目录

版本	修订时间	修订人	修订类型	修订章节	修订内容
A0.1	2024-01-29	石子洋	A	123456	serving 支持 yaml 配置，基本接口内容

MindSpore Serving 服务接口说明（dev）

1 配置文件

配置文件放在 serving-gitee/serving/configs 目录下，当前提供双动态的 llama2-70b 和 internLM 模型的 yaml，新增模型 yaml 的话需要按照以下说明进行填写。这里以 llma2_13b_pa_no_act.yam 为例

1.1 model_path 部分，输入 serving 启动所需要的所有文件路径

```
model_path:
    prefill_model: ["/path/to/llama_pa_models/no_act/output_no_act_len/output/mindir_full_checkpoint/rank_0_graph.mindir"]
    decode_model: ["/path/to/llama_pa_models/no_act/output_no_act_len/output/mindir_inc_checkpoint/rank_0_graph.mindir"]
    argmax_model: "/path/to/serving_dev/extends_13b/argmax.mindir"
    topk_model: "/path/to/serving_dev/extends_13b/topk.mindir"
    prefill_ini: ["/path/to/llama_pa_models/no_act/ini/910b_default_prefill.cfg"]
    decode_ini: ["/path/to/llama_pa_models/no_act/ini/910_inc.cfg"]
    post_model_ini: "/path/to/baichuan/congfig/config.ini"
```

1.1.1 prefill_model:

- 数据类型：List[str]
- 说明：全量模型的权重 mindir 路径，List 长度为模型的切分数；
- 类型：必要参数，用户必须传入

1.1.2 decode_model:

- 数据类型：List[str]
- 说明：增量模型的权重 mindir 路径，List 长度为模型的切分数；
- 类型：必要参数，用户必须传入

1.1.3 argmax_model:

- 数据类型：str
- 说明：argmax 模型的权重 mindir 路径；
- 类型：必要参数，用户需通过 post_sampling_model.py 脚本导出

1.1.4 topk_model：

- 数据类型：str
- 说明：topk 模型的权重 mindir 路径；
- 类型：必要参数，用户需通过 post_sampling_model.py 脚本导出

1.1.5 prefill_ini:

- 数据类型：List[str]
- 说明：全量模型的配置文件路径，具体 ini 配置参考 lite 模型的配置说明，多个档位的配置用 ’,’分开；
- 类型：必要参数，用户必须传入；

1.1.6 decode_ini:

- 数据类型：List[str]
- 说明：增量模型的配置文件路径，具体 ini 配置参考 lite 模型的配置说明，多个档位的配置用 ’,’分开；
- 类型：必要参数，用户必须传入；

1.1.7 post_model_ini:

- 数据类型：str
- 说明：后处理模型的配置文件路径，具体 ini 配置参考 lite 模型的配置说明；
- 类型：必要参数，用户必须传入；

1.2 model_config 部分，模型配置参数

```yaml
model_config:
    model_name: 'llama_dyn'
    max_generate_length: 4096
    end_token: 2
    seq_length: [4096]
    vocab_size: 125696
    prefill_batch_size: [1]
    decode_batch_size: [1]   # [16]
    zactivate_len: [4096]
    model_type: 1
    page_attention: True    # check
    batching_strategy: 'continuous'
    current_index: False
```

1.2.1 model_name

- 数据类型：str
- 说明：模型名称, e.g. internlm_*， llama_*；
- 类型：必要参数，用户必须传入

1.2.2 max_generate_length

- 数据类型：int
- 说明：如果开启 PagedAttention，则设置该参数为 seq_len（导出模型的最大 seq length）/ block_size
- 类型：必要参数，用户必须传入

1.2.3 end_token

- 数据类型：int
- 说明：分词器中 eos token（结束符）的 id；
- 类型：必要参数，用户自定义，根据实际分词器词表情况填写；

1.2.4 seq_length

- 数据类型：List[int]
- 说明：适配 prefill seq length 分档， list 为空时表示纯动态，设置一个档位时为静态 seq，设置多个档位，表示动态 seq 分档
- 类型：配合 seq_type 字段使用

1.2.5 vocab_size

- 数据类型：int
- 说明：分词器的词表长度；
- 类型：必要参数，用户自定义，根据实际分词器词表情况填写；

1.2.6 prefill_batch_size

- 数据类型：List[int]
- 说明：适配 prefill 动态分档，设置一个值时为静态 batch 模型；
- 类型：必要参数，用户必须传入，根据实际模型的配置情况填写

1.2.7 decode_batch_size

- 数据类型：List[int]
- 说明：适配 decode 动态分档，设置一个值时为静态 batch 模型；
- 类型：必要参数，用户必须传入，根据实际模型的配置情况填写

1.2.8 zactivate_len

- 数据类型：List[int]
- 说明：适配 act len 优化
- 类型：必要参数，用户自定义，根据实际部署的模型决定；
- 使用示例： [512, 1024, 2048, 4096] 配合 ini 使用，decode.ini 增加

```
[ge_graph_options]
ge.inputShape=batch_index:-1;batch_valid_length:
-1;input_position:-1;tokens:-1,1;zactivate_len:-1
ge.dynamicDims=1,1,1,1,512;1,1,1,1,1024;1,1,1,1,20
48;1,1,1,1,4096;8,8,8,8,512;8,8,8,8,1024;8,8,8,8,2048;
8,8,8,8,4096;16,16,16,16,512;16,16,16,16,1024;16,16
,16,16,2048;16,16,16,16,4096;
ge.dynamicNodeType=1
```

1.2.9 model_type

- 数据类型：str
- 说明：模型类型，用于构造输入使用，针对动态 seq，静态 seq 输入设置为”static”，动态设置为”dyn”；
- 类型：必要参数，用户自定义，根据实际部署的模型决定；

1.2.10 page_attention

- 数据类型：bool
- 说明：当启动 PagedAttention 算法时，将参数设置为 True
- 类型：可选参数，默认为 False；

1.2.11 batching_strategy

- 数据类型：str
- 说明：组 batch 的策略，当前支持 static batch 和 continuous batching 两种模式，static batch 该参数设置为'static'， continuous batching 设置为'continuous'
- 类型：可选参数，默认为'continuous'；

1.2.12 current_index

- 数据类型：bool
- 说明：ft 分支旧模型的输入参数，dev 分支没有此参数
- 类型：可选参数，默认为' False'；

1.2.13 pad_token_id

- 数据类型: int
- 说明：模型做输入如果要做 padding，padding 的 token_id，根据模型的分词器配置；
- 类型：必须参数，默认为 0（llama 的 pad_token_id）；

1.3 serving_config 部分:

```
serving_config:
    agent_ports: [61166]
    start_device_id: 0
    server_ip: 'localhost'
    server_port: 61155
```

1.3.1 agent_ports

- 数据类型：List[int]
- 说明：模型并行下，每份模型对应一个 socket server 进行管理，需要提供 socket server 的 port。size 需要和 prefill_model 一致
- 类型：必要参数

1.3.2 start_device_id

- 数据类型：int
- 说明：使用的 NPU 的起始 device id
- 类型：可选，默认为 0 卡

1.3.3 server_ip

- 数据类型：str
- 说明：serving server 侧 FastAPI 服务的 ip
- 类型：可选，默认为'localhost'

1.3.4 server_port

- 数据类型：int
- 说明：serving server 侧 FastAPI 服务的 port
- 类型：必要参数

1.4 pa_config 部分

```
30  pa_config:
31      num_blocks: 512
32      block_size: 16
33      decode_seq_length: 4096
```

当 model_config 中的 page_attention 为 True 时，需要根据模型设置此项

1.4.1 num_blocks

- 数据类型：int
- 说明： PA 预申请内存块总量
- 类型：必要参数

1.4.2 block_size

- 数据类型：int
- 说明：PA 单内存块容纳 slot（token）数量
- 类型：必要参数

1.4.3 decode_seq_length

- 数据类型：int
- 说明：用于计算每个请求的最大内存块申请量，公式：max_num_block_per_seq = decode_seq_length /block_size
- 类型：必要参数

1.5 tokenizer 部分，自定义分词器

```
35 tokenizer:
36     type: LlamaTokenizer
37     vocab_file: '/path/to/llama_pa_models/output/tokenizer_llama2_13b.model'
```

type: 自定义分词器名称
vocab_file: 模型对应的 tokenizer.model
自定义分词器参考自定义 tokenizer

1.6 模型入参部分

```
39 basic_inputs:
40     type: LlamaBasicInputs
41
42 extra_inputs:
43     type: LlamaExtraInputs
44
45 warmup_inputs:
46     type: LlamaWarmupInputs
```

basic_inputs 包括[input_ids, current_index, init_reset, valid_length, decode_batch_index]
extra_inputs 包括所有除了上述参数以外的模型入参，如 act_len
warmup_inputs 模型 warmup 所需要的入参
输入自定义部分参考自定义 inputs

2 自定义模块

2.1 自定义 tokenizer

在 mindspore_serving/models/tokenizer 目录创建 xxx_tokenizer.py

- 定义 class XxxTokenizer, 类的初始化参数第一个必须为 vocab_file, 然后调用 @Registers.TOKENIZER.register()进行注册

2.2 自定义 inputs

在 mindspore_serving/models/model_inputs 目录创建 xxx_inputs.py

- 定义 class XxxBasicInputs(BaseBasicInputs)，入参为(input_ids, current_index, init_reset, batch_valid_length, *args)，用@Registers.BASIC_INPUTS.register()进行注册
- 定义 class XxxExtraInputs(BaseExtraInputs)，入参为(input_ids, current_index, init_reset, is_prefill, valid_length, **kwargs)。注意 extra inputs 必须得通过入参计算而来，比如 act_len 就是通过 valid_length 计算来的，用@Registers.EXTRA_INPUTS.register()进行注册
- 定义 class XxxWarmupInputs(BaseWarmupInputs)，入参为(seq_length, batch_size, full_model, valid_length=None, **kwargs)，用@Registers.WARMUP_INPUTS.register()进行注册

3 client 请求入参配置

3.1 流式推理

每次推理请求结果按照单个单词依次返回，最后支持返回所有生成单词。

URI

- URI 格式

```
POST /models/{mode_type}/generate_stream
```

- 参数描述

参数	是否必选	类型	描述
inputs	是	String	用户输入提示
parameters	是	Json	生成参数

请求

- 请求样例

```
POST      https://endpoint/models/{mode_type}/generate_stream
{
      "inputs":"what is Monetary Policy?",
      "parameters":{
      "max_new_tokens":20,
      "temperature":1,
      "top_k":1,
      "top_p":1,
      "do_sample":"False",
      "repetition_penalty":1,
      "return_full_text":"True"
}
```

- 参数说明

参数	是否必选	类型	描述	取值
do_sample	否	bool	后处理是否采样	默认值为 False
max_new_tokens	否	int	最大生成数量	默认值为 300
repetition_penalty	否	float	重复单子惩罚参数	默认值为 1.0

参数	是否必选	类型	描述	取值
temperature	否	float	后处理随机性参数	默认值为 1.0
top_k	否	int	后处理 top_k 参数	默认值为 3，如果请求给的 top_k < 0，调整为 0
top_p	否	float	后处理 top_p 参数	默认值为 1.0，如果请求给的 top_p < 0.01，调整为 0.01,top_p > 1.0 调整为 1.0
return_full_text	否	bool	最后是否返回所有生成	默认值为 True
return_protocol	否	str	默认使用 sse 协议返回,其他情况均使用 fastapi 的流式返回（StreamingResponse）	默认值为’sse’

响应

- 流式中间响应样例

```
data:{
"token": {"text": "xx"}
}
```

1. 流式最后响应样例

```
data:{
"generated_text": "xxxxxxxxxxxxxxxxxxxxxxxx"
}
```

3.2 非流式推理

功能介绍

每次推理请求结果一次返回所有生成单词。

URI

- URI 格式

```
POST /models/{mode_type}/generate
```

- 参数描述

参数	是否必选	类型	描述
inputs	是	String	用户输入提示
parameters	是	Json	生成参数

- Parameters 字段描述

请求

- 请求样例

```
POST https://endpoint/models/{mode_type}/generate
{
"inputs":"what is Monetary Policy?",
"parameters":{
    "max_new_tokens":20,
    "temperature":1,
    "top_k":1,
    "top_p":1,
    "do_sample":"False",
    "repetition_penalty":1
}
```

- 参数说明

参数	是否必选	类型	描述	取值（同流式请求）
do_sample	否	bool	后处理是否采样	-
max_new_tokens	否	int	最大生成数量	-
repetition_penalty	否	float	重复单子惩罚参数	-
temperature	否	float	后处理随机性参数	-
top_k	否	int	后处理 top_k 参数	-
top_p	否	float	后处理 top_p 参数	-

响应

- 响应样例

```
{
"generated_text": "xxxxxxxxxxxxxxxxxxxxxxxx"
}
```

3.3 使能 Sample 推理

参数设置

“do_sample”设置为“True”

请求

- 请求样例

```
POST https://endpoint/models/{mode_type}/generate
{
"inputs":"what is Monetary Policy?",
"parameters":{
    "max_new_tokens":20,
    "temperature":1,
    "top_k":1,
    "top_p":1,
    "do_sample":"True",
    "repetition_penalty":1
}
```

3.4 关闭 Sample 推理

参数设置

“do_sample”设置为“False”

请求

- 请求样例

```
POST https://endpoint/models/{mode_type}/generate
{
"inputs":"what is Monetary Policy?",
"parameters":{
    "max_new_tokens":20,
    "do_sample":"False",
}
```

4 Serving 推理 API 接口

4.1 generate_answer(request_id, **params)

使用方法：

1、实例化 llm_server = LLMServer()

2、准备好请求 request_id 及入参，params;

3、results = llm_server.generate_answer(request_id, **params)

4、params 如下：

```
params = {
    "prompt": request.inputs,
    "do_sample": request.parameters.do_sample,
    "top_k": request.parameters.top_k,
    "top_p": request.parameters.top_p,
    "temperature": request.parameters.temperature,
    "repetition_penalty": request.parameters.repetition_penalty,
    "max_token_len": request.parameters.max_new_tokens
}
```

5、返回值为 async_generator 对象；

6、使用 API 方式调用的，可以将 async_generator 对象解析成流式的或非流式的结果，参照代码 clinet/server_app_post.py, 以下是以 sse 协议封装的用例。

```
async def get_stream_res_sse(request, results):
    all_texts = ""
    tokens_list = []
    async for result in results:
        for index, output in enumerate(result.outputs):
            answer_texts = output.text
            res_list = {
                "id": output.index,
                "logprob": output.logprob,
                "special": output.special,
                "text": answer_texts
            }
            tokens_list.append(res_list)
            all_texts += answer_texts

        ret = {
            "event": "message",
            "retry": 30000,
            "details": None,
            "generated_text": all_texts,
            "tokens": tokens_list,
            "top_tokens": [
                [tokens_list[0]]
            ],
        }
        yield (json.dumps(ret, ensure_ascii=False) + '\n').encode("utf-8")
```

5 Serving 状态获取接口

5.1 获取当前时刻的 batch_size 大小：

```
async def _get_batch_size():
    global llm_server
    batch_size = llm_server.get_bs_current()
    ret = {'event': "message", "retry": 30000, "data": batch_size}
    yield json.dumps(ret, ensure_ascii=False)
```

```
@app.get("/serving/get_bs")
async def get_batch_size():
    return EventSourceResponse(_get_batch_size(),
                               media_type="text/event-stream",
                               ping_message_factory=lambda: ServerSentEvent(**{"comment": "You can't see this ping"}),
                               ping=600)
```

以 sse 的形式提供，同流式和非流式的接收方法：

```
(sc1027) root@localhost:/home/sc/serving_1120$ curl 127.0.0.1:9811/serving/get_bs      -X GET
data: {"event": "message", "retry": 15000, "data": 0}
```

5.2 获取当前时刻请求队列里还在等待推理的请求数

同 5.1

```
async def _get_request_numbers():
    global llm_server
    queue_size = llm_server.get_queue_current()
    ret = {'event': "message", "retry": 30000, "data": queue_size}
    yield json.dumps(ret, ensure_ascii=False)
```

```
@app.get("/serving/get_request_numbers")
async def get_request_numbers():
    return EventSourceResponse(_get_request_numbers(),
                               media_type="text/event-stream",
                               ping_message_factory=lambda: ServerSentEvent(**{"comment": "You can't see this ping"}),
                               ping=600)
```

以 sse 的形式提供，同流式和非流式的接收方法：

```
(sc1027) root@localhost:/home/sc/serving_1120$ curl 127.0.0.1:9811/serving/get_request_numbers      -X GET
data: {"event": "message", "retry": 15000, "data": 0}
```

6 错误码

目前包含的错误码总结如下表：

错误码返回值	错误原因
202	分词长度校验，超过模型输入
203	解分词超过模型支持范围
204	入参超过边界值，返回警告和默认值
301	请求队列超过了设定阈值，返回警告