳，单位：秒。
owned_by	string	模型的拥有者，默认返回"MindIE Server"。

3.3.2.3.2 查询单个模型信息

接口功能

查询指定模型信息。

接口格式

操作类型：**GET**

URL：https://*{ip}*:*{port}*/v1/models/*{model}*

请求参数

URL链接中model字段为模型id。

使用样例

请求样例：

```
GET https://<ip>:<port>/v1/models/model-id-0
```

响应样例：

```
{
    "id": "model-id-0",
    "object": "model",
    "created": 1686935002,
    "owned_by": "organization-owner"
}
```

响应状态码：200

输出说明

参数	类型	说明
id	string	模型名称。
object	string	数据结果类型，默认"model"。
created	int	模型加载时间戳，即服务启动时间戳，单位：秒。
owned_by	string	模型的拥有者，默认返回"MindIE Server"。

3.3.2.3.3 推理接口

接口功能

提供文本/流式推理处理功能。

接口格式

操作类型：**POST**

URL：https://*{ip}*:*{port}*/v1/chat/completions

请求参数

参数	是否必选	说明	取值要求
model	必选	模型名，当前不校验该字段。	-
messages	必选	推理请求消息结构。	List类型。
role	可选	推理请求消息角色，当前不处理该字段。	-

参数	是否必选	说明	取值要求
content	必选	推理请求文本。	非空，0<字符数<=16000，支持中英文。tokenizer之后的token数量<=maxSeqLen-maxIterTimes（配置文件读取）。
max_tokens	可选	允许的最大新标记数目。控制从模型生成的文本中添加到最终输出中的最大词汇数量。该字段受到GIMIS配置文件maxIterTimes参数影响，推理token输出长度<=maxIterTimes。	int类型，取值范围(0, maxIterTimes]。默认值16。
presence_penalty	可选	存在惩罚介于-2.0和2.0之间，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。
frequency_penalty	可选	频率惩罚介于-2.0和2.0之间，它影响模型如何根据文本中词汇（token）的现有频率惩罚新词汇（token）。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。
seed	可选	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。	int_64类型，取值范围(0, 9223372036854775807]，不传递该参数，系统会产生一个随机seed值。
temperature	可选	控制生成的随机性，较高的值会产生更多样化的输出。	float类型，大于0，默认值1.0。 ● 1.0表示不进行计算。 ● 大于1.0表示输出随机性提高。

参数	是否必选	说明	取值要求
top_p	可选	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。	float类型，取值范围(0,1]，默认值1.0。
stream	可选	指定返回结果是文本推理还是流式推理。	bool类型参数，默认值false。

使用样例

请求样例：

```
POST https://<ip>:<port>/v1/chat/completions
```

请求消息体：

```
{
 "model": "gpt-3.5-turbo",
 "messages": [{
     "role": "system",
     "content": "You are a student who is good at math."
    },
    {
     "role": "user",
     "content": "what is your hobby?"
    }
],
 "max_tokens": 20,
 "presence_penalty": 1.03,
 "frequency_penalty": 1.0,
 "seed": null,
 "temperature": 0.5,
 "top_p": 0.95,
 "stream": false
}
```

文本推理（“stream”=“false”）响应样例：

```
{
  "id": "chatcmpl-123",
  "object": "chat.completion",
  "created": 1677652288,
  "model": "gpt-3.5-turbo-0613",
    "choices": [{
    "index": 0,
    "message": {
      "role": "assistant",
      "content": "\n\nHello there, how may I assist you today?"
    },
    "finish_reason": "stop"
  }],
  "usage": {
    "prompt_tokens": 9,
    "completion_tokens": 12,
    "total_tokens": 21
  }
}
```


流式推理（“stream”=“true”）响应样例（使用sse格式返回）：

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":"am"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":" a"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":" French"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":"man"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":" living"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":" in"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":" the"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":" UK"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":"."},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":" I"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":" am"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":" a"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":" keen"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":" photograph"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":"er"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":" and"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":" I"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":" have"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{"role":"assistant","content":" been"},"finish_reason":null}]}

data: {"id":"1","object":"chat.completion.chunk","created":1707275960,"model":"baichuan","choices":[{"index":0,"delta":{},"finish_reason":"length"}]}

输出说明

表 3-8 文本推理结果说明

参数名	类型	说明
id	string	请求id。
created	integer	推理请求时间戳，精确到秒。
model	string	使用的推理模型。
object	string	返回结果类型目前都返回"chat.completion"。
usage	object	推理结果统计数据。
completion_tokens	int	推理token数量。
prompt_tokens	int	请求文本tokonizer之后的数量。
total_tokens	int	请求+推理总token数。
choices	list	推理结果列表。
finish_reason	string	● stop：遇到stop条件自然停止。 ● length：达到max_tokens。
index	integer	choices消息index，从0开始。
message	object	推理消息。
role	string	角色，目前都返回"assistant"。
content	string	推理文本结果。

表 3-9 流式推理结果说明

参数名	类型	说明
id	string	请求id。
created	integer	推理请求时间戳，精确到秒。
model	string	使用的推理模型。
object	string	目前都返回"chat.completion.chunk"。
choices	list	流式推理结果。
finish_reason	string	● stop：遇到stop条件自然停止。 ● length：达到max_tokens。 最后一个消息结果才返回该值。
index	integer	choices消息index，从0开始。

参数名	类型	说明
delta	object	推理返回结果，最后一个响应为空。
role	string	角色，目前都返回"assistant"。
content	string	推理文本结果。

3.3.2.4 兼容 Triton 接口

3.3.2.4.1 健康检查

接口功能

检查服务状态是否正常。服务正常时，返回状态码200，消息体没有内容。

接口格式

操作类型：**GET**

URL：https://*{ip}:{port}*/v2/health/live

URL：https://*{ip}:{port}*/v2/health/ready

URL：https://*{ip}:{port}*/v2/models/*${MODEL_NAME}[/versions/${MODEL_VERSION}]*/ready

请求参数

无

说明

[/versions/${MODEL_VERSION}]字段暂不支持，不传递。

使用样例

请求样例：

```
GET https://<ip>:<port>/v2/health/live
GET https://<ip>:<port>/v2/health/ready
GET https://<ip>:<port>/v2/models/llama_65b/ready
```

响应状态码：200

输出说明

- 状态码200，服务状态正常。
- 其他状态码，服务状态异常。

3.3.2.4.2 查询服务元数据

接口功能

查询服务元数据信息。

接口格式

操作类型：**GET**

URL：https://*{ip}*:*{port}*/v2

请求参数

无

使用样例

请求样例：

```
GET https://<ip>:<port>/v2
```

响应样例：

```
{
 "name": "MindIE Server",
 "version": "{version}",
 "extensions": {
  "max_iter_times": 512,
  "prefill_policy_type": 0,
  "decode_policy_type": 0,
  "max_prefill_batch_size": 50,
  "max_prefill_tokens": 8192
 }
}
```

响应状态码：200

输出说明

参数	类型	说明
name	string	服务名称，暂定"MindIE Server"。
version	string	服务版本。
extensions	object	扩展字段。
max_iter_times	int	最大可进行的decode次数。
prefill_policy_type	int	prefill阶段的调度策略。
decode_policy_type	int	decode阶段的调度策略。
max_prefill_batch_size	int	最大prefill batch size。

参数	类型	说明
max_prefill_tokens	int	最大prefill token数量。

3.3.2.4.3 查询模型元数据

接口功能

查询模型元数据信息。

接口格式

操作类型：**GET**

URL：https://*{ip}*:*{port}*/v2/models/*${MODEL_NAME}[/versions/${MODEL_VERSION}]*

请求参数

URL中${MODEL_NAME}字段指定需要查询的模型名称。

说明

[/versions/${MODEL_VERSION}]字段暂不支持，不传递。

使用样例

请求样例：

```
GET https://<ip>:<port>/v2/models/llama_65b
```

响应样例：

```
{
 "name": "llama_65b",
 "platform": "MindIE Server",
 "inputs": [{
  "name": "input0",
  "shape": [-1],
  "datatype": "UINT32"
 }],
 "outputs": [{
  "name": "output0",
  "shape": [-1],
  "datatype": "UINT32"
 }]
}
```

响应状态码：200

输出说明

参数	类型	说明
name	string	模型名称。

参数	类型	说明
platform	string	固定返回“MindIE Server”。

inputs	json object	表示该模型infer接口接受的inputs参数格式，目前固定返回以下内容。
		"inputs" : [
		{
		"name" : "input0",
		"shape" : [-1],
		"datatype" : "UINT32",
		}
		]

outputs	json object	表示该模型infer接口接受的outputs参数格式，目前固定返回以下内容。
		"outputs" : [
		{
		"name" : "output0",
		"shape" : [-1],
		"datatype" : "UINT32"
		}
		]

3.3.2.4.4 查询模型配置数据

接口功能

查询模型配置数据信息。

接口格式

操作类型：**GET**

URL：https://*{ip}:{port}***/v2/models/***${MODEL_NAME}[/versions/${MODEL_VERSION}]***/config**

请求参数

URL中${MODEL_NAME}字段指定需要查询的模型名称。

说明

[/versions/${MODEL_VERSION}]字段暂不支持，不传递。

使用样例

请求样例：

```
GET https://<ip>:<port>/v2/models/llama_65b/config
```

响应样例：

```
{
 "model_name": "llama_65b",
 "input_datatype": "INT64",
 "output_datatype": "INT64",
 "max_seq_len": 2560,
```



```
 "eos_token_id": "2",
 "npu_mem_size": 8,
 "cpu_mem_size": 5,
 "world_size": 8,
 "model_weight_path": "/home/data/llama1-65b-safetensors",
 "model_instance_type": "Target_model"
}
```

响应状态码：200

输出说明

参数	类型	说明
model_name	string	推理选取的模型名字。
input_datatype	string	输入的数据类型。
output_datatype	string	输出的数据类型。
max_seq_len	int	最大序列长度。
eos_token_id	string	序列结束token。
npu_mem_size	int	NPU中可以用来申请kv cache的size上限。
cpu_mem_size	int	CPU中可以用来申请kv cache的size上限。
world_size	int	使用几张卡进行推理。
model_weight_path	string	模型权重路径。
model_instance_type	string	模型类型。

3.3.2.4.5 Token 推理接口

接口功能

提供Token推理处理功能。

接口格式

操作类型：**POST**

URL：https://*{ip}:{port}***/v2/models/***${MODEL_NAME}[/versions/${MODEL_VERSION}]***/infer**

请求参数

表 3-10 请求参数

参数	是否必选	说明	取值范围
id	可选	推理请求id。	字符串。
inputs	必选	只有一个元素的数组。	-
outputs	必选	推理结果输出结构。	object。

表 3-11 inputs 参数

参数	是否必选	说明	取值范围
name	必选	输入名称，固定"input0"。	"input0"。
shape	必选	参数维度，1行，n列，和data长度相关。	(0, len(data)]。
dataType	必选	data数据类型，目前场景仅支持UINT32，传递tokenid。	“UINT32”。
seed	可选	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。	uint_64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。
temperature	可选	控制生成的随机性，较高的值会产生更多样化的输出。	float类型，大于0，默认值1.0。 • 1.0表示不进行计算。 • 大于1.0表示输出随机性提高。
top_k	可选	控制模型生成过程中考虑的词汇范围，只从概率最高的k个候选词中选择。使用限制请参见**使用限制**。	int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值，若不存在则vocabSize取默认值0。

参数	是否必选	说明	取值范围
top_p	可选	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。	float类型，取值范围(0, 1]，默认值1.0。
do_sample	可选	是否做sampling。	bool类型，默认值false。
repetition_penalty	可选	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，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。
max_new_tokens	可选	允许的最大新标记数目。控制从模型生成的文本中添加到最终输出中的最大词汇数量。该字段受到GIMIS配置文件maxIterTimes参数影响，推理token输出长度<=maxIterTimes。	int类型，取值范围(0, maxIterTimes]。默认值20。

表 3-12 outputs 参数

参数	是否必选	说明	取值范围
name	必选	推理结果输出名。	只支持"output0"。

说明

[/versions/${MODEL_VERSION}]字段暂不支持，不传递。

使用样例

请求样例：

```
POST https://<ip>:<port>/v2/models/llama_65b/infer
```

请求消息体：

```
{
 "id": "42",
 "inputs": [{
  "name": "input0",
```



```
 "shape": [
  1,
  10
 ],
 "datatype": "UINT32",
 "data": [
  396, 319, 13996, 29877, 29901, 29907, 3333, 20718, 316, 23924
 ],
 "parameters": {
  "temperature": 0.5,
  "top_k": 10,
  "top_p": 0.95,
  "do_sample": true,
  "seed": null,
  "repetition_penalty": 1.03,
  "max_new_tokens": 512
 }
}],
"outputs": [{
 "name": "output0"
}]
}
```

响应样例：

```
{
 "id": "42",
 "outputs": [{
  "name": "output0",
  "shape": [1, 512],
  "datatype": "UINT32",
  "data": [1, 396, 319, 13996, 29877, 29901, 29907, 3333, 20718, 316, 23924, 562, 2142, 1702, 425, 14015,
16060, 316, 383, 19498, 316, 29871, 29896, 29929, 29929, 29900, 13, 13, 5661, 22215, 20718, 316, 23924,
562, 2142, 1702, 425, 14015, 16060, 316, 383, 19498, 316, 29871, 29896, 29929, 29929, 29900, 3576, 560,
23300, 712, 11806, 29980, 263, 1232, 22215, 928, 2255, 1277, 3810, 316, 425, 10811, 287, 1572, 1290, 316,
383, 19498, 316, 21035, 29892, 18726, 314, 10820, 343, 560, 1704, 18673, 313, 1168, 29883, 562, 2142,
29897, 263, 425, 14015, 16060, 316, 383, 19498, 316, 29871, 29896, 29929, 29929, 29900, 29892, 712, 409,
25001, 29980, 427, 12201, 29889, 13, 13, 2525, 3001, 316, 29871, 29896, 29953, 16954, 14980, 12306, 267,
316, 29345, 316, 19537, 628, 21035, 29892, 19537, 8068, 343, 560, 1704, 18673, 752, 277, 10243, 1277,
3248, 2174, 16095, 1513, 294, 1702, 560, 28743, 316, 29345, 29889, 1260, 23300, 27712, 560, 29871,
29941, 316, 9019, 316, 29871, 29896, 29929, 29947, 29947, 343, 2186, 466, 29980, 560, 29871, 29906,
29945, 316, 9350, 316, 29871, 29896, 29929, 29947, 29929, 29889, 13, 13, 2277, 317, 28474, 316, 5100,
6396, 13, 13, 6489, 23300, 1040, 29980, 316, 9941, 24627, 294, 29889, 1174, 425, 8633, 364, 14287, 29892,
1869, 29871, 29896, 29953, 16954, 14980, 5221, 3794, 12004, 25227, 8817, 427, 19545, 26161, 29892,
3248, 316, 21135, 316, 19545, 7462, 359, 343, 1232, 11929, 3248, 316, 9941, 29889, 4602, 3248, 27141,
7245, 5114, 316, 9747, 12249, 1029, 4096, 5022, 263, 425, 17329, 364, 14287, 29892, 8334, 1232, 29871,
29947, 7462, 359, 5221, 3794, 12004, 25227, 4396, 427, 3248, 26161, 316, 19545, 29889, 4602, 3248,
27141, 316, 9747, 12249, 1029, 4096, 5022, 263, 425, 1935, 23322, 364, 14287, 29892, 8334, 1232, 29871,
29946, 7462, 359, 5221, 3794, 12004, 25227, 4396, 427, 443, 6651, 12249, 29889, 4602, 3248, 27141,
22215, 928, 5022, 263, 425, 14015, 16060, 316, 383, 19498, 316, 29871, 29896, 29929, 29929, 29900,
29889, 13, 13, 2277, 11243, 666, 359, 5221, 3794, 13, 13, 2369, 18580, 4244, 29892, 1232, 7462, 359, 9512,
3794, 29889, 13, 13, 2277, 22915, 364, 14287, 13, 13, 2277, 29937, 5430, 1129, 29871, 29896, 13, 13, 29286,
28122, 409, 8740, 5022, 427, 1605, 262, 2368, 343, 22354, 4425, 29889, 13, 13, 2277, 29937, 5430, 1129,
29871, 29906, 13, 13, 29286, 28122, 409, 8740, 5022, 427, 21810, 26421, 29889, 13, 13, 2277, 29937, 5430,
1129, 29871, 29941, 13, 13, 29286, 28122, 409, 8740, 5022, 427, 9145, 29976, 29889, 13, 13, 2277, 29937,
5430, 1129, 29871, 29946, 13, 13, 29286, 28122, 409, 8740, 5022, 427, 14756, 29889, 13, 13, 2277, 24153,
364, 14287, 13, 13, 2277, 29937, 5430, 1129, 29871, 29896, 13, 13, 29286, 28122, 409, 8740, 5022, 427,
9702, 9726, 29889, 13, 13, 2277, 29937, 5430, 1129, 29871, 29906, 13, 13, 29286, 28122, 409, 8740, 5022,
427, 379, 898, 10939, 29889, 13, 13, 2277, 5061, 23322, 364, 14287, 13, 13, 29286, 28122, 409, 8740, 5022,
427, 9702, 9726, 29889, 13, 13, 2277, 8867, 8795, 13, 13, 29930, 21581, 29889, 510, 448, 2233, 294, 928,
1061, 423, 316, 23924, 562, 2142, 1702, 425, 14015, 16060, 316]
 }]
}
```


输出说明

返回值	类型	说明
id	string	请求id。
outputs	list	推理结果列表。
name	string	默认"output0"。
shape	list	结构为[1, n]，1表示1维数组，n表示data字段中token结果长度。
datatype	string	"UINT32"。
data	list	推理后生成的token id集合。

3.3.2.4.6 文本推理接口

提供文本推理处理功能。

接口格式

操作类型：**POST**

URL：https://*{ip}*:*{port}***/v2/models/***${MODEL_NAME}[/versions/${MODEL_VERSION}]***/generate**

请求参数

参数	是否必选	说明	取值要求
id	可选	请求id。	stirng，非空。
text_input	必选	推理请求文本。	非空，0<字符数<=16000，支持中英文。tokenizer之后的token数量<=maxSeqLen-maxIterTimes（配置文件读取）。
details	可选	是否返回推理详细输出结果。	bool类型，默认值false。
do_sample	可选	是否做sampling。	bool类型，默认值false。

参数	是否必选	说明	取值要求
max_new_tokens	可选	允许的最大新标记数目。控制从模型生成的文本中添加到最终输出中的最大词汇数量。该字段受到GIMIS配置文件maxIterTimes参数影响，推理token输出长度<=maxIterTimes。	int类型，取值范围(0, maxIterTimes]。默认值20。
repetition_penalty	可选	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，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。
seed	可选	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。	uint_64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。
temperature	可选	控制生成的随机性，较高的值会产生更多样化的输出。	float类型，大于0，默认值1.0。 • 1.0表示不进行计算。 • 大于1.0表示输出随机性提高。
top_k	可选	控制模型生成过程中考虑的词汇范围，只从概率最高的k个候选词中选择。使用限制请参见**使用限制**。	int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值，若不存在则vocabSize取默认值0。
top_p	可选	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。	float类型，取值范围(0, 1]，默认值1.0。

参数	是否必选	说明	取值要求
batch_size	可选	推理请求batch_size。	int类型，大于0，默认值1。

说明

[/versions/${MODEL_VERSION}]字段暂不支持，不传递。

使用样例

请求样例：

```
POST https://<ip>:<port>/v2/models/llama_65b/generate
```

请求消息体：

```
{
  "id":"a123",
  "text_input": "My name is Olivier and I",
  "parameters": {
    "details": true,
    "do_sample": true,
    "max_new_tokens":200,
    "repetition_penalty": 1.1,
    "seed": 123,
    "temperature": 1,
    "top_k": 2147483647,
    "top_p": 0.99,
    "batch_size":100
  }
}
```

响应样例：

```
{
 "id":"a123",
 "model_name": "llama_65b",
 "model_version": null,
 "text_output": "am living in South of France.\nI have been addicted to Jurassic Park since very young. I played some video game versions but especially the great first pinball model from William which reminds me a lot of JPOG1 by song (deluxe). Unfortunately, it stopped working and has been unprofitable for a long time before being exchanged for another game. Fortunately there was the computer version. Nevertheless, it came out only on PC in 2003 when mine was too weak... It's just been a couple of months that the game came out on Mac (a whole 15 years late) with the Version 0.91JAMS ! I know this may be a little antique with the realistic animations and versions today, but the memories are very deep-seated . So thank you all rebuilders for keeping alive wonderful games like this one.\nSince then, I try to keep me updated about this game and test if possible later Alpha. Thank you so much for your work!</s>",
 "details": {
  "finish_reason": "eos_token",
  "generated_tokens": 221,
  "first_token_cost": null,
  "decode_cost": null
 }
}
```

输出说明

返回值	类型	说明
id	string	请求id。

返回值	类型	说明
model_name	string	模型名称。
model_version	string	模型版本。当前未统计该数据，返回null。
text_output	string	推理返回结果。
finish_reason	string	推理结束原因。
generated_tokens	int	推理产生token数量。
first_token_cost	List[token]	文本推理返回，首token产生时间，单位：ms，当前未统计该数据，返回null。
decode_cost	int	decode时间，单位：ms，当前未统计该数据，返回null。

3.3.2.4.7 流式推理接口

提供文本推理处理功能。

接口格式

操作类型：**POST**

URL：https://*{ip}*:*{port}*/v2/models/*${MODEL_NAME}[/versions/${MODEL_VERSION}]*/generate_stream

请求参数

参数	是否必选	说明	取值要求
id	可选	请求id	stirng，非空
text_input	必选	推理请求文本。	非空，0<字符数<=16000，支持中英文。tokenizer之后的token数量<=maxSeqLen-maxIterTimes（配置文件读取）。
details	可选	是否返回推理详细输出结果。	bool类型，默认值false。
do_sample	可选	是否做sampling。	bool类型，默认值false。

参数	是否必选	说明	取值要求
max_new_tokens	可选	允许的最大新标记数目。控制从模型生成的文本中添加到最终输出中的最大词汇数量。该字段受到GIMIS配置文件maxIterTimes参数影响，推理token输出长度<=maxIterTimes。	int类型，取值范围(0, maxIterTimes]。默认值20。
repetition_penalty	可选	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，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。
seed	可选	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。	uint_64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。
temperature	可选	控制生成的随机性，较高的值会产生更多样化的输出。	float类型，大于0，默认值1.0。 • 1.0表示不进行计算。 • 大于1.0表示输出随机性提高。
top_k	可选	控制模型生成过程中考虑的词汇范围，只从概率最高的k个候选词中选择。使用限制请参见**使用限制**。	int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值，若不存在则vocabSize取默认值0。
top_p	可选	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。	float类型，取值范围(0, 1]，默认值1.0。

参数	是否必选	说明	取值要求
batch_size	可选	推理请求batch_size	int类型，大于0，默认值1。

说明

[/versions/${MODEL_VERSION}]字段暂不支持，不传递。

使用样例

请求样例：

```
POST https://<ip>:<port>/v2/models/llama_65b/generate_stream
```

请求消息体：

```
{
  "id":"a123",
  "text_input": "My name is Olivier and I",
  "parameters": {
    "details": true,
    "do_sample": true,
    "max_new_tokens":200,
    "repetition_penalty": 1.1,
    "seed": 123,
    "temperature": 1,
    "top_k": 2147483647,
    "top_p": 0.99,
    "batch_size":100
  }
}
```

响应样例：

```
data:{"id":"a123","model_name":"llama_65b","model_version":null,"text_output":"am","details":
{"generated_tokens":1,"first_token_cost":null,"decode_cost":null}}

data:{"id":"a123","model_name":"llama_65b","model_version":null,"text_output":" passion","details":
{"generated_tokens":2,"first_token_cost":null,"decode_cost":null}}

data:{"id":"a123","model_name":"llama_65b","model_version":null,"text_output":"ate","details":
{"generated_tokens":3,"first_token_cost":null,"decode_cost":null}}

data:{"id":"a123","model_name":"llama_65b","model_version":null,"text_output":" about","details":
{"generated_tokens":4,"first_token_cost":null,"decode_cost":null}}

data:{"id":"a123","model_name":"llama_65b","model_version":null,"text_output":" music","details":
{"generated_tokens":5,"first_token_cost":null,"decode_cost":null}}

data:{"id":"a123","model_name":"llama_65b","model_version":null,"text_output":".","details":
{"generated_tokens":6,"first_token_cost":null,"decode_cost":null}}

data:{"id":"a123","model_name":"llama_65b","model_version":null,"text_output":"\n","details":
{"generated_tokens":7,"first_token_cost":null,"decode_cost":null}}

data:{"id":"a123","model_name":"llama_65b","model_version":null,"text_output":"T","details":
{"generated_tokens":8,"first_token_cost":null,"decode_cost":null}}

data:{"id":"a123","model_name":"llama_65b","model_version":null,"text_output":"od","details":
{"generated_tokens":9,"first_token_cost":null,"decode_cost":null}}

data:{"id":"a123","model_name":"llama_65b","model_version":null,"text_output":"ay","details":
{"generated_tokens":10,"first_token_cost":null,"decode_cost":null}}
```



```
data:{"id":"a123","model_name":"llama_65b","model_version":null,"text_output":"</s>","details":
{"finish_reason":"eos_token","generated_tokens":424,"first_token_cost":null,"decode_cost":null}}
```

输出说明

返回值	类型	说明
id	可选	请求id
model_name	string	模型名称。
model_version	string	模型版本。
text_output	string	推理返回结果。
finish_reason	string	推理结束原因，最后一个token返回该字段。
generated_tokens	int	推理产生token数量。
first_token_cost	List[token]	文本推理返回，首token产生时间，单位：ms，当前未统计该数据，返回null。
decode_cost	int	decode时间，单位：ms，当前未统计该数据，返回null。

3.3.2.5 自研接口

3.3.2.5.1 Slot 统计接口

接口功能

参考Triton格式，自定义的slot统计信息查询接口。

接口格式

操作类型：**GET**

URL：https://*{ip}:{port}*/v2/models/*${MODEL_NAME}[/versions/${MODEL_VERSION}]*/getSlotCount

请求参数

无

> **说明**
>
> [/versions/${MODEL_VERSION}]字段暂不支持，不传递。

使用样例

请求样例：

```
GET https://<ip>:<port>/v2/models/llama_65b/getSlotCount
```


响应样例：

```
{
 "total_slots": 50,
 "free_slots": 50
}
```

响应状态码：200

输出说明

返回值	类型	说明
total_slots	int	推理服务支持的最大batch_size，取值为配置文件中maxBatchSize字段。当前未统计该数据，返回null。
free_slots	int	当前剩余slots字段，通过调度模块管理的参数获取。当前未统计该数据，返回null。

3.3.2.5.2 提前终止请求接口

接口功能

参考Triton接口定义，提供提前终止请求接口。

接口格式

操作类型：**POST**

URL：https://*{ip}*:*{port}*/v2/models/*${MODEL_NAME}[/versions/${MODEL_VERSION}]*/stopInfer

请求参数

参数	是否必选	说明	取值要求
id	必选	推理请求id。	字符串。

说明

[/versions/${MODEL_VERSION}]字段暂不支持，不传递。

使用样例

请求样例：

```
POST https://<ip>:<port>/v2/models/llama_65b/stopInfer
```

请求消息体：


```
{
  "id":"a123"
}
```

响应样例：

```
{
  "id":"a123"
}
```

响应状态码：200

输出说明

返回值	类型	说明
id	string	成功停止推理请求id。

3.3.2.5.3 文本/流式推理接口

接口功能

提供文本/流式推理处理功能。

接口格式

操作类型：**POST**

URL：https://*{ip}*:*{port}*/infer

请求参数

参数名	是否必选	说明	取值要求
inputs	必选	推理请求文本。	非空，0<字符数<=16000，支持中英文。tokenizer之后的token数量<=maxSeqLen-maxIterTimes（配置文件读取）。
do_sample	可选	是否做sampling。	bool类型，默认值false。
repetition_penalty	可选	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，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。

参数名	是否必选	说明	取值要求
seed	可选	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。	uint_64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。
temperature	可选	控制生成的随机性，较高的值会产生更多样化的输出。	float类型，大于0，默认值1.0。 ● 1.0表示不进行计算。 ● 大于1.0表示输出随机性提高。
top_k	可选	控制模型生成过程中考虑的词汇范围，只从概率最高的k个候选词中选择。使用限制请参见**使用限制**。	int类型，取值范围(0, vocabSize)&&(0, 2147483647]，默认值0。 vocabSize是从modelWeightPath路径下的config.json文件中读取的vocab_size值，若不存在则vocabSize取默认值0。
top_p	可选	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。	float类型，取值范围(0, 1)，默认值1.0。
stream	可选	指定返回结果是文本推理还是流式推理。	bool类型，默认值false。
details	可选	是否返回推理详细输出结果。	bool类型，默认值false。

使用样例

请求样例：

```
POST https://<ip>:<port>/infer
```

请求消息体：

```
{
 "inputs": "My name is Olivier and I",
 "stream": true,
 "parameters": {
```



```
  "temperature": 0.5,
  "top_k": 10,
  "top_p": 0.95,
  "do_sample": true,
  "seed": null,
  "repetition_penalty": 1.03,
  "details": true
 }
}
```

文本推理（“stream”=“false”）响应样例：

```
{
 "generated_text": "am a french native speaker. I am looking for a job in the hospitality industry. I",
 "details": {
  "finish_reason": "length",
  "generated_tokens": 20,
  "seed": 846930886
 }
}
```

流式推理（“stream”=“true”）响应样例（使用sse格式返回）：

```
data: {"tokens":{"id":626,"text":"am"}}

data: {"tokens":{"id":263,"text":" a"}}

data: {"tokens":{"id":5176,"text":" French"}}

data: {"tokens":{"id":17739,"text":" photograph"}}

data: {"tokens":{"id":261,"text":"er"}}

data: {"tokens":{"id":2729,"text":" based"}}

data: {"tokens":{"id":297,"text":" in"}}

data: {"tokens":{"id":3681,"text":" Paris"}}

data: {"tokens":{"id":29889,"text":"."}}

data: {"tokens":{"id":13,"text":"\n"}}

data: {"tokens":{"id":29902,"text":"I"}}

data: {"tokens":{"id":505,"text":" have"}}

data: {"tokens":{"id":1063,"text":" been"}}

data: {"tokens":{"id":27904,"text":" shooting"}}

data: {"tokens":{"id":1951,"text":" since"}}

data: {"tokens":{"id":306,"text":" I"}}

data: {"tokens":{"id":471,"text":" was"}}

data: {"tokens":{"id":29871,"text":" "}}

data: {"tokens":{"id":29896,"text":"1"}}

data: {"generated_text":"am a French photographer based in Paris.\nI have been shooting since I was
15","details":{"finish_reason":"length","generated_tokens":20},"tokens":{"id":29945,"text":null}}
```


输出说明

表 3-13 文本推理结果说明

返回值	类型	说明
generated_text	string	推理返回结果，只在最后一次推理结果才返回。
details	object	推理details结果。目前定义以下字段，支持扩展。
finish_reason	string	推理结束原因。
generated_tokens	int	推理结果token数量。
seed	int	如果请求指定了sampling seed，返回该seed值。

表 3-14 流式推理结果说明

返回值	类型	说明
generated_text	string	推理文本结果，只在最后一次推理结果才返回。
details	object	推理details结果，只在最后一次推理结果返回，支持扩展。
finish_reason	string	推理结束原因。
generated_tokens	int	推理结果token数量。
seed	int	如果请求指定了sampling seed，返回改seed值。
token	List[token]	每一次推理的token。
id	int	token id。
text	string	token对应文本。

3.4 命令介绍

3.4.1 seceasy_encrypt

3.4.1.1 encryptFile

命令功能

加密文件。

命令格式

```
./seceasy_encrypt --encryptFile domainId domainCount filePath
```

参数说明

参数	说明
domainId	域ID，白名单规划使用1。
domainCount	域的数量，规划值为2。
filePath	文件路径。

3.4.1.2 encrypt

命令功能

EndPoint开启https时，对服务端密钥口令加密。交互输入明文字符串，输出密文字符串。

命令格式

```
./seceasy_encrypt --encrypt domainId domainCount
```

参数说明

参数	说明
domainId	域ID，当前规划值为1。
domainCount	域的数量，当前规划值为2。

3.4.1.3 activeKey

命令功能

更新主密钥。

命令格式

```
./seceasy_encrypt --activeKey domainId domainCount
```

参数说明

参数	说明
domainId	域ID，当前规划值为1。
domainCount	域的数量，当前规划值为2。

3.4.1.4 secureEraseAllKeystore

命令功能

删除密钥。

须知

- 执行该指令后会擦除密钥，会导致功能异常。
- 仅在卸载MindIE Server时需执行该命令。

命令格式

```
./seceasy_encrypt --secureEraseAllKeystore domainCount
```

参数说明

参数	说明
domainCount	域的数量，当前规划值为2。

3.4.2 mindieservice_backend_connector

命令功能

运行多卡模型时，每张卡上会有一个对应的Rank进程去执行对应的模型分片，该执行文件负责启动Rank进程及Rank进程与主进程通信。

命令格式

说明

该执行文件由主程序调用，不允许手动调用。

```
./mindieservice_backend_connector Rank World_Size Npu_Device_Id Model_Instance_Type Deploy_Type Executor_Type Parent_Pid Shared_Memory_Name Log_Error Log_Warning Log_Info Log_Verbose Log_File
```

参数说明

参数	说明
Rank	当前Rank号。
World_Size	多卡模型张量并行数。
Npu_Device_Id	当前进程使用的npu device Id。
Model_Instance_Type	大模型类型，目前为Standard。
Deploy_Type	部署类型，目前为INTER_PROCESS。

参数	说明
Executor_Type	执行类型，目前为LLM_EXECUTOR_PYTHON。
Parent_Pid	父进程Id。
Shared_Memory_Name	共享内存名称。
Log_Error	是否启用Error级别的日志。
Log_Warning	是否启用Warning级别的日志。
Log_Info	是否启用Info级别的日志。
Log_Verbose	是否启用Verbose级别的日志。
Log_File	日志文件路径。

3.4.3 llm_engine_test

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

```
./bin/llm_engine_test 可选参数1 可选参数2
```

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。

操作步骤

步骤1 配置conf目录下的config.json，请参见《**MindIE安装指南**》中的“附录 > 配置MindIE Server”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。

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）。

- 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3200。

```
{
  "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
```

----结束

3.5 脚本介绍

3.5.1 config_mindie_server_tls_cert.py

脚本功能

EndPoint开启https时，使用该脚本对服务端密钥口令加密。

使用指南

```
python3 scripts/config_mindie_server_tls_cert.py 软件包安装目录
# 样例 python3 scripts/config_mindie_server_tls_cert.py /home/Ascend-mindie-server_{version}_linux-{arch}
```


4 MindIE Client

4.1 功能介绍

MindIE-Client是MindIE-Service的模块之一，在启动MindIE-Server Endpoint服务后，MindIE-Client可以使用HTTPS协议对它发送请求。MindIE-Client提供了多种功能的接口，包括模型推理、请求管理和服务状态查询，用户调用接口即可实现与MindIE-Server通信，提升了易用性。

4.2 应用场景

- 支持以下模型推理接口：
  - 同步推理（token_id to token_id）
  - 异步推理（token_id to token_id）
  - 全量文本推理（text to text）
  - 流式文本推理（text to text）
- 支持以下请求管理接口：
  - 提前终止推理请求
  - 统计slot数量
- 支持以下服务状态查询接口：
  - 查询Server和Model的状态和元数据
  - 查询Model配置

4.3 API 参考（Python）

4.3.1 类参考

4.3.1.1 class AsyncRequest

4.3.1.1.1 类说明

异步请求类，保存了异步推理的结果。

4.3.1.1.2 def __init__(self, greenlet, verbose)

函数功能

类构造函数。

函数原型

```
def __init__(self, greenlet, verbose)
```

参数说明

参数名	参数类型	输入/输出	说明
greenlet	greenlet对象	输入	当前请求对应的协程，用于获取结果。
verbose	bool	输入	是否打印详细日志。

4.3.1.1.3 def get_result(self, timeout)

函数功能

获取异步推理的结果。

函数原型

```
def get_result(self, timeout)
```

约束说明

timeout大于0。

参数说明

参数名	参数类型	输入/输出	说明
timeout	float	输入	表示当前协程获取结果的时间限制，默认为None。

返回值

Result对象，表示推理结果。

4.3.1.2 class MindIEHTTPClient

4.3.1.2.1 类说明

HttpClient类，封装了http请求接口，包括推理类：infer、async_infer、generate_stream、generate和cancel；查询类：is_server_live、is_server_ready、is_model_ready、get_server_metadata、get_model_metadata和get_slot_count。

4.3.1.2.2 成员变量

成员名称	描述
valid_url	URL对象，通过str(valid_url)可以得到string类型的url，如https://127.0.0.1:3128。

4.3.1.2.3 def __init__(self, url, greenlet_size, ssl_options, network_timeout, connection_timeout concurrency, verbose)

函数功能

HttpClient对象初始化。

函数原型

```
def __init__(self, url, greenlet_size, ssl_options, network_timeout, connection_timeout, concurrency, verbose)
```

约束说明

- url的scheme只支持https和http。
- greenlet_size的取值范围：[异步请求数，concurrency]。
- concurrency的取值范围：[异步请求数，300]。
- network_timeout、connection_timeout的取值为大于0。

参数说明

参数名	参数类型	输入/输出	说明
url	str	输入	格式为http://IP:Port 或 https://IP:Port。
greenlet_size	int	输入	协程数量，默认为None。
enable_ssl	bool	输入	是否开启ssl认证，默认为False。

参数名	参数类型	输入/输出	说明
ssl_options	dict	输入	认证鉴权设置，默认为None。当开启https认证，用户需提供ca_certs、certfile、keyfile
ca_certs	str	输入	CA证书的路径
certfile	str	输入	客户端证书路径
keyfile	str	输入	客户端证书密钥路径
network_timeout	float	输入	网络时间限制，默认为600.0。
connection_timeout	float	输入	连接时间限制，默认为600.0。
concurrency	int	输入	并发连接数，默认为1。
verbose	bool	输入	是否输出详细日志，默认为False。

返回值

无

4.3.1.2.4 def close(self)

函数功能

关闭httpclient，在MindIEHTTPClient析构函数中调用。

函数原型

```
def close(self)
```

返回值

无

4.3.1.2.5 def is_server_live(self)

函数功能

判断服务状态是否正常。

函数原型

```
def is_server_live(self)
```

返回值

bool对象，表示服务状态是否正常，如果返回True，则表示正常。

4.3.1.2.6 def is_server_ready(self)

函数功能

判断server的准备状态。

函数原型

```
def is_server_ready(self)
```

返回值

bool对象，表示server是否准备好。

4.3.1.2.7 def is_model_ready(self, model_name, model_version)

函数功能

判断model的准备状态。

函数原型

```
def is_model_ready(self, model_name, model_version)
```

约束说明

- model_name只支持由大小写字母、数字、中横线和下划线组成。
- model_version为非空时只支持由大小写字母、数字、中横线和下划线组成。

参数说明

参数名	参数类型	输入/输出	说明
model_name	String	输入	模型名称。
model_version	String	输入	模型版本，默认为""。

返回值

bool对象，表示model是否准备好。

4.3.1.2.8 def get_server_metadata(self)

函数功能

查询server元数据。

函数原型

```
def get_server_metadata(self)
```


返回值

返回json字典形式的server元数据。

4.3.1.2.9 def get_model_metadata(self, model_name, model_version)

函数功能

查询model的元数据。

函数原型

```
def get_model_metadata(self, model_name, model_version)
```

约束说明

- model_name只支持由大小写字母、数字、中横线和下划线组成。
- model_version为非空时只支持由大小写字母、数字、中横线和下划线组成。

参数说明

参数名	参数类型	输入/输出	说明
model_name	String	输入	模型名称。
model_version	String	输入	模型版本，默认为""。

返回值

返回json字典形式的model元数据。

4.3.1.2.10 def get_model_config(self, model_name, model_version)

函数功能

查询model配置。

函数原型

```
def get_model_config(self, model_name, model_version)
```

约束说明

- model_name只支持由大小写字母、数字、中横线和下划线组成。
- model_version为非空时只支持由大小写字母、数字、中横线和下划线组成。

参数说明

参数名	参数类型	输入/输出	说明
model_name	String	输入	模型名称。

参数名	参数类型	输入/输出	说明
model_version	String	输入	模型版本，默认为""。

返回值

返回json字典形式的model配置。

4.3.1.2.11 def infer(self, model_name, inputs, model_version, outputs, request_id, parameters)

函数功能

实现同步推理。

函数原型

```
def infer(self, model_name, inputs, model_version, outputs, request_id, parameters)
```

参数说明

参数名	参数类型	输入/输出	说明
model_name	str	输入	模型名称。 模型名称只支持由大小写字母、数字、中横线和下划线组成。
inputs	list	输入	模型输入。 目前只支持1个输入，shape取值为[1,n]（ n <= 16000 ），dataType只支持UINT32。
model_version	str	输入	模型版本，默认""。 非空时只支持由大小写字母、数字、中横线和下划线组成。
outputs	list	输入	指定需要返回的模型输出。如果为None则全部返回。目前只支持指定一个输出，name为“output0”。
request_id	str	输入	请求ID，默认""。
parameters	dict	输入	额外的请求参数，包括seed、temperature、top_k、top_p、do_sample和repetition_penalty，默认None。

参数名	参数类型	输入/输出	说明
seed	int64	输入	随机种子数。 取值(0,18446744073709551615]，不传递随机指定。
temperature	float	输入	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。
top_k	int32	输入	控制模型生成过程中考虑的词汇范围，只从概率最高的 k 个候选词中选择，0表示不做top_k。 取值范围：[0,2147483647]，默认值为0。
top_p	float	输入	使用累计概率选择候选词，直到累计概率超过给定的阈值。 取值范围：(0,1]，默认值为1。
do_sample	bool	输入	是否做sampling。
repetition_penalty	float	输入	用于减少在文本生成过程中出现重复片段的概率。 取值范围大于0，默认值为1.0。

返回值

Result对象表示同步推理结果。

4.3.1.2.12 def async_infer(self, model_name, inputs, model_version, outputs, request_id, parameters)

函数功能

实现异步推理。

函数原型

```
def async_infer(self, model_name, inputs, model_version, outputs, request_id, parameters)
```

参数说明

参数名	参数类型	输入/输出	说明
model_name	str	输入	模型名称。 模型名称只支持由大小写字母、数字、中横线和下划线组成。

参数名	参数类型	输入/输出	说明
inputs	list	输入	模型输入。 目前只支持1个输入，shape取值为[1,n]（ n <= 16000 ），dataType只支持UINT32。
model_version	str	输入	模型版本，默认为""。 非空时只支持由大小写字母、数字、中横线和下划线组成。
outputs	list	输入	指定需要返回的模型输出。 如果为None则全部返回。目前只支持指定一个输出，name为“output0”。
request_id	str	输入	请求ID。
parameters	dict	输入	额外的请求参数。 包括：seed、temperature、top_k、top_p、do_sample和repetition_penalty。
seed	int64	输入	随机种子数。 取值(0,18446744073709551615]，不传递随机指定。
temperature	float	输入	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。
top_k	int32	输入	控制模型生成过程中考虑的词汇范围，只从概率最高的k个候选词中选择。0表示不做top_k。 取值范围：[0,2147483647]，默认值为0。
top_p	float	输入	使用累计概率选择候选词，直到累计概率超过给定的阈值。 取值范围：(0,1]，默认值为1。
do_sample	bool	输入	是否做sampling。
repetition_penalty	float	输入	用于减少在文本生成过程中出现重复片段的概率。 取值范围大于0，默认值为1.0。

返回值

AsyncRequest对象表示异步推理结果。

4.3.1.2.13 def generate(self, model_name, prompt, model_version, request_id, parameters)

函数功能

实现全量文本生成。

函数原型

```
def generate(self, model_name, prompt, model_version, request_id, parameters)
```

参数说明

参数名	参数类型	输入/输出	说明
model_name	str	输入	模型名称。 模型名称只支持由大小写字母、数字、中横线和下划线组成。
prompt	str	输入	模型输入字符串。
model_version	str	输入	模型版本，默认为""。 非空时只支持由大小写字母、数字、中横线和下划线组成。
request_id	str	输入	请求ID。
parameters	dict	输入	额外的请求参数，包括seed、temperature、top_k、top_p、do_sample、repetition_penalty、typical_p、batch_size和details。
seed	int64	输入	随机种子数。 取值范围为(0,18446744073709551615]，不传递随机指定。
temperature	float	输入	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。
top_k	int32	输入	控制模型生成过程中考虑的词汇范围，只从概率最高的k个候选词中选择。0表示不做top_k。 取值范围：[0,2147483647]，默认值为0。
top_p	float	输入	使用累计概率选择候选词，直到累计概率超过给定的阈值。 取值范围：(0,1]，默认值为1。
do_sample	bool	输入	是否做sampling。

参数名	参数类型	输入/输出	说明
repetition_penalty	float	输入	用于减少在文本生成过程中出现重复片段的概率。 取值范围大于0，默认值为1.0。
typical_p	float	输入	典型采样，从分布率接近typical_p的单次集合中采样。 取值范围为(0,1]。
batch_size	int	输入	推理请求batch_size。 取值大于0。
details	bool	输入	是否返回details字段。

返回值

Result对象表示全量文本推理结果。

4.3.1.2.14 def generate_stream(self, model_name, prompt, model_version, request_id, parameters)

函数功能

实现流式文本生成。

函数原型

```
def generate_stream(self, model_name, prompt, model_version, request_id, parameters)
```

参数说明

参数名	参数类型	输入/输出	说明
model_name	str	输入	模型名称。 模型名称只支持由大小写字母、数字、中横线和下划线组成。
prompt	str	输入	模型输入字符串。
model_version	str	输入	模型版本，默认为""。 非空时只支持由大小写字母、数字、中横线和下划线组成。
request_id	str	输入	请求ID。

参数名	参数类型	输入/输出	说明
parameters	dict	输入	额外的请求参数，包括seed、temperature、top_k、top_p、do_sample、repetition_penalty、typical_p、batch_size和details。
seed	int64	输入	随机种子数。 取值范围为(0,18446744073709551615]，不传递随机指定。
temperature	float	输入	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。
top_k	int32	输入	控制模型生成过程中考虑的词汇范围，只从概率最高的k个候选词中选择。0表示不做top_k。 取值范围：[0,2147483647]，默认值为0。
top_p	float	输入	使用累计概率选择候选词，直到累计概率超过给定的阈值。
do_sample	bool	输入	是否做sampling。 取值范围：(0,1]，默认值为1。
repetition_penalty	float	输入	用于减少在文本生成过程中出现重复片段的概率。 取值范围大于0，默认值为1.0。
typical_p	float	输入	典型采样，从分布率接近typical_p的单次集合中采样。 取值范围为(0,1]。
batch_size	int	输入	推理请求batch_size。 取值大于0。
details	bool	输入	是否返回details字段。

返回值

Result对象表示流式文本推理结果。

4.3.1.2.15 def cancel(self, model_name, request_id, model_version)

函数功能

提前中止请求。

函数原型

```
def cancel(self, model_name, request_id, model_version)
```

约束说明

- model_name只支持由大小写字母、数字、中横线和下划线组成。
- model_version为非空时只支持由大小写字母、数字、中横线和下划线组成。

参数说明

参数名	参数类型	输入/输出	说明
model_name	String	输入	模型名称。
request_id	String	输入	请求ID。
model_version	String	输入	模型版本，默认为""。

返回值

Result对象表示中止请求的结果。

4.3.1.2.16 def get_slot_count(self, model_name, model_version)

函数功能

获取slot统计结果。

函数原型

```
def get_slot_count(self, model_name, model_version)
```

约束说明

- model_name只支持由大小写字母、数字、中横线和下划线组成。
- model_version为非空时只支持由大小写字母、数字、中横线和下划线组成。

参数说明

参数名	参数类型	输入/输出	说明
model_name	String	输入	模型名称。
model_version	String	输入	模型版本，默认为""。

返回值

Result对象表示slot统计结果。

4.3.1.3 class Input

4.3.1.3.1 类说明

Client输入类。

4.3.1.3.2 def __init__(self, name, shape, datatype)

函数功能

Input对象初始化。

函数原型

```
def __init__(self, name, shape, datatype)
```

参数说明

参数名	参数类型	输入/输出	说明
name	str	输入	输入名称。
shape	array	输入	输入形状。 只支持[1,n]，n为tokenid数量。
datatype	str	输入	输入名称数据类型。 只支持UINT32。

返回值

无

4.3.1.3.3 def get_input_name(self)

函数功能

获取输入名称。

函数原型

```
def get_input_name(self)
```

返回值

返回输入名称。

4.3.1.3.4 def get_input_shape(self)

函数功能

获取输入形状。

函数原型

```
def get_input_shape(self)
```

返回值

返回输入形状。

4.3.1.3.5 def set_input_shape(self, shape)

函数功能

设置输入形状。

函数原型

```
def set_input_shape(self, shape)
```

参数说明

参数名	参数类型	输入/输出	说明
shape	array	输入	输入形状。 只支持[1,n]，n为tokenid数量。

返回值

无

4.3.1.3.6 def get_input_data_type(self)

函数功能

获取输入数据类型。

函数原型

```
def get_input_data_type(self)
```

返回值

返回输入数据类型。

4.3.1.3.7 def initialize_data(self, input_tensor)

函数功能

初始化输入的tensor数据。

函数原型

```
def initialize_data(self, input_tensor)
```


约束说明

input_tensor需要符合Input类要求的数据类型和shape格式。

参数说明

参数名	参数类型	输入/输出	说明
input_tensor	array	输入	输入tensor。

返回值

无

4.3.1.3.8 def get_input_tensor(self)

函数功能

获取输入数据tensor。

函数原型

```
def get_input_tensor(self)
```

返回值

返回输入数据tensor。

4.3.1.4 class RequestedOutput

4.3.1.4.1 类说明

client需要输出类，表示请求返回哪些输入。

4.3.1.4.2 def __init__(self, name)

函数功能

RequestedOutput对象初始化。

函数原型

```
def __init__(self, name)
```

参数说明

参数名	参数类型	输入/输出	说明
name	String	输入	输入名称。

返回值

无

4.3.1.4.3 def get_output_name(self)

函数功能

获取需要输出的名称。

函数原型

```
def get_output_name(self)
```

返回值

返回需要输出的名称。

4.3.1.4.4 def get_output_tensor(self)

函数功能

获取需要输出的tensor。

函数原型

```
def get_output_tensor(self)
```

返回值

返回需要输出的tensor。

4.3.1.5 class Result

4.3.1.5.1 类说明

Result请求结果类。

4.3.1.5.2 def __init__(self, response, verbose)

函数功能

Result对象初始化。

函数原型

```
def __init__(self, response, verbose)
```

参数说明

参数名	参数类型	输入/输出	说明
response	JSON对象	输入	编码后的JSON对象。

参数名	参数类型	输入/输出	说明
verbose	bool	输入	是否打印详细日志。

返回值

无

4.3.1.5.3 def retrieve_output_index_to_numpy(self, index)

函数功能

将指定索引的输出转化成numpy形式。

函数原型

```
def retrieve_output_index_to_numpy(self, index)
```

约束说明

index值大于或等于0。

参数说明

参数名	参数类型	输入/输出	说明
index	Int	输入	表示输出索引。

返回值

返回numpy格式的特定索引的输出

4.3.1.5.4 def retrieve_output_name_to_numpy(self, name)

函数功能

将特定名称的输出转化成numpy形式。

函数原型

```
def retrieve_output_name_to_numpy(self, name)
```

参数说明

参数名	参数类型	输入/输出	说明
name	String	输入	表示输出的名称。

返回值

返回numpy格式的特定名称的输出。

4.3.1.5.5 def get_response(self)

函数功能

获取推理结果。

函数原型

```
def get_response(self)
```

返回值

返回推理结果。

4.3.1.6 class MindIEClientException(Exception)

4.3.1.6.1 类说明

异常类，用于表示Client的异常信息。

4.3.1.6.2 def __init__(self, message)

函数功能

MindIEClientException对象初始化。

函数原型

```
def __init__(self, message)
```

参数说明

参数名	参数类型	输入/输出	说明
message	String	输入	异常信息。

返回值

无

4.3.1.6.3 def get_message(self)

函数功能

获取异常的信息。

函数原型

```
def get_message(self)
```

返回值

异常的信息。

4.4 代码样例

4.4.1 创建客户端

功能接口调用可参考/*$home*/mindieclient/python/test目录下的测试用例。从mindieclient.httpclient中导出需要的模块，调用接口并发送请求。首先可以创建create_client函数，将函数所在文件命名为utils.py，后续可持续使用该方法。

```
import argparse
from mindieclient.python.httpclient import MindIEHTTPClient
def create_client(request_count=1):
    # get argument
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-u",
        "--url",
        required=False,
        default="https://127.0.0.1:1025",
        help="MindIE-Server URL.",
    )
    parser.add_argument(
        "-v",
        "--verbose",
        action="store_true",
        required=False,
        default=True,
        help="Enable detailed information output.",
    )
    parser.add_argument(
        "-s",
        "--ssl",
        action="store_true",
        required=False,
        default=False,
        help="Enable encrypted link with https",
    )
    parser.add_argument(
        "-ca",
        "--ca_certs",
        required=False,
        default="ca.pem",
        help="Provide https ca certificate.",
    )
    parser.add_argument(
        "-key",
        "--key_file",
        required=False,
        default="client.key.pem",
        help="Provide https client certificate.",
    )
    parser.add_argument(
        "-cert",
        "--cert_file",
        required=False,
        default="client.pem",
        help="Provide https client keyfile.",
```

```
    )
    args = parser.parse_args()
    # create client
    try:
        if args.ssl:
            ssl_options = {}
            if args.ca_certs is not None:
                ssl_options["ca_certs"] = args.ca_certs
            if args.key_file is not None:
                ssl_options["keyfile"] = args.key_file
            if args.cert_file is not None:
                ssl_options["certfile"] = args.cert_file
            mindie_client = MindIEHTTPClient(
                url=args.url,
                verbose=args.verbose,
                enable_ssl=True,
                ssl_options=ssl_options,
                concurrency=request_count,
            )
        else:
            mindie_client = MindIEHTTPClient(
                url=args.url, verbose=args.verbose, concurrency=request_count
            )
    except Exception as e:
        raise e
    return mindie_client
```

4.4.2 同步推理

```
import sys
import numpy as np
from mindieclient.python.httpclient import Input, RequestedOutput
from utils import create_client
from mindieclient.python.common import Log

logger = Log(__name__).getlog()

if __name__ == "__main__":
    # get argument and create client
    try:
        mindie_client = create_client()
    except Exception as e:
        logger.exception("Client Creation falied!")
        sys.exit(1)
    # create input and requested output
    inputs = []
    outputs = []
    input_data = np.arange(start=0, stop=16, dtype=np.uint32)
    input_data = np.expand_dims(input_data, axis=0)
    inputs.append(Input("INPUT0", [1, 16], "UINT32"))
    inputs[0].initialize_data(input_data)
    outputs.append(RequestedOutput("OUTPUT0"))
    # apply model inference
    model_name = "llama_65b"
    results = mindie_client.infer(
        model_name,
        inputs,
        outputs=outputs,
    )
    logger.info(results.get_response())
    output_data = results.retrieve_output_name_to_numpy("OUTPUT0")
    logger.info("input_data: %s", np.array2string(input_data))
    logger.info("output_data: %s", np.array2string(output_data))
```

4.4.3 异步推理

```
import sys
import numpy as np
```



```
from mindieclient.python.httpclient import Input, RequestedOutput
from utils import create_client
from mindieclient.python.common import Log

logger = Log(__name__).getlog()

if __name__ == "__main__":
    # get argument and create client
    request_count = 2
    try:
        mindie_client = create_client(request_count)
    except Exception as e:
        logger.exception("Client Creation falied!")
        sys.exit(1)
    # create input and requested output
    inputs = []
    outputs = []
    input_data = np.arange(start=0, stop=16, dtype=np.uint32)
    input_data = np.expand_dims(input_data, axis=0)
    inputs.append(Input("INPUT0", [1, 16], "UINT32"))
    inputs[0].initialize_data(input_data)
    outputs.append(RequestedOutput("OUTPUT0"))
    # apply async inference
    model_name = "llama_65b"
    async_requests = []
    for _ in range(request_count):
        async_requests.append(
            mindie_client.async_infer(
                model_name,
                inputs,
                outputs=outputs,
            )
        )
    # get_result
    for async_request in async_requests:
        result = async_request.get_result()
        logger.info(result.get_response())
        output_data = result.retrieve_output_name_to_numpy("OUTPUT0")
        logger.info("output_data: %s", output_data)
```

4.4.4 全量文本生成

```
import sys
from utils import create_client
from mindieclient.python.common import Log

logger = Log(__name__).getlog()

if __name__ == "__main__":
    # get argument and create client
    try:
        mindie_client = create_client()
    except Exception as e:
        logger.exception("Client Creation falied!")
        sys.exit(1)
    # create input
    prompt = "My name is Olivier and I"
    model_name = "llama_65b"
    parameters = {
        "do_sample": True,
        "temperature": 0.5,
        "top_k": 10,
        "top_p": 0.9,
        "truncate": 5,
        "typical_p": 0.9,
        "seed": 1,
        "repetition_penalty": 1,
        "watermark": True,
        "details": True,
```



```
    }
    # apply model inference
    result = mindie_client.generate(
        model_name,
        prompt,
        request_id="1",
        parameters=parameters,
    )
    logger.info(result.get_response())
```

4.4.5 流式文本生成

```
import sys
from utils import create_client
from mindieclient.python.common import Log

logger = Log(__name__).getlog()

if __name__ == "__main__":
    # get argument and create client
    try:
        mindie_client = create_client()
    except Exception as e:
        logger.exception("Client Creation falied!")
        sys.exit(1)
    # create input
    prompt = "My name is Olivier and I"
    model_name = "llama_65b"
    parameters = {
        "do_sample": True,
        "temperature": 0.5,
        "top_k": 10,
        "top_p": 0.9,
        "truncate": 5,
        "typical_p": 0.9,
        "seed": 1,
        "repetition_penalty": 1,
        "watermark": True,
        "details": True,
    }
    # apply model inference
    results = mindie_client.generate_stream(
        model_name,
        prompt,
        request_id="1",
        parameters=parameters,
    )
    generated_text = ""
    for cur_res in results:
        if cur_res == "</s>":
            break
        generated_text += cur_res
        logger.info("cur text: %s", cur_res)
    logger.info("final generated text: %s", generated_text)
```

4.4.6 查询服务状态

```
import sys
import json
from utils import create_client
from mindieclient.python.common import Log

logger = Log(__name__).getlog()

if __name__ == "__main__":
    # get argument and create client
    try:
```



```
    mindie_client = create_client()
except Exception as e:
    logger.exception("Client Creation falied!")
    sys.exit(1)
model_name = "llama_65b"
if mindie_client.is_server_live():
    logger.info("The server is alive!")
else:
    logger.error("The server is not alive!")
    sys.exit(1)
if mindie_client.is_server_ready():
    logger.info("The server is ready!")
else:
    logger.error("The server is not ready!")
    sys.exit(1)
if mindie_client.is_model_ready(model_name):
    logger.info("The model is ready!")
else:
    logger.error("The model is not ready!")
    sys.exit(1)
server_metadata_dict = mindie_client.get_server_metadata()
logger.info("get_server_metadata: %s", json.dumps(server_metadata_dict))
model_metadata_dict = mindie_client.get_model_metadata(model_name)
logger.info("get_model_metadata: %s", json.dumps(model_metadata_dict))
model_config_dict = mindie_client.get_model_config(model_name)
logger.info("get_model_config: %s", json.dumps(model_config_dict))
# get slots
result = mindie_client.get_slot_count(model_name)
logger.info("gets_slot_count: %s", result.get_response())
```

4.4.7 提前中止请求

```
import sys
from utils import create_client
from mindieclient.python.common import Log

logger = Log(__name__).getlog()

if __name__ == "__main__":
    # get argument and create client
    try:
        mindie_client = create_client()
    except Exception as e:
        logger.exception("Client Creation falied!")
        sys.exit(1)
    # create input
    prompt = "My name is Olivier and I"
    model_name = "llama_65b"
    parameters = {
        "do_sample": True,
        "temperature": 0.5,
        "top_k": 10,
        "top_p": 0.9,
        "truncate": 5,
        "typical_p": 0.9,
        "seed": 1,
        "repetition_penalty": 1,
        "watermark": True,
        "details": True,
    }
    # apply model inference
    results = mindie_client.generate_stream(
        model_name,
        prompt,
        request_id="1",
        parameters=parameters,
    )
    generated_text = ""
    index = 0
```



```
    for cur_res in results:
        index += 1
        if index == 10:
            flag = mindie_client.cancel(model_name, "1")
            if flag:
                logger.info("Test cancel api succeed!")
                sys.exit(0)
            else:
                logger.error("Test cancel api failed!")
                sys.exit(1)
        logger.info("current result: %s", cur_res)
```


A 附录

A.1 环境变量

参数名称	参数说明	取值范围	缺省值
MIES_INSTALL_PATH	安装路径。	路径参数	Ascend-mindie-service_*{version}*_linux-*{arch}*
MIES_CONTINUOUS_BATCHING	是否开启CB。	• 0：关闭CB • 1：开启CB	未设置环境变量时，默认开启CB。
MIES_PYTHON_LOG_TO_FILE	是否向日志文件中输出Python日志。	• 0：关闭 • 1：开启	0
MIES_PYTHON_LOG_TO_STDOUT	是否向控制台输出Python日志。	• 0：关闭 • 1：开启	0
MIES_PYTHON_LOG_LEVEL	Python日志级别。	• FATAL • ERROR • WARNING • INFO • DEBUG	INFO
MIES_PYTHON_LOG_PATH	Python日志保存路径。	路径参数。	/workspace/log/pythonlog.log

参数名称	参数说明	取值范围	缺省值
LD_LIBRARY_PATH	lib所在的路径	路径参数	${MIES_INSTALL_PATH}/lib:$LD_LIBRARY_PATH
ASCEND_SLOG_PRINT_TO_STDOUT	CANNDEV日志打印控制开关。	• 1：打屏 • 0：落盘到“~/ascend”目录	0
ASCEND_GLOBAL_LOG_LEVEL	CANNDEV日志级别。	• 0：debug • 1：info • 2：warn • 3：error	3
ASCEND_GLOBAL_EVENT_ENABLE	设置应用类日志是否开启Event日志。	• 0：关闭Event日志 • 1：开启Event日志	0
ATB_STREAM_SYNC_EVERY_KERNEL_ENABLE	每个Kernel的Execute时就做同步。	• 0：关闭 • 1：开启	0
ATB_STREAM_SYNC_EVERY_RUNNER_ENABLE	每个Runner的Execute时就做同步。	• 0：关闭 • 1：开启	0
ATB_STREAM_SYNC_EVERY_OPERATION_ENABLE	每个Operation的Execute时就做同步。	• 0：关闭 • 1：开启	0
ATB_OPERATION_EXECUTE_ASYNC	异步下发。	• 0：关闭 • 1：开启	1
TASK_QUEUE_ENABLE	推理使能多stream。	• 0：表示推理使能单stream。 • 1：表示推理使能多stream。	1
ATB_OPSRUNNER_KERNEL_CACHE_LOCAL_COUNT	本地缓存个数。	1~1024	8
ATB_OPSRUNNER_KERNEL_CACHE_GLOABL_COUNT	全局缓存个数。	1~1024	16
ATB_LOG_TO_FILE_FLUSH	日志写文件是否刷新。	• 0：关闭 • 1：开启	0

参数名称	参数说明	取值范围	缺省值
HCCL_BUFFSIZE	控制两个NPU之间共享数据的缓存区大小。	大于或等于1，单位：MB。	120
ATB_LAYER_INTERNAL_TENSOR_REUSE	控制每个layer的内部tensor复用。	• 0：关闭 • 1：开启	1
ASDOPS_LOG_TO_FILE	算子库日志是否输出到文件。	• 0：输出到文件 • 1：不输出	0
ASDOPS_LOG_TO_STDOUT	算子库日志打印控制开关。	• 0：不打印日志 • 1：打印日志	0
ASDOPS_LOG_LEVEL	算子库日志级别。	• FATAL • WARN • INFO • DEBUG	FATAL
ASDOPS_MATMUL_PP_FLAG	算子库开启使用PPMATMUL。	• 0：关闭 • 1：开启	1
ASDOPS_LOG_TO_FILE_FLUSH	是否刷新日志写文件。	• 0：关闭 • 1：开启	0
ASDOPS_LOG_TO_BOOST_TYPE	算子库对应加速库日志类型。	-	atb
ASDOPS_TILING_PARSE_CACHE_DISABLE	算子库tilingParse禁止进行缓存优化。	• 0：关闭 • 1：开启	0
OCK_LOG_LEVEL	后处理环境变量。	• TRACE • DEBUG • INFO • WARN • ERROR • FATAL	-
OCK_LOG_TO_STDOUT	后处理环境变量，加速库日志打印控制开关。	• 0：不打印日志 • 1：打印日志	0
ATB_LOG_TO_FILE	加速库环境变量，加速库日志是否输出到文件。	• 0：输出到文件 • 1：不输出	0
ATB_LOG_TO_STDOUT	加速库环境变量，加速库日志打印控制开关。	• 0：不打印日志 • 1：打印日志	0

参数名称	参数说明	取值范围	缺省值
ATB_LOG_LEVEL	加速库环境变量，加速库日志级别。	• TRACE • DEBUG • INFO • WARN • ERROR • FATAL	ERROR
ATB_OPSRUNNER_SETUP_CACHE_ENABLE	加速库环境变量，是否开启SetupCache，当检查到输入和输出没有变化时，不做setup。	• 0：关闭 • 1：开启	1
ATB_OPSRUNNER_KERNEL_CACHE_TYPE	加速库环境变量。	• 0：不开启 • 1：开启本地缓存 • 2：开启全局缓存 • 3：同时开启本地和全局缓存	3
ATB_OPSRUNNER_KERNEL_CACHE_TILING_SIZE	加速库环境变量，tiling默认大小。	1~1073741824，单位：字节	10240
ATB_PROFILING_ENABLE	加速库环境变量，是否开启profiling工具。	• 0：关闭 • 1：开启	0
ATB_WORKSPACE_MEM_ALLOC_ALG_TYPE	workspace内存分配算法使用类型。	• 0：暴力算法 • 1：block分配算法 • 2：有序heap算法 • 3：引入block合并（SOMAS算法退化版）	1
ATB_WORKSPACE_MEM_ALLOC_GLOBAL	加速库环境变量，开启全局中间tensor内存分配控制开关。	• 0：不开启 • 1：开启全局中间tensor内存分配	0

参数名称	参数说明	取值范围	缺省值
ATB_COMPARE_TILING_EVERY_KERNEL	加速库环境变量，每个Kernel运行后，比较运行前和后的NPU上tiling内容是否变化。	• 0：关闭 • 1：开启	0
ATB_HOST_TILING_BUFFER_BLOCK_NUM	加速库环境变量，Context内部HostTilingBuffer块数，通常使用默认值即可。	128~1024	128
ATB_DEVICE_TILING_BUFFER_BLOCK_NUM	加速库环境变量，Context内部DeviceTilingBuffer块数，通常使用默认值即可。	32~1024	32
PYTHONDONTWRITEBYTECODE	服务启动会在bin目录下生成__pycache__目录，bin目录没有写入权限，需要关闭生成__pycache__目录。	• 0：关闭 • 1：开启	1
EP_OPENSSL_PATH	EndPoint开启https认证后，通过该环境变量来指定openssl加载运行时so文件。该环境变量在EndPoint模块启动时自动设置，不需要用户手动设置。	路径参数	${AIE_LLM_INSTALL_PATH}/lib
HSECEASY_PATH	EndPoint开启https认证后，使用HSECEASY工具对秘钥口令进行加密。该环境变量指定HSECEASY加载运行时so文件路径。	路径参数	${AIE_LLM_INSTALL_PATH}/lib
INFER_ENGINE_TYPE	InferEngine后端类型选择，不配置时使用默认同步接口功能后端，配置为RUNTIME_ENGINE后，使用异步接口功能后端。	RUNTIME_ENGINE：异步接口功能后端。	未设置环境变量时，默认使用同步接口功能后端。

环境变量使用样例

```
source /usr/local/Ascend/ascend-toolkit/set_env.sh                # CANN
source /usr/local/Ascend/atb/set_env.sh                           # ATB
source /home/package/atb_models/set_env.sh  ## atb_models
path="${BASH_SOURCE[0]}"
mies_path=$(cd $(dirname $path); pwd )
export MIES_INSTALL_PATH=${mies_path}/latest
export LD_LIBRARY_PATH=${MIES_INSTALL_PATH}/lib:${LD_LIBRARY_PATH}
export PYTHONPATH=${MIES_INSTALL_PATH}/bin:${PYTHONPATH}
export PYTHONDONTWRITEBYTECODE=1            # 服务启动会在bin目录下生成__pycache__目录，bin目录没有写入权限，需要关闭生成__pycache__目录
export MIES_CONTINUOUS_BATCHING=1
export ATB_OPERATION_EXECUTE_ASYNC=1
export TASK_QUEUE_ENABLE=1
export HCCL_BUFFSIZE=120

# 运行时日志
export ASCEND_SLOG_PRINT_TO_STDOUT=0
export ASCEND_GLOBAL_LOG_LEVEL=3
export ASCEND_GLOBAL_EVENT_ENABLE=0

# 加速库日志
export ATB_LOG_TO_FILE=0
export ATB_LOG_TO_FILE_FLUSH=0
export ATB_LOG_TO_STDOUT=0
export ATB_LOG_LEVEL=ERROR

# 模型库日志
export ASDOPS_LOG_TO_FILE=0
export ASDOPS_LOG_TO_STDOUT=0
export ASDOPS_LOG_LEVEL=ERROR

# OCK后处理日志
export OCK_LOG_LEVEL=ERROR
export OCK_LOG_TO_STDOUT=0

# Python日志
export MIES_PYTHON_LOG_TO_FILE=0
export MIES_PYTHON_LOG_TO_STDOUT=0
export MIES_PYTHON_LOG_PATH=/workspace/log/pythonlog.log
export MIES_PYTHON_LOG_LEVEL=INFO
```

A.2 数据集使用

以下是两个用于性能测试的常见数据集，在此提供两个脚本用于自动化加载模型，将数据集转换为tokenid。需要注意OA数据的平均SequenceLen较长，总量超过三千条，在模型体量大（65B及以上）而服务化配置的MaxBatchSize较小时，跑完整个数据集耗时久，可能需要数个小时。

OA 数据集

1. 单击**链接**获取OA数据集。
2. 转换为tokenid方式。

   使用tokenizer_model.encode进行加密。

   python脚本示例参考如下：

```
import csv
from pathlib import Path
import pyarrow.parquet as pq
import glob, os
from transformers import AutoTokenizer
def read_oa(dataset_path, tokenizer_model):
    out_list = []
```



```
    for file_path in glob.glob((Path(dataset_path) / "*.parquet").as_posix()):
        file_name = file_path.split("/")[-1].split("-")[0]
        data_dict = pq.read_table(file_path).to_pandas()
        data_dict = data_dict[data_dict['lang'] == 'zh']
        ques_list = data_dict['text'].to_list()
        for ques in ques_list:
            tokens = tokenizer_model.encode(ques)
            if len(out_list) <= 2048:
                out_list.append(tokens)
            else:
                out_list.append(tokens[0:2048])
    return out_list
def save_csv(file_path, out_tokens_list):
    with open(file_path, 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)
        for row in out_tokens_list:
            csv_writer.writerow(row)
if __name__ == '__main__':
    model_path = "/data/models/baichuan2-7b"
    oa_dir = "/home/xxx/oasst1"
    save_path = "oa_tokens.csv"
    tokenizer_model = AutoTokenizer.from_pretrained(model_path, trust_remote_code=True,
use_fast=True)
    tokens_lists = read_oa(oa_dir, tokenizer_model)
    save_csv(save_path, tokens_lists)
```

A.3 用户信息列表

用户	描述	初始密码	密码修改方法
{mindie-run-usr}	MindIE Server运行用户。	用户自定义。	使用**passwd**命令修改。

A.4 公网网址说明

以下表格中列出了产品中包含的公网网址，没有安全风险。

网址	说明
appro@openssl.org	该邮箱为开源软件OpenSSL引入，是代码中的版权信息声明，系统中无对外交互场景，无安全风险。
http://www.apache.org/licenses/LICENSE-2.0	该网址为版权信息声明，系统中无对外交互场景，无安全风险。

A.5 术语&缩略语

术语/缩略语	含义
AccTransformer	AccTransformer是面向通用模型的推理服务化场景，实现开放、可扩展的推理服务化平台架构，支持对接业界主流推理框架接口，满足大语言模型、文生图等多类型模型的高性能推理需求。
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。
Trition	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。
GMIS	General Model Inference Scheduler，是一个用于模型推理的调度器。它在大型模型训练中起着关键作用，旨在减少计算资源的空闲时间，提高计算资源的利用率，从而加快模型训练和模型推理的进度模型推理调度器，提供各种模型调度能力。
LLM-RT	运行时的插件。
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。
Backend	模型执行器，推理服务化框架后端对接模型推理层模块。
EndPoint	推理服务化协议和接口封装，兼容Triton/OpenAI/TGI/vLLM等第三方框架接口。
DevOps	Development和Operations的组合词，是一种重视“软件开发人员（Dev）”和“IT运维技术人员（Ops）”之间沟通合作的文化、运动或惯例。它通过自动化“软件交付”和“架构变更”的流程，使得构建、测试、发布软件能够更加地快捷、频繁和可靠。
PGP	Pretty Good Privacy，PGP是一种加密通信协议，可用于保护电子邮件、文件和其他数据的机密性和完整性。数字签名是一种数字证书，用于验证文件或电子邮件的发送者身份和文件或电子邮件的完整性。PGP签名指南提供了使用PGP进行数字签名的详细步骤和建议，以确保签名的安全性和有效性。

术语/缩略语	含义
haveged	haveged服务，提供一个简单易用的不可预测随机数生成器，其基于HAVEGE算法。
KMC	Key Management Center，密钥管理系统。用于管理和保护加密算法中使用的密钥。它可以为企业或组织提供安全的密钥存储、密钥分发、密钥轮换、密钥备份和密钥恢复等功能。KMC秘钥库可以确保密钥的安全性和可靠性，防止密钥泄露、丢失或被篡改。同时，KMC秘钥库还可以支持多种加密算法和密钥长度，满足不同应用场景的需求。
CB	Continuous Batching，连续批处理。
HDK	Hardware Developer Kit，硬件开发工具包。

A.6 FAQ

A.6.1 发送请求返回乱码

问题描述

使用curl命令，Client对Server发送请求时，返回如下乱码。

```
root@ubuntu:~# curl  http://[redacted]/info
<!DOCTYPE html
        PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/[redacted]">
<html xmlns="http://www.w3.[redacted]">

<head>
        <meta http-equiv="X-UA-Compatible" content="IE=edge" />
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <meta name="keywords" content="SWG,Proxy,NetentSec" />

        <title>HIS Proxy Notification</title>
        <link rel="icon" href="http://his.huawei.[redacted]o">
        <style type=text/css>
                body {
                        width: 100%;
                        float: none;
                        background-color: #FFFFFF;
                        font-size: 0.75em;
                        margin: 0 auto;
                }

                #header {
                        width: 100%;
                        min-width: 1342px;
                        height: 20px;
                        padding: 5px 0;
                        background-color: black;
                }

                #headerContainer {
                        width: 1200px;
                        height: 20px;
                        margin: auto;
                }
```

原因分析

设置代理后，代理服务器无法识别Service配置的内网IP，无法将请求转发给目标服务器，导致请求失败。

解决方案

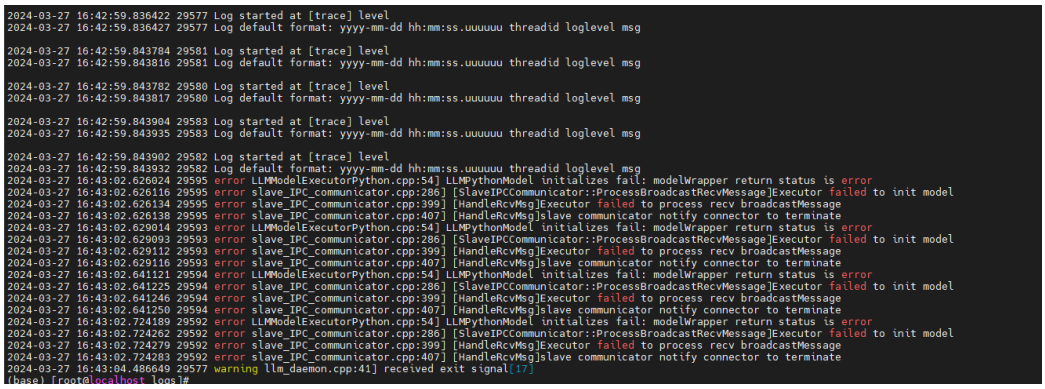
使用以下命令即可解决。

```
unset http_proxy
unset https_proxy
```

A.6.2 部署 Service 服务时出现 LLMPythonModel initializes fail 的报错提示

问题描述

部署Service服务时，出现LLMPythonModel initializes fail的报错，如下图所示。



原因分析

ibis缺少Python依赖。

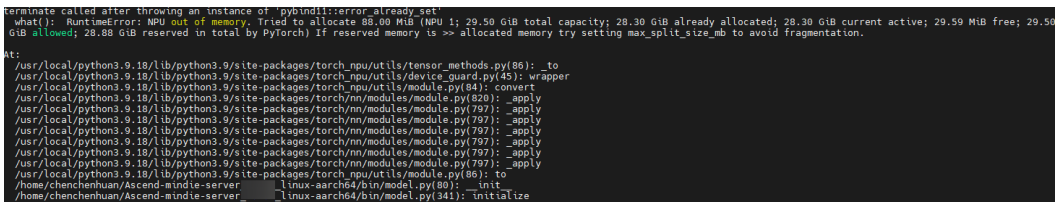
解决步骤

1. 将set_env.sh中的MIES_PYTHON_LOG_TO_FILE修改为1，并打开Python日志。
2. 提前创建/workspace/log目录，重新运行Service，日志会保存到该目录下。
3. 根据日志的报错，安装需要的依赖。

A.6.3 加载模型时出现 out of memory 报错提示

问题描述

部署service服务，加载llama-65b模型时出现out of memory报错提示，如下图所示。



原因分析

权重太大，内存不足。

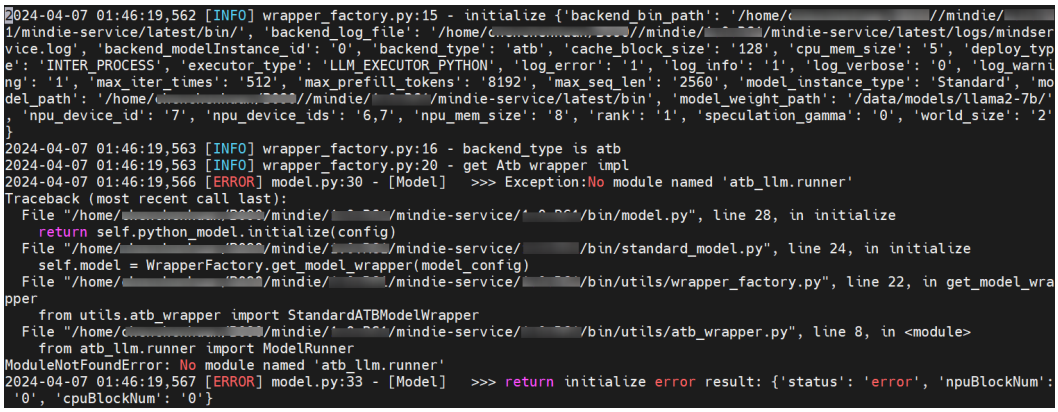
解决步骤

将config.json文件中ModelParam的npuMemSize调小，比如调成8。

A.6.4 部署 Service 服务时，出现 atb_llm.runner 无法 import 报错

问题描述

部署Service服务时，出现atb_llm.runner无法import，如下图所示。



原因分析

由于Python版本不是配套版本3.10，或者pip对应的Python版本不是目标版本3.10，找不到对应的包。可以通过python和pip -V查看对应的Python版本进行确认。

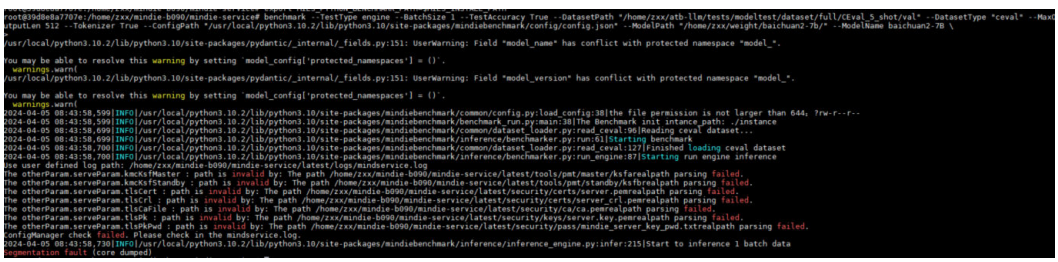
解决步骤

1. 使用以下命令打开bashrc文件。
   ```
   vim ~/.bashrc
   ```
2. 在bashrc文件内添加如下环境变量，保存并退出。
   ```
   export LD_LIBRARY_PATH=/usr/local/python3.10.13/lib:$LD_LIBRARY_PATH
   export PATH=/usr/local/python3.10.13/bin:$PATH
   ```
3. 使用以下命令使环境变量生效。
   ```
   source ~/.bashrc
   ```
4. 使用以下命令建立软链接。
   ```
   ln -s /usr/local/python3.10.0/bin/python3.10 /usr/bin/python
   ln -s /usr/local/python3.10.0/bin/pip3.10 /usr/bin/pip
   ```

A.6.5 部署 service 服务时，找不到 tlsCert 等路径

问题描述

部署service服务时，找不到tlsCert等路径，如下图所示。



原因分析

开启https服务时，未将需要的证书放到对应的目录下。

解决步骤

- 方法一：
  将mindie-service的config.json文件中的httpsEnabled设置成false。
- 方法二：
  将生成服务器证书、CA证书、和服务器私钥等认证需要的文件，放置在对应的目录。

