MindSpeed-Core-MS API 文档

发布 ***r0.1.0***

MindSpeed-Core-MS

2024 年 **12** 月 **30** 日

Contents

CHAPTER 1

MindSpeed-Core-MS API

1.1 MindSpeed-Core-MS API

mindspeed_ms.core.config.AllConfig	此配置项包含所有其他配置项，将在初始化配置函数中用作默认配置。
mindspeed_ms.core.config.DatasetConfig	数据集配置类。
mindspeed_ms.core.config.init_configs_from_yaml	从指定的 YAML 配置文件初始化 Config 类实例。
mindspeed_ms.core.config.ModelParallelConfig	模型并行配置类。
mindspeed_ms.core.config.MoEConfig	专家混合（Mixture of Experts，MoE）配置类。
mindspeed_ms.core.config.TrainingConfig	训练配置类。
mindspeed_ms.core.config.TransformerConfig	Transformer 配置类。
mindspeed_ms.core.parallel_state.get_context_parallel_rank	返回上下文并行组中的 rank。
mindspeed_ms.core.parallel_state.get_context_parallel_world_size	返回上下文并行组的 world 大小。
mindspeed_ms.core.parallel_state.initialize _model_parallel	初始化模型数据并行组。
mindspeed_ms.core.tensor_parallel.ColumnParallelLinear	稠密线性层计算，在权重的第二个维度按照张量并行大小进行切分，并行计算。
mindspeed_ms.core.tensor_parallel.RowParallelLinear	稠密线性层计算，在权重的第一个维度按照张量并行大小进行切分，并行计算。
mindspeed_ms.core.tensor_parallel.VocabParallelCrossEntropy	交叉熵损失函数的并行接口。
mindspeed_ms.core.tensor_parallel.VocabParallelEmbedding	在词汇维度并行的嵌入计算。

下页继续

表 1 – 续上页

mindspeed_ms.legacy.model.eos_mask.EosMask	生成特定 token 对应的注意力掩码。
mindspeed_ms.legacy.model.gpt_model.GPTModel	生成式预训练 Transformer（GPT）的实现，是一个仅有解码器的 Transformer 模型。
mindspeed_ms.legacy.model.language_model.Embedding	embedding 层包含 word embedding、position embedding 和 tokentypes embedding。
mindspeed_ms.legacy.model.language_model.get_language_model	使用此函数来获取语言模型。
mindspeed_ms.legacy.model.module.Module	具有流水线支持的特定扩展单元。
mindspeed_ms.legacy.model.moe.experts.SequentialMLP	定义 SequentialMLP 模块。
mindspeed_ms.legacy.model.moe.moe_layer.MoELayer	专家层。
mindspeed_ms.legacy.model.moe.router.TopKRouter	Top-K 路由，负责根据输入数据计算分数，并选择 Top-K 个专家来处理输入数据。
mindspeed_ms.legacy.model.moe.token_dispatcher.MoEAlltoAllTokenDispatcher	在 MoE 架构中，MoEAlltoAllTokenDispatcher 调度器负责将 token 令牌分配给各个专家进行处理，并将处理后的结果重新组合回原始的 token 顺序。
mindspeed_ms.legacy.model.ParallelAttention	该类表示并行注意力机制。
mindspeed_ms.legacy.model.ParallelLMLogits	计算 vocab 中每一个 token 的 logits。
mindspeed_ms.legacy.model.ParallelMLP	并行前馈模块实现。
mindspeed_ms.legacy.model.ParallelTransformer	Transformer 模块。
mindspeed_ms.legacy.model.ParallelTransformerLayer	单独的一层 transformer。
mindspeed_ms.legacy.model.RotaryEmbedding	用于语言模型的旋转位置嵌入。
mindspeed_ms.legacy.model.transformer.CoreAttention	核心注意力机制，用于计算查询、键和值层之间的注意力权重和上下文表示。
mindspeed_ms.legacy.model.TransformerLanguageModel	Transformer 语言模型。
mindspeed_ms.training.loss_func.LossWithMask	使用掩码和均值计算损失。
mindspeed_ms.training.training.pretrain	预训练接口。

1.1.1 mindspeed_ms.core.config.AllConfig

class mindspeed_ms.core.config.**AllConfig**(***kwargs*)

此配置项包含所有其他配置项，将在初始化配置函数中用作默认配置。

参数：

- **kwargs** (dict) - 额外的关键字配置参数。

支持平台：

`Ascend`

样例：

```
>>> from mindspeed_ms.core.config import AllConfig
>>> all_config = [AllConfig]
```

1.1.2 mindspeed_ms.core.config.DatasetConfig

class mindspeed_ms.core.config.**DatasetConfig**(*dataset_dir: str = './dataset', shuffle: bool = False, batch_size: int = 1, micro_batch_num: int = 1, train_samples: int = 0, data_layout: str = 'BSH', eos_token_id: int = 0, **kwargs*)

数据集配置类。

参数：

- **dataset_dir** (str，可选) - 数据集文件目录。默认值：`"./dataset"`。
- **shuffle** (bool，可选) - 是否打乱数据集。默认值：`False`。
- **batch_size** (int，可选) - 一次迭代（iteration）所处理的数据样本的数量。默认值：`1`。
- **micro_batch_num** (int，可选) - 使用流水线并行或者梯度累积时的微批数量。默认值：`1`。
- **train_samples** (int，可选) - 训练样本数量。默认值：`0`。
- **data_layout** (str，可选) - 输入布局。默认值：`"BSH"`。
- **eos_token_id** (int，可选) - 序列结束（EoS，End-of-Sequence）的 token id。默认值：`0`。
- **kwargs** (dict) - 额外的关键字配置参数。

支持平台：

`Ascend`

样例：

```
>>> from mindspeed_ms.core.config import DatasetConfig
>>> dataset_config = DatasetConfig()
```

1.1.3 mindspeed_ms.core.config.init_configs_from_yaml

mindspeed_ms.core.config.**init_configs_from_yaml**(*file_path: str, config_classes=None, **kwargs*)

从指定的 YAML 配置文件初始化 Config 类实例。

参数：

- **file_path** (str) - YAML 配置文件路径。
- **config_classes** (Union[list[BaseConfig], None]，可选) - 待初始化的 Config 类，支持 [TrainingConfig，ModelParallelConfig，OptimizerConfig，DatasetConfig，LoraConfig，TransformerConfig，MoEConfig]。当没有上述 Config 类传入时，所有已知的配置项将被初始化为*mindspeed_ms.core.config.AllConfig*。默认值：`None`。
- **kwargs** (dict) - 额外的关键字配置参数。

返回：

- 返回初始化的 Config 类的实例。当没有 Config 类传入时，返回 *AllConfig* 类的实例。

异常：

- **ValueError** - 入参 *file_path* 不是字符串。
- **ValueError** - 入参 *file_path* 不是以 yaml 或者 yml 结尾。

支持平台：

`Ascend`

样例：

```
>>> from mindspeed_ms.core.config import init_configs_from_yaml
>>> config_file = "/path/to/config/file"
>>> all_config = init_configs_from_yaml(config_file)
```

1.1.4 mindspeed_ms.core.config.ModelParallelConfig

class mindspeed_ms.core.config.**ModelParallelConfig**(*tensor_model_parallel_size: int = 1*, *pipeline_model_parallel_size: int = 1*, *context_parallel_size: int = 1*, *context_parallel_algo: str = 'ulysses_cp_algo'*, *ulysses_degree_in_cp: int = None*, *expert_model_parallel_size: int = 1*, *virtual_pipeline_model_parallel_size: int = None*, *sequence_parallel: bool = False*, *recv_dtype: str = 'float32'*, *zero_level: str = None*, *standalone_embedding_stage: bool = False*, *overlap_grad_reduce: bool = False*, *gradient_accumulation_fusion: bool = False*, *overlap_p2p_comm: bool = True*, *use_cpu_initialization: bool = False*, *deterministic_mode: bool = False*, *num_layer_list: list = None*, *recompute_config: dict = None*, *recompute: str = None*, *select_recompute: str = None*, *select_comm_recompute: str = None*, *variable_seq_lengths: bool = False*, ***kwargs*)

模型并行配置类。

参数：

- **tensor_model_parallel_size** (int，可选) - 张量并行的维数。默认值：`1` 。
- **pipeline_model_parallel_size** (int，可选) - 使用流水线并行时的阶段数。默认值：`1` 。
- **context_parallel_size** (int，可选) - 上下文并行的维度。默认值：`1` 。
- **context_parallel_algo** (str，可选) - 上下文并行算法。默认值：`"ulysses_cp_algo"` 。可选项：[`"ulysses_cp_algo"`，`"megatron_cp_algo"`，`"hybrid_cp_algo"`]。
- **ulysses_degree_in_cp** (int，可选) - 当 `--context-parallel-algo` 设置为 `hybrid_cp_algo` 且 ring-attention 并行度设置为 `cp//ulysses` 时，定义 ulysses 并行度的程度。默认值：`None` 。
- **expert_model_parallel_size** (int，可选) - 专家并行的维度。默认值：`1` 。
- **virtual_pipeline_model_parallel_size** (int，可选) - 使用虚拟流水线并行（VPP）时的阶段数。默认值：`None` 。
- **sequence_parallel** (bool，可选) - 启用序列并行。默认值：`False` 。
- **recv_dtype** (str，可选) - 使用流水线并行时 p2p 通信的通信数据类型。默认值：`"float32"` 。
- **zero_level** (str，可选) - ZeRO 优化器的级别，如果为 `None` ，则不会使用 ZeRO 优化器。默认值：`None` 。
- **standalone_embedding_stage** (bool，可选) - 启用独立嵌入。默认值：`False` 。
- **overlap_grad_reduce** (bool，可选) - 启用重叠梯度规约。默认值：`False` 。
- **gradient_accumulation_fusion** (bool，可选) - 在线性反向阶段启用梯度累加。默认值：`False` 。
- **overlap_p2p_comm** (bool，可选) - 在流水线交错模式下启用重叠 p2p 通信。默认值：`True` 。
- **use_cpu_initialization** (bool，可选) - 使用 CPU 初始化。默认值：`False` 。
- **deterministic_mode** (bool，可选) - 确定性计算模式。默认值：`False` 。

- **num_layer_list** (list，可选) - 自定义流水线并行模型层划分。默认值：`None` 。
- **recompute_config** (dict，可选) - 重计算配置。默认值：`None` 。
- **recompute** (str，可选) - 启用重计算。默认值：`None` 。
- **select_recompute** (str，可选) - 启用选择重计算。默认值：`None` 。
- **select_comm_recompute** (str，可选) - 启用选择通信重计算。默认值：`None` 。
- **variable_seq_lengths** (bool，可选) - 启用可变序列长度。默认值：`False` 。
- **kwargs** (dict) - 额外的关键字配置参数。

支持平台：

`Ascend`

样例：

```
>>> from mindspeed_ms.core.config import ModelParallelConfig
>>> parallel_config = ModelParallelConfig()
```

1.1.5 mindspeed_ms.core.config.MoEConfig

class mindspeed_ms.core.config.**MoEConfig**(*num_experts: int = 1*, *moe_grouped_gemm: bool = False*, *moe_router_topk: int = 2*, *moe_router_load_balancing_type: str = 'none'*, *moe_token_dispatcher_type: str = 'alltoall'*, *use_self_defined_alltoall: bool = False*, *moe_expert_capacity_factor: float = None*, *moe_pad_expert_input_to_capacity: bool = False*, *moe_token_drop_policy: str = 'probs'*, *moe_aux_loss_coeff: float = 0.0*, *moe_z_loss_coeff: float = None*, *moe_input_jitter_eps: float = None*, ***kwargs*)

专家混合（Mixture of Experts，MoE）配置类。

参数：

- **num_experts** (int，可选) - 专家的数量。默认值：`1` 。
- **moe_grouped_gemm** (bool，可选) - 使能分组 gemm。默认值：`False` 。
- **moe_router_topk** (int，可选) - TopK 路由数。默认值：`2` 。
- **moe_router_load_balancing_type** (str，可选) - MoE 路由负载均衡算法的类型。可选项：[`"aux_loss"` , `"none"`]。默认值：`"none"` 。
- **moe_token_dispatcher_type** (str，可选) - Moe 令牌调度算法的类型。可选项：[`'alltoall'`]。默认值：`'alltoall'` 。
- **use_self_defined_alltoall** (bool，可选) - 使用自定义的 alltoall 操作符。默认值：`False` 。
- **moe_expert_capacity_factor** (float，可选) - 每个专家的容量因子。默认值：`None` 。
- **moe_pad_expert_input_to_capacity** (bool，可选) - 是否填充每个专家的输入以匹配专家容量长度。默认值：`False` 。
- **moe_token_drop_policy** (str，可选) - 令牌弃置策略。默认值：`"probs"` 。
- **moe_aux_loss_coeff** (float，可选) - 专家混合辅助损失系数。默认值：`0.0` 。
- **moe_z_loss_coeff** (float，可选) - 专家混合 Z 损失系数。默认值：`None` 。
- **moe_input_jitter_eps** (float，可选) - 通过应用具有指定 epsilon 值的抖动为输入张量添加噪声。默认值：`None` 。

- **kwargs** (dict) - 额外的关键字配置参数。

支持平台：

`Ascend`

样例：

```
>>> from mindspeed_ms.core.config import MoEConfig
>>> moe_config = MoEConfig(num_experts=4, moe_router_topk=2)
```

1.1.6 mindspeed_ms.core.config.TrainingConfig

class mindspeed_ms.core.config.**TrainingConfig**(*parallel_config: ModelParallelConfig*, *dataset_config: DatasetConfig = DatasetConfig()*, *lora_config: LoraConfig = LoraConfig()*, *seed: int = None*, *output_dir: str = './output'*, *training_iters: int = 0*, *epochs: int = None*, *log_interval: int = None*, *eval_interval: int = None*, *save_interval: int = None*, *best_metric_comparison: str = None*, *eval_metric: str = None*, *grad_clip_kwargs: dict = None*, *loss_scale: Union[float, int] = None*, *loss_scale_value: Union[float, int] = None*, *loss_scale_factor: int = None*, *loss_scale_window: int = None*, *loss_reduction: str = 'mean'*, *calculate_per_token_loss: bool = False*, *wrap_with_ddp: bool = False*, *accumulate_allreduce_grads_in_fp32: bool = False*, *overlap_grad_reduce: bool = False*, *delay_grad_reduce: bool = False*, *use_distributed_optimizer: bool = False*, *bucket_size: Optional[int] = None*, *check_for_nan_in_grad: bool = False*, *fp16: bool = False*, *bf16: bool = False*, *resume_training: bool = False*, *crc_check: bool = False*, *load_checkpoint: str = ''*, *enable_compile_cache: bool = False*, *compile_cache_path: str = None*, *ckpt_format: str = 'ckpt'*, *prefix: str = 'network'*, *keep_checkpoint_max: int = 5*, *no_load_optim: bool = False*, *no_load_rng: bool = True*, *new_dataset: bool = False*, *enable_mem_align: bool = False*, *profile: bool = False*, *profile_save_path: str = None*, *profile_step_start: int = 1*, *profile_step_end: int = 5*, *profile_level: str = 'level0'*, *profile_with_stack: bool = False*, *profile_memory: bool = False*, *profile_framework: str = 'all'*, *profile_communication: bool = False*, *profile_parallel_strategy: bool = False*, *profile_aicore_metrics: int = 0*, *profile_l2_cache: bool = False*, *profile_hbm_ddr: bool = False*, *profile_pcie: bool = False*, *profile_data_process: bool = False*, *profile_data_simplification: bool = False*, *profile_op_time: bool = True*, *profile_offline_analyse: bool = False*, *profile_dynamic_profiler_config_path: str = ''*, ***kwargs*)

训练配置类。

参数：

- **parallel_config** (ModelParallelConfig) - 并行配置。
- **dataset_config** (DatasetConfig) - 数据配置。默认值：`DatasetConfig()` 。
- **lora_config** (LoraConfig) - Lora 配置。默认值：`LoraConfig()` 。
- **seed** (int，可选) - 初始化使用的随机种子。默认值：`None` 。

- **output_dir** (str，可选) - 用来存放 ckpt 和日志的目录。默认值：`"./output"` 。
- **training_iters** (int，可选) - 训练使用的迭代次数。默认值：`0` 。
- **epochs** (int，可选) - 训练使用的迭代周期次数。默认值：`None` 。
- **log_interval** (int，可选) - 训练日志输出间隔。默认值：`None` 。
- **eval_interval** (int，可选) - 训练评估间隔。默认值：`None` 。
- **save_interval** (int，可选) - 训练存储间隔。默认值：`None` 。
- **best_metric_comparison** (str，可选) - 比较最佳指标的方法。默认值：`None` 。
- **eval_metric** (str，可选) - 评估指标的名称。默认值：`None` 。
- **grad_clip_kwargs** (dict，可选) - 梯度裁剪参数。默认值：`None` 。
- **loss_scale** (Union[float, int]，可选) - 损失缩放的初始值。如果设置，将使用静态损失缩放器（static loss scaler）。默认值：`None` 。
- **loss_scale_value** (Union[float, int]，可选) - 动态损失缩放的初始值。默认值：`None` 。
- **loss_scale_factor** (int，可选) - 动态损失缩放因子。默认值：`None` 。
- **loss_scale_window** (int，可选) - 动态损失缩放窗口大小。默认值：`None` 。
- **loss_reduction** (str，可选) - 损失归约（Loss Reduction）方法。默认值：`"mean"` 。
- **calculate_per_token_loss** (bool，可选) - 根据令牌数量应用梯度和损失计算。默认值：`False` 。
- **wrap_with_ddp** (bool，可选) - 使用分布式数据并行封装模型。默认值：`False` 。
- **accumulate_allreduce_grads_in_fp32** (bool，可选) - 在 fp32 开启时, 是否累积 *allreduce* 梯度。默认值：`False` 。
- **overlap_grad_reduce** (bool，可选) - 使用分布式数据并行时启用梯度计算和同步通信重叠。默认值：`False` 。
- **delay_grad_reduce** (bool，可选) - 如果设置为 `True` ，则在除第一个 PP 阶段外的所有阶段中延迟梯度归约。默认值：`False` 。
- **use_distributed_optimizer** (bool，可选) - 使用分布式数据并行时使用分布式优化器。默认值：`False` 。
- **bucket_size** (Optional[int]，可选) - 当 *overlap_grad_reduce* 为 `True` 时，用于将缓冲区划分为桶的桶大小。默认值：`None` 。
- **check_for_nan_in_grad** (bool，可选) - 如果设置 `True` ，则同步后会检查缓冲区中的梯度是否是有限的（非 NaN 的）。默认值：`False` 。
- **fp16** (bool，可选) - 是否使用 `fp16` 类型。默认值：`False` 。
- **bf16** (bool，可选) - 是否使用 `bf16` 类型。默认值：`False` 。
- **resume_training** (bool，可选) - 是否开启续训。默认值：`False` 。
- **crc_check** (bool，可选) - 保存/加载 ckpt 时进行 CRC 检查，启用此选项可能会导致训练性能降低。默认值：`False` 。
- **load_checkpoint** (str，可选) - 加载 ckpt 的路径。默认值：`""` 。
- **enable_compile_cache** (bool，可选) - 保存编译缓存，启用此选项可能会导致训练性能降低。默认值：`False` 。
- **compile_cache_path** (str，可选) - 保存编译缓存的路径。默认值：`None` 。
- **ckpt_format** (str，可选) - ckpt 保存的格式。默认值：`"ckpt"` 。
- **prefix** (str，可选) - ckpt 保存的前缀。默认值：`"network"` 。

- **keep_checkpoint_max** (int，可选) - 存储 ckpt 的最大数量。默认值：`5` 。
- **no_load_optim** (bool，可选) - 恢复训练时，是否加载优化器状态。默认值：`False` 。
- **no_load_rng** (bool，可选) - 恢复训练时，是否加载 RNG 状态。默认值：`True` 。
- **new_dataset** (bool，可选) - 恢复训练时，是否使用新数据集。默认值：`False` 。
- **enable_mem_align** (bool，可选) - 是否启用内存对齐。默认值：`False` 。
- **profile** (bool，可选) - 是否打开分析。默认值：`False` 。
- **profile_save_path** (str，可选) - 保存分析文件的路径。默认值：`None` 。
- **profile_step_start** (int，可选) - 分析开始的步骤。默认值：`1` 。
- **profile_step_end** (int，可选) - 分析结束的步骤。默认值：`5` 。
- **profile_level** (str，可选) - 分析级别。默认值：`"level0"` 。
- **profile_with_stack** (bool，可选) - 使用堆栈信息进行分析。默认值：`False` 。
- **profile_memory** (bool，可选) - 使用内存信息进行分析。默认值：`False` 。
- **profile_framework** (str，可选) - 使用框架信息进行分析。默认值：`"all"` 。
- **profile_communication** (bool，可选) - 使用通信信息进行分析。默认值：`False` 。
- **profile_parallel_strategy** (bool，可选) - 使用并行策略信息进行分析。默认值：`False` 。
- **profile_aicore_metrics** (int，可选) - 使用 aicore 度量信息进行分析。默认值：`0` 。
- **profile_l2_cache** (bool，可选) - 使用二级缓存信息进行分析。默认值：`False` 。
- **profile_hbm_ddr** (bool，可选) - 使用 hbm ddr 信息进行分析。默认值：`False` 。
- **profile_pcie** (bool，可选) - 使用 pcie 信息进行分析。默认值：`False` 。
- **profile_data_process** (bool，可选) - 使用数据进程信息进行分析。默认值：`False` 。
- **profile_data_simplification** (bool，可选) - 使用数据简化进行分析。默认值：`False` 。
- **profile_op_time** (bool，可选) - 使用操作时间信息进行分析。默认值：`True` 。
- **profile_offline_analyse** (bool，可选) - 使用离线分析。默认值：`False` 。
- **profile_dynamic_profiler_config_path** (str，可选) - 动态分析器的配置路径。默认值：`""` 。
- **kwargs** (dict) - 额外的关键字配置参数。

异常：

- **ValueError** - *fp16* 是 `True` 并且 *bf16* 也是 `True` 。

支持平台：

`Ascend`

样例：

```
>>> from mindspeed_ms.core.config import TrainingConfig, ModelParallelConfig
>>> parallel_config = ModelParallelConfig()
>>> training_config = TrainingConfig(parallel_config=parallel_config)
```

1.1.7 mindspeed_ms.core.config.TransformerConfig

class mindspeed_ms.core.config.**TransformerConfig**(*vocab_size: int*, *num_layers: int*, *num_attention_heads: int*, *hidden_size: int*, *ffn_hidden_size: int*, *parallel_config:* ModelParallelConfig, *training_config:* TrainingConfig, *lora_config: LoraConfig = LoraConfig()*, *dataset_config:* DatasetConfig *= DatasetConfig()*, *moe_config:* MoEConfig *= MoEConfig()*, *attention_type: str = 'self_attn'*, *position_embedding_type: str = 'absolute'*, *parallel_position_embedding: bool = False*, *rotary_config: dict = None*, *use_query_layer: bool = False*, *use_visual_encoder: bool = False*, *use_retriever: bool = False*, *group_query_attention: bool = False*, *num_query_groups: int = 32*, *qkv_has_bias: bool = True*, *out_proj_has_bias: bool = True*, *head_skip_weight_param_allocation: bool = True*, *apply_query_key_layer_scaling: bool = False*, *use_flash_attention: bool = False*, *fa_config=None*, *enable_flash_sp: bool = False*, *mask_func_type: str = 'attn_mask_add'*, *mlp_has_bias: bool = True*, *hidden_act: str = 'gelu'*, *normalization: str = 'LayerNorm'*, *norm_epsilon: float = 1.0e-5*, *apply_residual_connection_post_norm: bool = False*, *use_final_norm: bool = True*, *residual_connection_dtype: str = 'float32'*, *init_method_std: float = 0.01*, *params_dtype: str = 'float32'*, *embedding_init_dtype: str = 'float32'*, *compute_dtype: str = 'float32'*, *softmax_compute_dtype: str = 'float32'*, *init_method: str = 'normal'*, *bias_init: str = 'zeros'*, *fp16_lm_cross_entropy: bool = False*, *attention_dropout: float = 0.0*, *out_hidden_size: int = None*, *num_experts: int = None*, *untie_embeddings_and_output_weights: bool = False*, *flatten_labels_and_input_mask: bool = True*, *recompute_method: str = None*, *recompute_num_layers: int = None*, *recompute_granularity: str = None*, *fp32_residual_connection: bool = False*, *kv_channels: int = None*, *hidden_dropout: float = 0.0*, *bias_dropout_fusion: bool = False*, *fp8_format: str = None*, *clone_scatter_output_in_embedding: bool = False*, *add_bias_linear: bool = False*, *attention_softmax_in_fp32: bool = True*, *masked_softmax_fusion: bool = False*, *distribute_saved_activations: bool = False*, *retro_add_retriever: bool = False*, *transformer_impl: str = 'local'*, *encoder_num_layers: int = None*, *decoder_num_layers: int = None*, *model_type: str = 'encoder_or_decoder'*, *select_comm_recompute: bool = False*, *select_recompute: bool = False*, *apply_rope_fusion: bool = False*, *use_sandwich_norm: bool = False*, *attn_post_norm_scale: float = 1.0*, *ffn_post_norm_scale: float = 1.0*, *apply_swiglu_fusion: bool = False*, ***kwargs*)

Transformer 配置类。

参数:

- **vocab_size** (int) - 词汇表大小。

- **num_layers** (int) - 模型层数。
- **num_attention_heads** (int) - 多头注意力机制中的头的数量。
- **hidden_size** (int) - 隐藏层大小。
- **ffn_hidden_size** (int) - 前馈网络的隐藏层大小。
- **parallel_config** (ModelParallelConfig) - 并行配置。
- **training_config** (TrainingConfig) - 训练配置。
- **lora_config** (LoraConfig，可选) - LoRA 配置。默认值：`LoraConfig()` 。
- **dataset_config** (DatasetConfig，可选) - 数据集配置。默认值：`DatasetConfig()` 。
- **moe_config** (MoEConfig，可选) - 混合专家（MoE）配置。默认值：`MoEConfig()` 。
- **attention_type** (str，可选) - 注意力类型。默认值：`"self_attn"` 。
- **position_embedding_type** (str，可选) - 位置嵌入类型。默认值：`'absolute'` 。
- **parallel_position_embedding** (bool，可选) - 在使用绝对位置嵌入时使用并行词汇嵌入层。默认值：`False` 。
- **rotary_config** (dict，可选) - 旋转位置编码配置。默认值：`None` 。
- **use_query_layer** (bool，可选) - 是否使用一个单独的查询层。默认值：`False` 。
- **use_visual_encoder** (bool，可选) - 使用视觉编码器。默认值：`False` 。
- **use_retriever** (bool，可选) - 使用检索器。默认值：`False` 。
- **group_query_attention** (bool，可选) - 启用组查询注意力（GQA）。默认值：`False` 。
- **num_query_groups** (int，可选) - 使用组查询注意力时键和值的头的数量。默认值：`32` 。
- **qkv_has_bias** (bool，可选) - 注意力模块中应用于查询、键和值的线性变换是否有偏置参数。默认值：`True` 。
- **out_proj_has_bias** (bool，可选) - 应用于核心注意力模块输出的线性变换是否有偏置参数。默认值：`True` 。
- **head_skip_weight_param_allocation** (bool，可选) - 如果为 True，头部将跳过权重分配并将词用作权重。默认值：`True` 。
- **apply_query_key_layer_scaling** (bool，可选) - 在核心注意力模块中使用查询键缩放。默认值：`False` 。
- **use_flash_attention** (bool，可选) - 启用 FlashAttention。默认值：`False` 。
- **fa_config** (dict，可选) - FlashAttention 配置。默认值：`None` 。
- **enable_flash_sp** (bool，可选) - 启用 FlashSP 操作。默认值：`False` 。
- **mask_func_type** (str，可选) - 注意力掩码计算方法。默认值：`"attn_mask_add"` 。
- **mlp_has_bias** (bool，可选) - 多层感知机（MLP）模块中的线性变换有偏置参数。默认值：`True` 。
- **hidden_act** (str，可选) - 多层感知机（MLP）模块中使用的激活函数。默认值：`"gelu"` 。
- **normalization** (str，可选) - Transformer 层模块中使用的归一化方法。默认值：`"LayerNorm"` 。
- **norm_epsilon** (float，可选) - 归一化的 epsilon 值。默认值：`1.0e-5` 。
- **apply_residual_connection_post_norm** (bool，可选) - 在归一化之后使用残差连接。默认值：`False` 。
- **use_final_norm** (bool，可选) - 在 Transformer 后使用最终归一化。默认值：`True` 。
- **residual_connection_dtype** (str，可选) - 残差连接的计算数据类型。默认值：`"float32"` 。
- **init_method_std** (float，可选) - 初始化方法的标准差数值。默认值：`0.01` 。
- **params_dtype** (str，可选) - 参数初始化的数据类型。默认值：`"float32"` 。

- **embedding_init_dtype** (str，可选) - 嵌入参数初始化的数据类型。默认值：`"float32"`。
- **compute_dtype** (str，可选) - 线性模块的计算数据类型。默认值：`"float32"`。
- **softmax_compute_dtype** (str，可选) - Softmax 层的计算数据类型。默认值：`"float32"`。
- **init_method** (str，可选) - 初始化方法。默认值：`'normal'`。
- **bias_init** (str，可选) - 偏置初始化方法。默认值：`'zeros'`。
- **fp16_lm_cross_entropy** (bool，可选) - 在计算交叉熵时使用半精度（FP16）。默认值：`False`。
- **attention_dropout** (float，可选) - 注意力模块的丢弃率。默认值：`0.0`。
- **out_hidden_size** (int，可选) - 输出隐藏层大小。默认值：`None`。
- **num_experts** (int，可选) - 专家数量。默认值：`None`。
- **untie_embeddings_and_output_weights** (bool，可选) - 如果为 False，则与头部层共享嵌入。默认值：`False`。
- **flatten_labels_and_input_mask** (bool，可选) - 扁平化标签（label）和输入掩码。默认值：`True`。
- **recompute_method** (str，可选) - 重计算方法。默认值：`None`。
- **recompute_num_layers** (int，可选) - 指定重计算的层数。默认值：`None`。
- **recompute_granularity** (str，可选) - 重计算粒度。默认值：`None`。
- **fp32_residual_connection** (bool，可选) - 启用全精度（FP32）残差连接。默认值：`False`。
- **kv_channels** (int，可选) - 键和值通道数。默认值：`None`。
- **hidden_dropout** (float，可选) - 注意力模块输出的丢弃率和多层感知机（MLP）输出的丢弃率。默认值：`0.0`。
- **bias_dropout_fusion** (bool，可选) - 启用偏置丢弃融合。默认值：`False`。
- **fp8_format** (str，可选) - 使用 FP8 格式。默认值：`None`。
- **clone_scatter_output_in_embedding** (bool，可选) - 在嵌入中启用克隆散射输出。默认值：`False`。
- **add_bias_linear** (bool，可选) - 启用带偏置的线性变换。默认值：`False`。
- **attention_softmax_in_fp32** (bool，可选) - 启用在全精度（FP32）下进行注意力 Softmax 操作。默认值：`True`。
- **masked_softmax_fusion** (bool，可选) - 启用掩码 Softmax 融合。默认值：`False`。
- **distribute_saved_activations** (bool，可选) - 启用分布式保存激活值。默认值：`False`。
- **retro_add_retriever** (bool，可选) - 启用回溯添加检索器。默认值：`False`。
- **transformer_impl** (str，可选) - Transformer 实现方式。默认值：`'local'`。
- **encoder_num_layers** (int，可选) - 编码器层数。默认值：`None`。
- **decoder_num_layers** (int，可选) - 解码器层数。默认值：`None`。
- **model_type** (str，可选) - 模型类型。默认值：`"encoder_or_decoder"`。
- **select_comm_recompute** (bool，可选) - 启用选择通信重计算。默认值：`False`。
- **select_recompute** (bool，可选) - 启用选择重计算。默认值：`False`。
- **apply_rope_fusion** (bool，可选) - 启用旋转位置编码（RoPE）融合。默认值：`False`。
- **use_sandwich_norm** (bool，可选) - 启用 Sandwich 归一化。默认值：`False`。
- **attn_post_norm_scale** (float，可选) - 注意力后归一化缩放比例。默认值：`1.0`。

- **ffn_post_norm_scale** (float，可选) - 前馈网络（FFN）后归一化缩放比例。默认值：`1.0` 。
- **apply_swiglu_fusion** (bool，可选) - 启用 SwiGLU 融合。默认值：`False` 。
- **kwargs** (dict，可选) - 额外的关键字配置参数。

1.1.8 mindspeed_ms.core.parallel_state.get_context_parallel_rank

mindspeed_ms.core.parallel_state.**get_context_parallel_rank**()

返回上下文并行组中的 rank。

返回：

- rank 号。

支持平台：

`Ascend`

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> from mindspore.communication.management import init
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> from mindspeed_ms.core.parallel_state import get_context_parallel_rank
>>> init()
>>> initialize_model_parallel(context_parallel_size=1)
>>> cp_rank = get_context_parallel_rank()
>>> print(cp_rank)
0
```

1.1.9 mindspeed_ms.core.parallel_state.get_context_parallel_world_size

mindspeed_ms.core.parallel_state.**get_context_parallel_world_size**()

返回上下文并行组的 world 大小。

返回：

- world 的大小。

支持平台：

`Ascend`

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> from mindspore.communication.management import init
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> from mindspeed_ms.core.parallel_state import get_context_parallel_world_size
>>> init()
>>> initialize_model_parallel(context_parallel_size=1)
>>> cp_world_size = get_context_parallel_world_size()
>>> print(cp_world_size)
1
```

1.1.10 mindspeed_ms.core.parallel_state.initialize_model_parallel

mindspeed_ms.core.parallel_state.**initialize_model_parallel**(*tensor_model_parallel_size=1*, *pipeline_model_parallel_size=1*, *virtual_pipeline_model_parallel_size=None*, *pipeline_model_parallel_split_rank=None*, *context_parallel_size=1*, *expert_model_parallel_size=1*, *order='tp-cp-ep-dp-pp'*, *communicator_config_path=None*, ***kwargs*)

初始化模型数据并行组。

参数：

- **tensor_model_parallel_size** (int，可选) - 将单个张量分割到多少个设备（如 NPU 等）上。默认值：`1` 。
- **pipeline_model_parallel_size** (int，可选) - 将 Transformer 层分割到多少个张量并行设备组中。默认值：`1` 。
- **virtual_pipeline_model_parallel_size** (int，可选) - 每个流水线组将拥有多少个阶段，必要时进行交错。如果为 `None` ，则不进行交错。默认值：`None` 。
- **pipeline_model_parallel_split_rank** (int，可选) - 对于具有编码器和解码器的模型，在流水线中切换编码器和解码器的秩（即解码器的第一个秩）（秩，英文为 rank，可以简单理解为在分布式训练中设备或者处理单元的编号、顺序之类的标识）。这允许用户独立设置编码器和解码器的流水线并行大小。默认值：`None` 。
- **context_parallel_size** (int，可选) - 将网络输入序列长度分割到多少个张量并行设备组中。计算注意力模块需要完整的序列长度的 token，因此上下文并行组中的设备需要相互通信以交换其他序列块的信息。每个设备及其在其他张量并行组中的对应物组成一个上下文并行组。默认值：`1` 。
- **expert_model_parallel_size** (int，可选) - 在 MoE 模型中，将专家分割到多少个设备上。默认值：`1` 。
- **order** (str，可选) - 每个并行策略遵循的顺序。默认值：`"tp-cp-ep-dp-pp"` 。
- **communicator_config_path** (str，可选) - HCCL 通信器配置的 yaml 文件路径。目前未使用。默认值：`None` 。
- **kwargs** (dict) - 额外的关键字配置参数。

异常：

- **RuntimeError** - *mindspore.communication._comm_helper* 没有被初始化。
- **RuntimeError** - *world_size* 不能被 *tensor_model_parallel_size* * *pipeline_model_parallel_size* * *context_parallel_size* 整除。
- **RuntimeError** - *data_parallel_size* 不能被 *expert_model_parallel_size* 整除。
- **RuntimeError** - *expert_model_parallel_size* > *1* 且 *context_parallel_size* > *1*。
- **RuntimeError** - *virtual_pipeline_model_parallel_size* 不是 `None` 且 *pipeline_model_parallel_size* < *2*。
- **RuntimeError** - *order* 是 `None` 。

- **RuntimeError** - *order* 中有重复元素。
- **RuntimeError** - *ep* 在 *order* 中，*ep-dp* 不在 *order* 中且 *dp-ep* 也不在 *order* 中。
- **RuntimeError** - *_GLOBAL_STREAM* 已经被初始化。

支持平台：

`Ascend`

样例：

说明：

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> from mindspore.communication.management import init
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> from mindspeed_ms.core.parallel_state import get_context_parallel_world_size
>>> init()
>>> initialize_model_parallel(context_parallel_size=1)
>>> context_parallel_world_size = get_context_parallel_world_size()
>>> print(context_parallel_world_size)
1
```

1.1.11 mindspeed_ms.core.tensor_parallel.ColumnParallelLinear

class mindspeed_ms.core.tensor_parallel.**ColumnParallelLinear**(*input_size*, *output_size*, *, *config*, *init_method*, *bias=True*, *gather_output=False*, *stride=1*, *keep_master_weight_for_test=False*, *skip_bias_add=False*, *skip_weight_param_allocation=False*, *embedding_activation_buffer=None*, *grad_output_buffer=None*, *is_expert=False*, *tp_comm_buffer_name=None*, *disable_grad_reduce=False*, *bias_init=Zero()*, *param_init_dtype=None*, *compute_dtype=None*, *transpose_b=True*)

稠密线性层计算，在权重的第二个维度按照张量并行大小进行切分，并行计算。

$$\text{outputs} = \text{inputs} * \text{weight} + \text{bias}$$

其中，*inputs* 代表输入张量，weight 代表该线性层的权重矩阵，bias 代表该线性层的偏置向量（当且仅当 *bias* 为 `True` 时会参与计算）。

参数：

- **input_size** (int) - 输入空间的通道数。
- **output_size** (int) - 输出空间的通道数。

关键字参数：

- **config** (dict) - Transformer 模型的配置，详情请参考 TransformerConfig 类。
- **init_method** (Union[Tensor, str, Initializer, numbers.Number]) - 可训练参数的初始化方式。若传入值类型为字符串，则对应 *initializer* 的函数名。
- **bias** (bool，可选) - 指定该层是否使用偏置向量。默认值：`True` 。
- **gather_output** (bool，可选) - 指定是否要在每个张量并行进程计算完成后进行聚合。默认值：`False` 。
- **stride** (int，可选) - 描述线性层计算的步长。默认值：`1` 。
- **keep_master_weight_for_test** (bool，可选) - 用于测试，正常使用时应当设置为 `False` 。该参数会返回用于初始化的主权重。默认值：`False` 。
- **skip_bias_add** (bool，可选) - 如果为 `True` ，则计算式不会加上偏置，而是会将偏置返回用于后续的融合计算。默认值：`False` 。
- **skip_weight_param_allocation** (bool，可选) - 指定是否跳过权重参数初始化。当设置为 `True` 时，需往前向接口传入一个权重张量。默认值：`False` 。
- **embedding_activation_buffer** (Tensor，可选) - 该缓冲区存放了最后一个流水线阶段的最后一个线性嵌入层的输入激活值。默认值：`None` 。
- **grad_output_buffer** (Tensor，可选) - 该缓冲区存放了最后一个流水线阶段的最后一个线性嵌入层的输出梯度。默认值：`None` 。
- **is_expert** (bool，可选) - 指定该线性层是否为专家。默认值：`False` 。
- **tp_comm_buffer_name** (str，可选) - 通信缓冲区在非 Transformer 引擎模块中不会被使用。默认值：`None` 。
- **disable_grad_reduce** (bool，可选) - 如果设置为 `True` ，将不会使能跨张量并行进程的输出梯度聚合。默认值：`False` 。
- **bias_init** (Union[Tensor, str, Initializer, numbers.Number]，可选) - 可训练的偏置参数初始化方法。若传入值类型为字符串，则对应 *initializer* 的函数名。默认值：`Zero()` 。
- **param_init_dtype** (dtype.Number，可选) - 参数初始化类型。默认值：`None` 。
- **compute_dtype** (dtype.Number，可选) - 计算类型。默认值：`None` 。
- **transpose_b** (bool，可选) - 指定是否将权重参数初始化为转置矩阵。默认值：`True` 。

输入：

- **input_** (Tensor) - 形状为 $(*, in\_channels)$ 的输入张量。接口参数中的 *input_size* 需与 *in_channels* 一致。
- **weight** (Tensor) - 形状为 $(in\_channels, out\_channels)$ 的张量。接口参数中的 *input_size* 需与 *in_channels* 一致。接口参数中的 *output_size* 需与 *out_channels* 一致。

输出：

- **output** (Tensor) - 形状为 $(*, out\_channels)$ 的张量。接口参数中的 *output_size* 需与 *out_channels* 一致。
- **output_bias** (Tensor) - Tensor of shape $(out\_channels,)$ 。

异常：

- **ValueError** - *skip_weight_param_allocation=True* 但是权重张量并未传给前向函数。
- **NotImplementedError** - 当前不支持 *stride > 1* 。
- **NotImplementedError** - 当前不支持 *keep_master_weight_for_test=True* 。
- **NotImplementedError** - 当前不支持 *embedding_activation_buffer* 。
- **NotImplementedError** - 当前不支持 *grad_output_buffer* 。

- **NotImplementedError** - 当前不支持 *tp_comm_buffer_name* 。
- **NotImplementedError** - 当前不支持 *disable_grad_reduce=True* 。
- **NotImplementedError** - 当前不支持 *config.parallel_config.use_cpu_initialization* 。
- **RuntimeError** - 使用了 *zero3* 优化器并行，但是未初始化数据并行通信。
- **RuntimeError** - *allreduce_dgrad* 和 *sequence_parallel* 不能同时使能。

支持平台：

```
Ascend
```

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import nn, Tensor
>>> from mindspore.communication.management import init
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> from mindspeed_ms.core.config import (
...     ModelParallelConfig,
...     TrainingConfig,
...     TransformerConfig
... )
>>> from mindspeed_ms.core.tensor_parallel import ColumnParallelLinear
>>> class TestNet(nn.Cell):
...     def __init__(self, config):
...         super(TestNet, self).__init__()
...         hidden_size = config.hidden_size
...         self.columnlinear = ColumnParallelLinear(
...             input_size=hidden_size,
...             output_size=hidden_size,
...             config=config,
...             init_method=config.init_method,
...             bias=config.mlp_has_bias,
...             gather_output=False,
...             skip_bias_add=False,
...             bias_init=config.bias_init
...         )
...     def construct(self, input_):
...         output, _ = self.columnlinear(input_)
...         return output
>>> dataset_size = 1
>>> seq_length = 2
>>> hidden_size = 4
>>> tensor_parallel = 1
>>> ms.set_context(device_target='Ascend', mode=ms.PYNATIVE_MODE)
>>> ms.set_seed(2024)
>>> init()
>>> initialize_model_parallel(tensor_model_parallel_size=tensor_parallel)
>>> input_shape = (dataset_size, seq_length, hidden_size)
```

(下页继续)

(续上页)

```
>>> input_data = Tensor(np.random.random(input_shape).astype(np.float32))
>>> parallel_config = ModelParallelConfig()
>>> train_config = TrainingConfig(parallel_config=parallel_config)
>>> model_config = TransformerConfig(vocab_size=40000,
...                                  num_layers=1,
...                                  num_attention_heads=1,
...                                  hidden_size=hidden_size,
...                                  ffn_hidden_size=4*hidden_size,
...                                  parallel_config=parallel_config,
...                                  training_config=train_config,
...                                  mlp_has_bias=True,
...                                  gated_linear_unit=False,
...                                  hidden_act='gelu',
...                                  params_dtype='float32',
...                                  compute_dtype='float32')
>>> network = TestNet(config=model_config)
>>> output = network(input_data)
>>> print(output)
>>> print(output.shape)
[[[ 0.01780816  0.00895902 -0.00554341 -0.00185049]]
[[ 0.02319741 -0.00320548 -0.0062025  -0.0050142 ]]]
(1, 2, 4)
```

1.1.12 mindspeed_ms.core.tensor_parallel.RowParallelLinear

class mindspeed_ms.core.tensor_parallel.**RowParallelLinear**(*input_size*, *output_size*, *, *config*, *init_method*, *bias*, *input_is_parallel*, *skip_bias_add=True*, *stride=1*, *keep_master_weight_for_test=False*, *is_expert=False*, *tp_comm_buffer_name=None*, *bias_init=Zero()*, *param_init_dtype=None*, *compute_dtype=None*, *transpose_b=True*)

稠密线性层计算，在权重的第一个维度按照张量并行大小进行切分，并行计算。该层实现的计算公式为：

$$\text{outputs} = \text{inputs} * \text{weight} + \text{bias}$$

其中，*inputs* 代表输入张量，weight 代表该线性层的权重矩阵，bias 代表该线性层的偏置向量（当且仅当 *has_bias* 为 `True` 时会参与计算）。

参数：

- **input_size** (int) - 输入空间的通道数。
- **output_size** (int) - 输出空间的通道数。

关键字参数：

- **config** (dict) - Transformer 模型的配置，详情请参考 TransformerConfig 类。
- **init_method** (Union[Tensor, str, Initializer, numbers.Number]) - 可训练参数的初始化方式。若传入值类型为字符串，则对应 *initializer* 的函数名。
- **bias** (bool) - 指定该层是否使用偏置向量。
- **input_is_parallel** (bool) - 指定输入张量是否已经按照张量并行策略进行了切分，若是，那么我们就不用再进行切分了。

- **skip_bias_add** (bool，可选) - 如果为 `True` ，则计算式不会加上偏置，而是会将偏置返回用于后续的融合计算。默认值：`False` 。
- **stride** (int，可选) - 描述线性层计算的步长。默认值：`1` 。
- **keep_master_weight_for_test** (bool，可选) - 用于测试，正常使用时应当设置为 `False` 。该参数会返回用于初始化的主权重。默认值：`False` 。
- **is_expert** (bool，可选) - 指定该线性层是否为专家。默认值：`False` 。
- **tp_comm_buffer_name** (str，可选) - 通信缓冲区在非 Transformer 引擎模块中不会被使用。默认值：`None` 。
- **bias_init** (Union[Tensor, str, Initializer, numbers.Number]，可选) - 可训练的偏置参数初始化方法。若传入值类型为字符串，则对应 *initializer* 的函数名。默认值：`Zero()` 。
- **param_init_dtype** (dtype.Number，可选) - 参数初始化类型。默认值：`None` 。
- **compute_dtype** (dtype.Number，可选) - 计算类型。默认值：`None` 。
- **transpose_b** (bool，可选) - 指定是否将权重参数初始化为转置矩阵。默认值：`True` 。

输入：

- **input_** (Tensor) - 形状为 $(*, in\_channels)$ 的输入张量。接口参数中的 *input_size* 需与 *in_channels* 一致。

输出：

- **output** (Tensor) - 形状为 $(*, out\_channels)$ 的张量。接口参数中的 *output_size* 需与 *out_channels* 一致。
- **output_bias** (Tensor) - 形状为 $(out\_channels,)$ 的张量。

异常：

- **ValueError** - 当 *input_is_parallel* 为 `False` 时，*sequence_parallel* 应当为 `False` ，但是被设置为了 `True` 。
- **ValueError** - 当 *explicit_expert_comm* 为 `True` 时，*skip_bias_add* 应当为 `True` ，但是被设置为了 `False` 。
- **NotImplementedError** - 当前不支持 *stride > 1* 。
- **NotImplementedError** - 当前不支持 *keep_master_weight_for_test=True* 。
- **NotImplementedError** - 当前不支持 *tp_comm_buffer_name* 。
- **RuntimeError** - 使用了 *zero3* 优化器并行，但是未初始化数据并行通信。
- **RuntimeError** - 为了使能 *sequence_arallel* ， *input_is_parallel* 必须为 `True` 但是得到了 `False` 。

支持平台：

`Ascend`

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import nn, Tensor
>>> from mindspore.communication.management import init
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> from mindspeed_ms.core.config import (
```

(下页继续)

(续上页)

```
...     ModelParallelConfig,
...     TrainingConfig,
...     TransformerConfig
... )
>>> from mindspeed_ms.core.tensor_parallel import RowParallelLinear
>>> class TestNet(nn.Cell):
...     def __init__(self, config):
...         super(TestNet, self).__init__()
...         hidden_size = config.hidden_size
...         self.rowlinear = RowParallelLinear(input_size=hidden_size,
...                                            output_size=hidden_size,
...                                            config=config,
...                                            init_method=config.init_method,
...                                            bias=config.mlp_has_bias,
...                                            input_is_parallel=True,
...                                            skip_bias_add=False,
...                                            bias_init=config.bias_init)
...     def construct(self, input_):
...         output, _ = self.rowlinear(input_)
...         return output
>>> seq_length = 2
>>> dataset_size = 1
>>> hidden_size = 4
>>> tensor_parallel = 1
>>> ms.set_context(device_target='Ascend', mode=ms.PYNATIVE_MODE)
>>> ms.set_seed(2024)
>>> init()
>>> initialize_model_parallel(tensor_model_parallel_size=tensor_parallel)
>>> shape = (seq_length, dataset_size, hidden_size)
>>> input_data = Tensor(np.random.random(shape).astype(np.float32))
>>> parallel_config = ModelParallelConfig()
>>> training_config = TrainingConfig(parallel_config=parallel_config)
>>> model_config = TransformerConfig(vocab_size=40000,
...                                  num_layers=1,
...                                  num_attention_heads=1,
...                                  hidden_size=hidden_size,
...                                  ffn_hidden_size=4*hidden_size,
...                                  parallel_config=parallel_config,
...                                  training_config=training_config,
...                                  mlp_has_bias=True,
...                                  gated_linear_unit=False,
...                                  hidden_act='gelu',
...                                  params_dtype='float32',
...                                  compute_dtype='float32')
>>> network = TestNet(config=model_config)
>>> output = network(input_data)
>>> print(output)
>>> print(output.shape)
[[[ 0.01780816  0.00895902 -0.00554341 -0.00185049]]
[[ 0.02319741 -0.00320548 -0.0062025  -0.0050142 ]]]
(2, 1, 4)
```

1.1.13 mindspeed_ms.core.tensor_parallel.VocabParallelCrossEntropy

class mindspeed_ms.core.tensor_parallel.**VocabParallelCrossEntropy**(**args*, ***kwargs*)

交叉熵损失函数的并行接口。

参数：

- **args** (tuple) - 位置参数。
- **kwargs** (dict) - 其他输入。

输入：

- **vocab_parallel_logits** (Tensor) - 主干网络的输出。形状为 (N, C) 的张量。数据类型必须为 float16 或 float32。
- **target** (Tensor) - 样本的真值。形状为 $(N,)$ 。
- **label_smoothing** (float，可选) - 平滑因子，必须在范围 `[0.0, 1.0)` 内。默认值：`0.0` 。

输出：

- **loss** (Tensor) - 对应的交叉熵损失。

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor
>>> from mindspore.communication.management import init
>>> from mindspeed_ms.core.tensor_parallel.cross_entropy import          ...     ␣
↪VocabParallelCrossEntropy
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> init()
>>> initialize_model_parallel()
>>> loss = VocabParallelCrossEntropy()
>>> logits = Tensor([[2., 1., 0.2],[2., 1., 0.2]], mstype.float32)
>>> labels = Tensor([1,1], mstype.int32)
>>> output = loss(logits, labels)
>>> print(output.shape)
(2,)
>>> print(output)
[1.4273429 1.4273429]
```

1.1.14 mindspeed_ms.core.tensor_parallel.VocabParallelEmbedding

class mindspeed_ms.core.tensor_parallel.**VocabParallelEmbedding**(*num_embeddings*, *embedding_dim*, ***, *init_method*, *reduce_scatter_embeddings=False*, *config*, *param_init_dtype=None*)

在词汇维度并行的嵌入计算。

参数：

- **num_embeddings** (int) - 词表大小。

- **embedding_dim** (int) - 隐含层大小。

关键字参数：

- **init_method** (Union[Tensor, str, Initializer, numbers.Number]) - 可训练参数的初始化方式。若传入值类型为字符串，则对应 *initializer* 的函数名。
- **reduce_scatter_embeddings** (bool，可选) - 指定在嵌入查询后是否要执行 ReduceScatter 操作。默认值：`False`。
- **config** (dict) - Transformer 模型的配置，详情请参考 TransformerConfig 类。
- **param_init_dtype** (dtype.Number，可选) - 参数初始化类型。默认值：`None`。

输入：

- **input_** (Tensor) - 形状为 (B, S) 或 (S, B) 的输入张量。

输出：

- **output** (Tensor) - 形状为 (B, S, H) 或 (S, B, H) 的张量，与输入张量对应。

异常：

- **ValueError** - 词表大小无法被张量并行数整除。
- **NotImplementedError** - 当前不支持 *config.parallel_config.deterministic_mode*。
- **NotImplementedError** - 当前不支持 *config.parallel_config.use_cpu_initialization*。

支持平台：

`Ascend`

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import nn, ops, Tensor
>>> from mindspore.communication.management import init
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> from mindspeed_ms.core.tensor_parallel.layers import VocabParallelEmbedding
>>> from mindspeed_ms.core.config import (
...     ModelParallelConfig,
...     TrainingConfig,
...     DatasetConfig,
...     TransformerConfig
... )
>>> ms.set_context(device_target="Ascend", mode=ms.PYNATIVE_MODE, deterministic='ON')
>>> ms.set_context(pynative_synchronize=True)
>>> ms.set_seed(2024)
>>> ms.set_auto_parallel_context(parallel_mode=ms.ParallelMode.DATA_PARALLEL)
>>> class ParallelTransformerLayerNet(nn.Cell):
...     def __init__(self, config):
...         super(ParallelTransformerLayerNet, self).__init__()
...         self.config = config
```

(下页继续)

(续上页)

```
...         self.embedding = VocabParallelEmbedding(
...             num_embeddings=config.vocab_size,
...             embedding_dim=config.hidden_size,
...             init_method=config.init_method,
...             reduce_scatter_embeddings=config.parallel_config.sequence_parallel,
...             config=config,
...         )
...     def construct(self, x):
...         x = self.embedding(x)
...         return x
>>> parallel_config = ModelParallelConfig(tensor_model_parallel_size=2,
...                                       pipeline_model_parallel_size=1,
...                                       context_parallel_size=1,
...                                       expert_model_parallel_size=1,
...                                       sequence_parallel=True)
>>> training_config = TrainingConfig(parallel_config=parallel_config)
>>> dataset_config = DatasetConfig(batch_size=1,
...                                dataset_size=2,
...                                seq_length=1024)
>>> model_config = TransformerConfig(vocab_size=50304,
...                                  num_layers=1,
...                                  num_attention_heads=32,
...                                  hidden_size=2560,
...                                  ffn_hidden_size=7680,
...                                  parallel_config=parallel_config,
...                                  training_config=training_config)
>>> model_config.dataset_config = dataset_config
>>> batch_size = dataset_config.batch_size
>>> dataset_size = dataset_config.dataset_size
>>> seq_length = dataset_config.seq_length
>>> vocab_size = model_config.vocab_size
>>> init()
>>> tensor_parallel = parallel_config.tensor_model_parallel_size
>>> initialize_model_parallel(tensor_model_parallel_size=tensor_parallel)
>>> network = ParallelTransformerLayerNet(config=model_config)
>>> input_shape = (batch_size, seq_length)
>>> input_ids = Tensor(np.ones(input_shape).astype(np.int32))
>>> output = network(input_ids)
>>> print(output)
[[[-0.00546161  0.00440422  0.00252223 ...  0.00539334 -0.00625365 -0.01025379]
[-0.00546161  0.00440422  0.00252223 ...  0.00539334 -0.00625365 -0.01025379]
[-0.00546161  0.00440422  0.00252223 ...  0.00539334 -0.00625365 -0.01025379]
...
[-0.00546161  0.00440422  0.00252223 ...  0.00539334 -0.00625365 -0.01025379]
[-0.00546161  0.00440422  0.00252223 ...  0.00539334 -0.00625365 -0.01025379]
[-0.00546161  0.00440422  0.00252223 ...  0.00539334 -0.00625365 -0.01025379]]]
```

1.1.15 mindspeed_ms.legacy.model.eos_mask.EosMask

class mindspeed_ms.legacy.model.eos_mask.**EosMask**(*batch_size*, *seq_len*, *eod_token_id*, *reset_position_ids*)

生成特定 token 对应的注意力掩码。

参数：

- **batch_size** (int) - 批大小。
- **seq_len** (int) - 序列长度。
- **eod_token_id** (int) - 文档结束标记的标识符。
- **reset_position_ids** (bool) - 如果为 `True` ， 位置索引将被重置。

输入：

- **input_ids** (Tensor) - 输入索引。形状为 (B, S) 的张量。

输出：

- **position_ids** (Tensor) - 位置索引。形状为 (B, S) 的张量。
- **mint.sub(1, mask)** (Tensor) - 掩码。形状为 (B, S, S) 的张量。

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> from mindspore.communication.management import init
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, nn
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> from mindspeed_ms.legacy.model.eos_mask import EosMask
>>> ms.set_context(device_target='Ascend', mode=ms.PYNATIVE_MODE)
>>> ms.set_seed(2024)
>>> init()
>>> initialize_model_parallel()
>>> b = 2
>>> s = 4
>>> eod_token_id = 4
>>> loss = EosMask(b, s, eod_token_id, reset_position_ids=False)
>>> input_ids = ms.Tensor(np.random.random((b, s)).astype(np.float32))
>>> output, mask = loss(input_ids)
>>> print(output.shape)
>>> print(mask.shape)
(2, 4)
(2, 4, 4)
```

1.1.16 mindspeed_ms.legacy.model.gpt_model.GPTModel

class mindspeed_ms.legacy.model.gpt_model.**GPTModel**(*config*, *num_tokentypes=0*, *parallel_output=True*, *pre_process=True*, *post_process=True*, ***kwargs*)

生成式预训练 Transformer（GPT）的实现，是一个仅有解码器的 Transformer 模型。

参数：

- **config** (TransformerConfig) - Transformer 模型配置，包括初始化函数和并行参数配置等。
- **num_tokentypes** (int，可选) - 如果大于 0，则使用 tokentype 嵌入。默认值：0.
- **parallel_output** (bool，可选) - 指定是否返回各张量并行权重上的并行输出。默认值：`True` 。
- **pre_process** (bool，可选) - 使用流水线并行时，标记它是否为第一阶段。默认值：`True` 。
- **post_process** (bool，可选) - 使用流水线并行时，标记它是否为最后的阶段。默认值：`True` 。
- **kwargs** (dict) - 其他输入。

输入：

- **tokens** (tuple[Tensor]) - 输入索引。形状为 (B, S) 。
- **position_ids** (tuple[Tensor]) - 位置偏移量。形状为 (B, S) 。
- **attention_mask** (tuple[Tensor]) - 注意力掩码。形状为 (B, S) 。
- **loss_mask** (tuple[Tensor]) - 损失掩码。形状为 (B, S) 。
- **retriever_input_ids** (tuple[Tensor]，可选) - 检索器输入标记索引。默认值：`None` 。
- **retriever_position_ids** (tuple[Tensor]，可选) - 检索器输入位置索引。默认值：`None` 。
- **labels** (tuple[Tensor]，可选) - 样本的基准真实值。形状为 $(N,)$ 。默认值：`None` 。
- **tokentype_ids** (tuple[Tensor]，可选) - 给模型输入的标记类型索引列表。形状为 (B, S) 。默认值：`None` 。
- **inference_params** (tuple[Tensor]，可选) - 推理参数，用于在推理过程中指定特定设置，如最大生成长度、最大批处理大小等。默认值：`None` 。

输出：

- 返回 GPT 模型的 loss 或 hidden states。

支持平台：

`Ascend`

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import os
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> from mindspore.communication.management import init
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> from mindspeed_ms.core.config import (
```

(下页继续)

(续上页)

```
...     ModelParallelConfig,
...     TrainingConfig,
...     DatasetConfig,
...     TransformerConfig
... )
>>> from mindspeed_ms.legacy.model.gpt_model import GPTModel
>>> ms.set_context(device_target='Ascend', mode=ms.PYNATIVE_MODE)
>>> init()
>>> initialize_model_parallel()
>>> os.environ['HCCL_BUFFSIZE'] = "200"
>>> batch_size = 8
>>> seq_length = 32
>>> parallel_config = ModelParallelConfig()
>>> data_config = DatasetConfig(batch_size=batch_size)
>>> training_config = TrainingConfig(parallel_config=parallel_config)
>>> config = TransformerConfig(vocab_size=128,
...                            seq_length=seq_length,
...                            num_layers=4,
...                            num_attention_heads=4,
...                            num_query_groups=32,
...                            hidden_size=64,
...                            ffn_hidden_size=256,
...                            parallel_config=parallel_config,
...                            training_config=training_config,
...                            dataset_config=data_config)
>>> gpt_model = GPTModel(config)
>>> input_data = Tensor(np.random.random((batch_size, seq_length)).astype(np.int32))
>>> attention_mask = Tensor(np.zeros((batch_size, 1, seq_length, seq_length)).astype(np.
↪int32))
>>> loss_mask = Tensor(np.random.random((batch_size, seq_length)).astype(np.int32))
>>> lm_output = gpt_model(tokens=input_data,
...                       position_ids=None,
...                       attention_mask=attention_mask,
...                       loss_mask=loss_mask)
>>> print(lm_output.shape)
(32, 8, 128)
```

1.1.17 mindspeed_ms.legacy.model.language_model.Embedding

class mindspeed_ms.legacy.model.language_model.**Embedding**(*hidden_size*, *vocab_size*, *max_sequence_length*, *embedding_dropout_prob*, *config*, *num_tokentypes=0*, ***kwargs*)

embedding 层包含 word embedding、position embedding 和 tokentypes embedding。

参数：

- **hidden_size** (int) - embedding 层的隐藏状态大小。
- **vocab_size** (int) - 词汇表大小。
- **max_sequence_length** (int) - 序列的最大长度，用于 position embedding。如果使用了 position embedding，必须设置最大序列长度。
- **embedding_dropout_prob** (float) - embedding 层的 dropout rate。
- **config** (TransformerConfig) - Transformer 模型的配置，详情请参考 TransformerConfig 类。

- **num_tokentypes** (int, 可选) - token-type embeddings 的数量. 如果大于 0，则使用 tokentype 嵌入。默认值：0 。
- **kwargs** (dict) - 其他输入。

输入：

- **input_ids** (Tensor) - int32 类型的的输入索引, 形状为 (B, S) 。
- **position_ids** (Tensor) - 用于 position embedding 的位置索引, 形状为 (B, S) 。
- **tokentype_ids** (Tensor) - 用于区分不同类型标记（例如，在 BERT 中区分句子 A 和句子 B）的标记类型，形状为 (B, S) 。

输出：

- **embeddings** (Tensor)- embedding 后的输出, 形状为 (B, S, H) 。

异常：

- **NotImplementedError** - 如果 *config.clone_scatter_output_in_embedding* 为 `True` 。
- **RuntimeError** - 如果 *tokentype_ids* 不为 `None` 并且 *tokentype_embeddings* 为 `None` 。如果 *tokentype_ids* 为 `None` 并且 *tokentype_embeddings* 不为 `None` 。

支持平台：

`Ascend`

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore.communication import init
>>> from mindspeed_ms.core.config import TransformerConfig, ModelParallelConfig
>>> from mindspeed_ms.legacy.model.language_model import Embedding
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> ms.set_context(device_target="Ascend", mode=ms.PYNATIVE_MODE, deterministic='ON')
>>> init()
>>> initialize_model_parallel()
>>> parallel_config = ModelParallelConfig()
>>> config = TransformerConfig(vocab_size=128,
...                            num_layers=1,
...                            num_attention_heads=8,
...                            num_query_groups=4,
...                            hidden_size=256,
...                            ffn_hidden_size=128,
...                            parallel_config=parallel_config,
...                            training_config=None)
>>> embedding = Embedding(hidden_size=config.hidden_size,
...                       vocab_size=128,
...                       max_sequence_length=64,
...                       embedding_dropout_prob=0.0,
...                       config=config)
>>> shape = (2, 64)
>>> input_ids = ms.Tensor(np.random.randint(0, 100, size=shape), dtype=ms.int32)
```

(下页继续)

(续上页)

```
>>> position_array = np.expand_dims(np.arange(64), axis=0)
>>> position_array = position_array.repeat(repeats=2, axis=0)
>>> position_ids = ms.Tensor(position_array, dtype=ms.int32)
>>> out = embedding(input_ids, position_ids)
>>> print(out.shape)
(2, 64, 256)
```

1.1.18 mindspeed_ms.legacy.model.language_model.get_language_model

mindspeed_ms.legacy.model.language_model.**get_language_model**(*config*, *num_tokentypes*, *add_pooler*, *encoder_attn_mask_type*, *add_encoder=True*, *add_decoder=False*, *decoder_attn_mask_type=None*, *pre_process=True*, *post_process=True*)

使用此函数来获取语言模型。

参数：

- **config** (TransformerConfig) - Transformer 模型配置，包括初始化函数和并行参数配置等。
- **num_tokentypes** (int) - 如果大于 0，则使用 tokentypes 嵌入。
- **add_pooler** (bool) - 如果为 `True` ，使用池化层。
- **encoder_attn_mask_type** (int) - 编码器注意力掩码类型。
- **add_encoder** (bool，可选) - 如果为 `True` ，使用编码器。默认值：`True` 。
- **add_decoder** (bool，可选) - 如果为 `True` ，使用解码器。默认值：`False` 。
- **decoder_attn_mask_type** (int，可选) - 解码器注意力掩码类型。默认值：`None` 。
- **pre_process** (bool，可选) - 使用流水线并行时，标记它是否为第一阶段。默认值：`True` 。
- **post_process** (bool，可选) - 使用流水线并行时，标记它是否为最后的阶段。默认值：`True` 。

返回：

- **language_model** (TransformerLanguageModel) - Transformer 模型。
- **language_model_key** (str) - 模型的键。

支持平台：

`Ascend`

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import os
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> from mindspore.communication import init
```

(下页继续)

(续上页)

```
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> from mindspeed_ms.core.config import (
...     init_configs_from_yaml,
...     TrainingConfig,
...     ModelParallelConfig,
...     TransformerConfig,
...     DatasetConfig,
... )
>>> from mindspeed_ms.legacy.model.language_model import get_language_model
>>> os.environ['HCCL_BUFFSIZE'] = "1"
>>> config_path = "test_language_model.yaml"
>>> training_config, parallel_config, dataset_config, model_config = init_configs_from_yaml(
...     config_path, [TrainingConfig, ModelParallelConfig, DatasetConfig, TransformerConfig]
... )
>>> ms.set_context(device_target="Ascend", mode=ms.PYNATIVE_MODE,
...     deterministic="ON")
>>> init()
>>> initialize_model_parallel()
>>> batch_size = dataset_config.batch_size
>>> sq = model_config.seq_length
>>> language_model, _ = get_language_model(model_config,
...                                        num_tokentypes=0,
...                                        add_pooler=False,
...                                        encoder_attn_mask_type=None)
>>> input_data = Tensor(np.random.random((batch_size, sq)).astype(np.int32))
>>> attention_mask = Tensor(np.zeros((batch_size, 1, sq, sq)).astype(np.int32))
>>> hidden_states = language_model(input_data, None, attention_mask)
```

1.1.19 mindspeed_ms.legacy.model.module.Module

class mindspeed_ms.legacy.model.module.**Module**(*config=None*, *share_embeddings_and_output_weights=True*, ***kwargs*)

具有流水线支持的特定扩展单元。

参数：

- **config** (dict，可选) - 模型的配置。如果它不是 `None` ，*self.pre_process* 和 *self.post_process* 将根据流水线阶段进行设置。默认值：`None` 。
- **share_embeddings_and_output_weights** (bool，可选) - 决定是否共享嵌入层和输出层权重。如果不是 `True` ，将无法调用 *shared_embedding_or_output_weight()* 和 *initialize_word_embeddings()* 。默认值：`True` 。
- **kwargs** (dict) - 额外的关键字配置参数。

异常：

- **RuntimeError** - 在流水线的一个阶段中设置了多个权重的 *'share'* 属性。
- **RuntimeError** - 模型中有一个权重具有 *'share'* 属性，但参数共享要求在第一阶段和最后阶段分别有两个权重具有 *'share'* 属性。
- **RuntimeError** - 在调用 *shared_embedding_or_output_weight()* 时 *share_embeddings_and_output_weights* 不为 `True` 。
- **RuntimeError** - 在调用 *initialize_word_embeddings()* 时 *share_embeddings_and_output_weights* 不为 `True` 。
- **ValueError** - 如果是最后一阶段（后处理），但是权重之和不为 `0.0` 。

支持平台：

Ascend

样例：

```
>>> from mindspeed_ms.legacy.model.module import Module
>>> class CellExample(Module):
...     def __init__(self, layers):
...         super(CellExample, self).__init__()
...         self.layers = layers
...     def construct(self, hidden_states, *args, **kwargs):
...         for layer in self.layers:
...             hidden_states = layer(hidden_states, *args, **kwargs)
...         return hidden_states
```

1.1.20 mindspeed_ms.legacy.model.moe.experts.SequentialMLP

class mindspeed_ms.legacy.model.moe.experts.**SequentialMLP**(*num_local_experts: int*, *config:* TransformerConfig, *submodules=None*)

定义 SequentialMLP 模块。

参数：

- **num_local_experts** (int) - 局部专家的数量。
- **config** (TransformerConfig) - Transformer 模型的配置，详情请参考 TransformerConfig 类。
- **submodules** (MLPSubmodules) - 线性全连接层的类型。保留参数，目前没有使用。

输入：

- **permuted_local_hidden_states** (Tensor) - 局部专家的隐藏层排列输入。
- **token_per_expert** (Tensor) - 每个专家的 token 数量。

输出：

两个张量的元组。

- **output_local** (Tensor) - 局部专家的输出。
- **output_bias_local** (Tensor) - 局部专家输出偏置。目前没有使用。默认返回 None 。

异常：

- **NotImplementedError** - 如果 *submodules* 不为 None 。

样例：

说明：

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore.communication import init
>>> from mindspeed_ms.core.config import TransformerConfig, ModelParallelConfig,␣
↪TrainingConfig
>>> from mindspeed_ms.legacy.model.moe.experts import SequentialMLP
```

(下页继续)

(续上页)

```
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> ms.set_context(device_target="Ascend", mode=ms.PYNATIVE_MODE, deterministic='ON')
>>> init()
>>> initialize_model_parallel()
>>> parallel_config = ModelParallelConfig()
>>> training_config = TrainingConfig(parallel_config=parallel_config)
>>> config = TransformerConfig(vocab_size=128,
...                            num_layers=1,
...                            num_attention_heads=8,
...                            num_query_groups=4,
...                            hidden_size=64,
...                            ffn_hidden_size=128,
...                            parallel_config=parallel_config,
...                            training_config=training_config)
>>> expert = SequentialMLP(num_local_experts=2, config=config)
>>> shape = (32, 64)
>>> tokens_per_expert = [20, 12]
>>> permuted_local_hidden_states = ms.Tensor(np.random.standard_normal(shape).astype(np.
↪float32))
>>> output_local, output_bias_local = expert(permuted_local_hidden_states, tokens_per_expert)
>>> print(output_local.shape)
(32, 64)
```

1.1.21 mindspeed_ms.legacy.model.moe.moe_layer.MoELayer

class mindspeed_ms.legacy.model.moe.moe_layer.**MoELayer**(*config:* TransformerConfig, *submodules=None*, *layer_number:* *int* = *None*)

专家层。

参数：

- **config** (TransformerConfig) - Transformer 模型的配置，详情请参考 TransformerConfig 类。
- **submodules** (MLPSubmodules，可选) - 保留参数，目前没有使用。默认值：`None` 。
- **layer_number** (int，可选) - 保留参数，目前没有使用。默认值：`None` 。

输入：

- **hidden_states** (Tensor) - 局部专家的隐藏层输入。

输出：

两个张量的元组。

- **output** (Tensor) - 局部专家的输出。
- **mlp_bias** (Tensor) - 目前没有使用。

异常：

- **ValueError** - 如果 *ep_world_size* 小于等于 `0` 。
- **ValueError** - 如果 *num_experts* 不能被 *ep_world_size* 整除。
- **ValueError** - 如果 *local_expert_indices* 的元素数量大于等于 *num_experts* 。
- **ValueError** - 如果 *moe_config.moe_token_dispatcher_type* 不为 *alltoall* 。
- **ValueError** - 如果 *self.training* 为 `True` 且 *get_tensor_model_parallel_world_size()* 大于 `1` 且 *self.sp* 不为 `True` 。

样例：

说明：

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore.communication import init
>>> from mindspeed_ms.core.config import TransformerConfig, ModelParallelConfig,␣
↪TrainingConfig, MoEConfig
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> from mindspeed_ms.legacy.model.moe.moe_layer import MoELayer
>>> ms.set_context(device_target="Ascend", mode=ms.PYNATIVE_MODE, deterministic='ON')
>>> init()
>>> initialize_model_parallel()
>>> parallel_config = ModelParallelConfig()
>>> training_config = TrainingConfig(parallel_config=parallel_config)
>>> moe_config = MoEConfig(num_experts=4,
...                        moe_router_topk=2)
>>> config = TransformerConfig(vocab_size=128,
...                            num_layers=1,
...                            num_attention_heads=1,
...                            num_query_groups=1,
...                            hidden_size=64,
...                            ffn_hidden_size=128,
...                            parallel_config=parallel_config,
...                            training_config=training_config,
...                            moe_config=moe_config)
>>> mlp = MoELayer(config)
>>> shape = (8, 2, 64)
>>> hidden_states = ms.Tensor(np.random.standard_normal(shape).astype(np.float32))
>>> output, mlp_bias= mlp(hidden_states)
>>> print(output.shape)
(8, 2, 64)
```

1.1.22 mindspeed_ms.legacy.model.moe.router.TopKRouter

class mindspeed_ms.legacy.model.moe.router.**TopKRouter**(*config:* TransformerConfig)

Top-K 路由，负责根据输入数据计算分数，并选择 Top-K 个专家来处理输入数据。

参数：

- **config** (TransformerConfig) - Transformer 模型的配置，详情请参考 TransformerConfig 类。

输入：

- **input** (Tensor) - 输入张量。

输出：

两个张量组成的元组。

- **scores** (Tensor) - 负载均衡后的概率张量。
- **indices** (Tensor) - 经过 top-k 选择后的索引张量。

异常：

- **NotImplementedError** - 如果 *moe_config.moe_router_load_balancing_type* 为 `sinkhorn` 。
- **ValueError** - 如果 *moe_config.moe_router_load_balancing_type* 不是 *sinkhorn* ， 不是 *aux_loss* ， 也不是 *none* 。

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import numpy as np
>>> from mindspeed_ms.core.config import ModelParallelConfig, MoEConfig, TrainingConfig,␣
↪TransformerConfig
>>> from mindspeed_ms.legacy.model.moe.router import TopKRouter
>>> from mindspore.communication.management import init
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> import mindspore as ms
>>> import mindspore.common.dtype as mstype
>>> ms.set_context(device_target="Ascend", mode=ms.PYNATIVE_MODE, deterministic='ON')
>>> init()
>>> initialize_model_parallel(tensor_model_parallel_size=1,expert_model_parallel_size=1)
>>> parallel_config = ModelParallelConfig(tensor_model_parallel_size=1)
>>> training_config = TrainingConfig(parallel_config=parallel_config)
>>> moe_cfg = MoEConfig(num_experts=4, moe_router_topk=2)
>>> model_cfg = TransformerConfig(
...     vocab_size=1,
...     num_layers=1,
...     num_attention_heads=1,
...     seq_length=8,
...     hidden_size=16,
...     ffn_hidden_size=64,
...     gated_linear_unit=True,
...     param_init_type=mstype.float32,
...     parallel_config=parallel_config,
...     moe_config=moe_cfg,
...     training_config=training_config,
... )
>>> router = TopKRouter(model_cfg)
>>> data = ms.Tensor(np.random.random((32, 16)).astype(np.float32))
>>> scores, indices = router(data)
>>> print(scores.shape)
(32 ,2)
```

1.1.23 mindspeed_ms.legacy.model.moe.token_dispatcher.MoEAlltoAllTokenDispatcher

class mindspeed_ms.legacy.model.moe.token_dispatcher.**MoEAlltoAllTokenDispatcher**(*num_local_experts: int*, *local_expert_indices: List[int]*, *config: TransformerConfig*)

在 MoE 架构中，MoEAlltoAllTokenDispatcher 调度器负责将 token 令牌分配给各个专家进行处理，并将处理后的结果重新组合回原始的 token 顺序。

参数：

- **num_local_experts** (int) - 表示当前 *rank* 有多少专家。
- **local_expert_indices** (List[int]) - 当前 *rank* 中专家的索引序号。
- **config** (TransformerConfig) - Transformer 模型的配置，详情请参考 TransformerConfig 类。

异常：

- **ValueError** - 如果 *num_local_experts* 不大于 `0` 。
- **ValueError** - 如果 *local_expert_indices* 的元素数量不等于 `num_local_experts` 。

样例：

说明：

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore.communication import init
>>> from mindspeed_ms.core.config import TransformerConfig, ModelParallelConfig,␣
↪TrainingConfig, MoEConfig
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> from mindspeed_ms.legacy.model.moe.token_dispatcher import MoEAlltoAllTokenDispatcher
>>> num_local_experts = 4
>>> ms.set_seed(1024)
>>> ms.set_context(device_target="Ascend", mode=ms.PYNATIVE_MODE, deterministic='ON')
>>> init()
>>> initialize_model_parallel()
>>> parallel_config = ModelParallelConfig()
>>> training_config = TrainingConfig(parallel_config=parallel_config)
>>> moe_config = MoEConfig(num_experts=num_local_experts,
...                        moe_router_topk=2)
>>> config = TransformerConfig(vocab_size=128,
...                            num_layers=1,
...                            num_attention_heads=1,
...                            num_query_groups=1,
...                            hidden_size=64,
...                            ffn_hidden_size=128,
...                            parallel_config=parallel_config,
```

(下页继续)

(续上页)

```
...                                 training_config=training_config,
...                                 moe_config=moe_config)
>>> dispatcher = MoEAlltoAllTokenDispatcher(num_local_experts=num_local_experts,
...                                         local_expert_indices=range(num_local_experts),
...                                         config=config)
>>> hidden_states = ms.Tensor(np.random.standard_normal((8, 2, 64)).astype(np.float32))
>>> hidden_states = hidden_states.reshape(-1, hidden_states.shape[-1])
>>> scores_first_column = np.random.rand(16, 1)
>>> complementary_scores = 1 - scores_first_column
>>> scores = ms.Tensor(np.hstack((scores_first_column, complementary_scores)))
>>> indices_array = np.array([np.random.choice(num_local_experts, size=2, replace=False) for␣
↪_ in range(16)])
>>> indices = ms.Tensor(indices_array, dtype=ms.int32)
>>> dispatched_input, tokens_per_expert = dispatcher.token_permutation(hidden_states, scores,
↪ indices)
>>> print(dispatched_input.shape)
(32, 64)
>>> print(tokens_per_expert)
[7 9 7 9]
>>> expert_output = ms.Tensor(np.random.standard_normal((32, 64)).astype(np.float32))
>>> output, _ = dispatcher.token_unpermutation(expert_output, bias=None)
>>> print(output.shape)
(16, 64)
```

1.1.24 mindspeed_ms.legacy.model.ParallelAttention

class mindspeed_ms.legacy.model.**ParallelAttention**(*config*, *layer_number*, *attention_type=AttnType.self_attn*, *attn_mask_type=AttnMaskType.padding*)

该类表示并行注意力机制。它可以处理不同的注意力类型，并且可以使用各种参数进行配置。

参数：

- **config** (dict) - Transformer 模型配置。详情请参考 TransformerConfig 类。
- **layer_number** (int) - 该 transformer 层在整个 transformer 块中的索引。
- **attention_type** (int，可选) - 注意力类型。支持 1 为 self_attn，2 为 cross_attn，默认值：`AttnType.self_attn` 。
- **attn_mask_type** (int，可选) - 注意力掩码类型。支持 1 为 padding，2 为 causal，默认值：`AttnMaskType.padding` 。

输入：

- **hidden_states** (Tensor) - 隐藏层状态张量，形状为 (B, S, H) 。
- **attention_mask** (Tensor) - attention 掩码矩阵，形状为 (B, N, S_q, S_k) 。
- **encoder_output** (Tensor，可选) - 用于交叉注意力的编码器输出张量。默认值：`None` 。
- **inference_params** (Tensor，可选) - 推理参数的张量，当前不支持该参数。默认值：`None` 。
- **rotary_pos_emb** (Tensor，可选) - 旋转位置嵌入张量。默认值：`None` 。

输出：

- **output** (Tensor) - 输出张量形状为 (B, S, H) 。
- **bias** (Tensor) - 可训练的偏置参数。

异常：

- **NotImplementedError** - 如果使用了 flash attention，但是 *attention_type* 是 *AttnType.self_attn* 。
- **ValueError** - 如果 *group_query_attention* 是 *True* 但是 *num_query_groups* 不能被 *tp_group_size* 整除。
- **ValueError** - 如果 *attention_type* 既不是 `AttnType::self_attn` 也不是 `AttnType::cross_attn` 。
- **NotImplementedError** - 如果 *attention_type* 是 2 并且 *config* 中的 *group_query_attention* 是 `True` 。
- **ValueError** - 如果 *config* 中的 *hidden_size* 不等于 *config* 中的 *kv_hidden_size* 并且 *attention_type* 是 2。
- **NotImplementedError** - 如果 *get_context_parallel_world_size() > 1* 并且 *args.context_parallel_algo* 为 *ulysses_cp_algo* 并且没有使用 flash attention。
- **NotImplementedError** - 如果 *inference_params* 不是 `None` 。

支持平台：

`Ascend`

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> from mindspore import Tensor
>>> from mindspore.communication.management import init
>>> from mindspeed_ms.legacy.model import ParallelAttention
>>> from mindspeed_ms.core.config import (ModelParallelConfig,
...                                       TrainingConfig,
...                                       TransformerConfig)
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> class MyNet(nn.Cell):
...     def __init__(self, config):
...         super(MyNet, self).__init__()
...         self.attention = ParallelAttention(layer_number=1, config=config)
...     def construct(self, x, attention_mask):
...         output, _ = self.attention(x, attention_mask)
...         return output
>>> ms.set_context(device_target="Ascend", mode=ms.PYNATIVE_MODE,
...     deterministic='ON')
>>> init()
>>> initialize_model_parallel(tensor_model_parallel_size=2)
>>> parallel_config = ModelParallelConfig(tensor_model_parallel_size=2)
>>> training_config = TrainingConfig(parallel_config=parallel_config)
>>> config = TransformerConfig(vocab_size=1,
>>>                            num_layers=1,
>>>                            num_attention_heads=8,
>>>                            num_query_groups=4,
>>>                            hidden_size=256,
>>>                            ffn_hidden_size=256,
>>>                            parallel_config=parallel_config,
>>>                            training_config=training_config)
```

(下页继续)

(续上页)

```
>>> input_shape = (32, 1024, 256)
>>> input = Tensor(np.random.standard_normal(input_shape).astype(np.float32))
>>> mask = np.ones((32, 1024, 1024), dtype=np.uint8)
>>> mask = Tensor(np.expand_dims(mask, axis=1))
>>> out = MyNet(config=config)(input, mask)
>>> print(out.shape)
(32, 1024, 256)
```

1.1.25 mindspeed_ms.legacy.model.ParallelLMLogits

class mindspeed_ms.legacy.model.**ParallelLMLogits**(*config*, *bias=False*, *compute_dtype=None*)

计算 vocab 中每一个 token 的 logits。

参数：

- **config** (dict) - Transformer 模型配置。详情请参考 TransformerConfig 类。
- **bias** (bool，可选) - 指定模型是否使用偏置向量。默认值：`False` 。
- **compute_dtype** (dtype.Number，可选) - 计算类型。默认值：`None` 。

输入：

- **input_** (Tensor) - 隐藏状态的张量。
- **word_embedding_table** (Parameter) - 从嵌入层通过的权重矩阵。
- **parallel_output** (bool，可选) - 指定是否返回各张量并行权重上的并行输出。默认值：`True` 。
- **bias** (Tensor，可选) - 可训练的偏置参数。默认值：`None` 。

输出：

- **logits_parallel** (Tensor) - 如果在 ParallelLMLogits 中设置 parallel_output 为 `True` ，则每个张量并行等级上的输出将是一个并行的 logits 张量，否则，输出将是一个收集所有并行输出的 logits 张量。

支持平台：

`Ascend`

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> from mindspore.communication.management import init
>>> from mindspeed_ms.core.config import ModelParallelConfig, TrainingConfig,␣
↪TransformerConfig
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> from mindspeed_ms.legacy.model import ParallelLMLogits
>>> ms.set_context(device_target="Ascend", mode=ms.PYNATIVE_MODE, deterministic='ON')
>>> init()
```

(下页继续)

(续上页)

```
>>> initialize_model_parallel(tensor_model_parallel_size=2)
>>> parallel_config = ModelParallelConfig(tensor_model_parallel_size=2)
>>> training_config = TrainingConfig(parallel_config=parallel_config)
>>> config = TransformerConfig(seq_length=16,
...                            vocab_size=1,
...                            num_layers=1,
...                            num_attention_heads=8,
...                            num_query_groups=4,
...                            hidden_size=256,
...                            ffn_hidden_size=256,
...                            parallel_config=parallel_config,
...                            training_config=training_config)
>>> model = ParallelLMLogits(config=config, bias=False, compute_dtype=ms.float32)
>>> input = Tensor(np.random.random((2, 3, 6)).astype(np.float32))
>>> word_emb = Tensor(np.random.random((6, 6)).astype(np.float32))
>>> logits = model(input, word_emb, parallel_output=True)
>>> print(logits.shape)
(2, 3, 6)
```

1.1.26 mindspeed_ms.legacy.model.ParallelMLP

class mindspeed_ms.legacy.model.**ParallelMLP**(*config*, *is_expert=False*)

并行前馈模块实现。

参数：

- **config** (TransformerConfig) - transformer 模型的 config。
- **is_expert** (bool，可选) - 指定这个 block 是否是专家. 默认值：`False` 。

输入：

- **hidden_states** (Tensor) - 一个形状为 (B, S, H) 或 (S, B, H) 的张量。

输出：

- **output** (Tensor) - 一个形状为 (B, S, H) 或 (S, B, H) 的张量。

支持平台：

`Ascend`

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> from mindspore.communication.management import init
>>> from mindspeed_ms.core.config import (ModelParallelConfig,
...                                       TrainingConfig,
...                                       TransformerConfig)
```

(下页继续)

(续上页)

```
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> from mindspeed_ms.legacy.model.transformer import ParallelMLP
>>> ms.set_seed(2024)
>>> init()
>>> initialize_model_parallel()
>>> parallel_config = ModelParallelConfig(tensor_model_parallel_size=1)
>>> training_config = TrainingConfig(parallel_config=parallel_config)
>>> seq_length=4
>>> dataset_size=1
>>> hidden_size=8
>>> model_config = TransformerConfig(vocab_size=4000,
...                                  num_layers=1,
...                                  num_attention_heads=4,
...                                  hidden_size=hidden_size,
...                                  ffn_hidden_size=hidden_size * 4,
...                                  parallel_config=parallel_config,
...                                  training_config=training_config)
>>> mlp = ParallelMLP(config=model_config)
>>> shape = (seq_length, dataset_size, hidden_size)
>>> input_data = Tensor(np.random.random(shape).astype(np.float32))
>>> output, _ = mlp(input_data)
>>> print(output.shape)
(4, 1, 8)
```

1.1.27 mindspeed_ms.legacy.model.ParallelTransformer

class mindspeed_ms.legacy.model.**ParallelTransformer**(*config*, *model_type*, *layer_type=LayerType.encoder*, *self_attn_mask_type=AttnMaskType.padding*, *post_norm=True*, *pre_process=False*, *post_process=False*, *drop_path_rate=0.0*)

Transformer 模块。它由多个单独的 transformer 层组成，可以处理各种配置和处理步骤。

参数：

- **config** (dict) - 一个配置字典，提供了并行 transformer 层的各种参数配置。
- **model_type** (int) - model 类型。支持 1 为 encoder_or_decoder，2 为 encoder_and_decoder，3 为 retro_encoder，4 为 retro_decoder。
- **layer_type** (int，可选) - layer 类型。支持 1 为 encoder，2 为 decoder，3 为 retro_encoder，4 为 retro_decoder，5 为 retro_decoder_with_retriever, 默认值：`LayerType.encoder` 。
- **self_attn_mask_type** (int，可选) - 注意力 mask 类型。支持 1 为 padding，2 为 causal，默认值：`AttnMaskType.padding` 。
- **post_norm** (bool，可选) - 是否在转换器块的末尾插入归一化层。默认值：`True` 。
- **pre_process** (bool，可选) - 使用流水线并行时，表明它是否是第一阶段。默认值：`False` 。
- **post_process** (bool，可选) - 使用流水线并行时，表明它是否是最后一个阶段。默认值：`False` 。
- **drop_path_rate** (float，可选) - 丢弃率。当前不支持该参数大于 0，默认值：`0.0` 。

输入：

- **hidden_states** (Tensor) - 隐藏层状态张量，形状为 (B, S, H) 。
- **attention_mask** (Tensor) - 注意力掩码张量。

- **encoder_output** (Tensor，可选) - 用于交叉注意力的编码器输出张量，当前不支持该参数。默认值：`None` 。
- **enc_dec_attn_mask** (Tensor，可选) - 编码器-解码器注意力 mask 张量，当前不支持该参数。默认值：`None` 。
- **retriever_input** (Tensor，可选) - 检索输入张量，当前不支持该参数。默认值：`None` 。
- **retriever_output** (Tensor，可选) - 检索输出张量，当前不支持该参数。默认值：`None` 。
- **retriever_attn_mask** (Tensor，可选) - 检索注意力 mask 张量，当前不支持该参数。默认值：`None` 。
- **inference_params** (Tensor，可选) - 推理参数的张量，当前不支持该参数。默认值：`None` 。
- **rotary_pos_emb** (Tensor，可选) - 旋转位置嵌入张量。默认值：`None` 。

输出：

- **hidden_states** (Tensor) - 输出张量形状为 (B, S, H) 。

异常：

- **NotImplementedError** - 如果 *drop_path_rate* 大于 `0` 。
- **NotImplementedError** - 如果 *config* 中的 *distribute_saved_activations* 是 `True` 并且 *config* 中的 *sequence_parallel* 是 `False` 。
- **NotImplementedError** - 如果 *config* 中的 *transformer_impl* 是 `transformer_engine` 。
- **NotImplementedError** - 如果 *config* 中的 *fp8* 不是 `None` 。
- **NotImplementedError** - 如果 *config* 中的 *retro_add_retriever* 是 `True` 。
- **NotImplementedError** - 如果 *model_type* 是 `3` 或 `4` 。
- **NotImplementedError** - 如果 *encoder_output* 、*enc_dec_attn_mask* 、*retriever_input* 、*retriever_output* 、*retriever_attn_mask* 或 *inference_params* 不是 `None` 。

支持平台：

`Ascend`

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import mindspore.common.dtype as mstype
>>> from mindspore import Tensor
>>> from mindspore.communication.management import init
>>> from mindspeed_ms.legacy.model import ParallelTransformer
>>> from mindspeed_ms.core.config import (ModelParallelConfig,
...                                       TrainingConfig,
...                                       TransformerConfig)
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> class MyNet(nn.Cell):
...     def __init__(self, config):
...         super(MyNet, self).__init__()
...         self.transformer = ParallelTransformer(config=config, model_type=None)
```

(下页继续)

(续上页)

```
...     def construct(self, x, attention_mask):
...         output = self.transformer(x, attention_mask)
...         return output
>>> ms.set_context(device_target="Ascend", mode=ms.PYNATIVE_MODE, deterministic='ON')
>>> init()
>>> initialize_model_parallel(tensor_model_parallel_size=2)
>>> parallel_config = ModelParallelConfig(tensor_model_parallel_size=2)
>>> training_config = TrainingConfig(parallel_config=parallel_config)
>>> config = TransformerConfig(seq_length=16,
>>>                            vocab_size=1,
>>>                            num_layers=1,
>>>                            num_attention_heads=8,
>>>                            num_query_groups=4,
>>>                            hidden_size=256,
>>>                            ffn_hidden_size=256,
>>>                            parallel_config=parallel_config,
>>>                            training_config=training_config)
>>> input_shape = (32, 1024, 256)
>>> input = Tensor(np.random.standard_normal(input_shape).astype(np.float32))
>>> mask = Tensor(np.triu(np.ones((1024, 1024)), 1), mstype.uint8)
>>> out = MyNet(config=config)(input, mask).shape
>>> print(out)
(32, 1024, 256)
```

1.1.28 mindspeed_ms.legacy.model.ParallelTransformerLayer

class mindspeed_ms.legacy.model.**ParallelTransformerLayer**(*config*, *layer_number*, *layer_type=LayerType.encoder*, *self_attn_mask_type=AttnMaskType.padding*, *drop_path_rate=0.0*)

单独的一层 transformer。它结合了归一化、注意力、交叉注意力和 MLP 来处理输入隐藏状态。

参数：

- **config** (dict) - 一个配置字典，提供了 transformer 层的各种参数配置。
- **layer_number** (int) - 该 transformer 层在整个 transformer 块中的索引。
- **layer_type** (int，可选) - layer 类型。支持 1 为 encoder，2 为 decoder，3 为 retro_encoder，4 为 retro_decoder，5 为 retro_decoder_with_retriever, 默认值：`LayerType.encoder` 。
- **self_attn_mask_type** (int，可选) - 注意力 mask 类型。支持 1 为 padding，2 为 causal，默认值：`AttnMaskType.padding` 。
- **drop_path_rate** (float，可选) - drop_path rate。当前不支持该参数大于 0，默认值：`0.0` 。

输入：

- **hidden_states** (Tensor) - 隐藏层状态张量，形状为 (B, S, H) 。
- **attention_mask** (Tensor) - attention 掩码矩阵。
- **encoder_output** (Tensor，可选) - 用于交叉注意力的编码器输出张量，当前不支持该参数。默认值：`None` 。
- **enc_dec_attn_mask** (Tensor，可选) - 编码器-解码器注意力 mask 张量，当前不支持该参数。默认值：`None` 。
- **retriever_input** (Tensor，可选) - 检索输入张量，当前不支持该参数。默认值：`None` 。
- **retriever_output** (Tensor，可选) - 检索输出张量，当前不支持该参数。默认值：`None` 。

- **retriever_attn_mask** (Tensor，可选) - 检索注意力 mask 张量，当前不支持该参数。默认值：`None` 。
- **inference_params** (Tensor，可选) - 推理参数的张量，当前不支持该参数。默认值：`None` 。
- **rotary_pos_emb** (Tensor，可选) - 旋转位置嵌入张量。默认值：`None` 。

输出：

- **output** (Tensor) - 输出张量形状为 (B, S, H) 。

异常：

- **NotImplementedError** - 如果 *config* 中的 *bias_dropout_fusion* 是 `True` 。
- **NotImplementedError** - 如果 *drop_path_rate* 大于 0。
- **NotImplementedError** - 如果 *config* 中的 *retro_add_retriever* 是 `True` 。
- **NotImplementedError** - 如果 *encoder_output* 、*enc_dec_attn_mask* 、*retriever_input* 、*retriever_output* 、*retriever_attn_mask* 或 *inference_params* 不是 `None` 。

支持平台：

`Ascend`

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import mindspore.common.dtype as mstype
>>> from mindspore import Tensor
>>> from mindspore.communication.management import init
>>> from mindspeed_ms.legacy.model import ParallelTransformerLayer
>>> from mindspeed_ms.core.config import ModelParallelConfig, TrainingConfig,␣
↪TransformerConfig
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> class MyNet(nn.Cell):
...     def __init__(self, config):
...         super(MyNet, self).__init__()
...         self.layer = ParallelTransformerLayer(layer_number=1, config=config)
...     def construct(self, x, attention_mask):
...         output = self.layer(x, attention_mask)
...         return output
...
>>> ms.set_context(device_target="Ascend", mode=ms.PYNATIVE_MODE, deterministic='ON')
>>> init()
>>> initialize_model_parallel(tensor_model_parallel_size=2)
>>> parallel_config = ModelParallelConfig(tensor_model_parallel_size=2)
>>> training_config = TrainingConfig(parallel_config=parallel_config)
>>> config = TransformerConfig(vocab_size=1,
>>>                            num_layers=1,
>>>                            num_attention_heads=8,
>>>                            num_query_groups=4,
>>>                            hidden_size=256,
```

(下页继续)

(续上页)

```
>>>                             ffn_hidden_size=256,
>>>                             parallel_config=parallel_config,
>>>                             training_config=training_config)
>>> input_shape = (32, 1024, 256)
>>> input = Tensor(np.random.standard_normal(input_shape).astype(np.float32))
>>> mask = Tensor(np.triu(np.ones((1024, 1024)), 1), mstype.uint8)
>>> out = MyNet(config=config)(input, mask).shape
>>> print(out)
(32, 1024, 256)
```

1.1.29 mindspeed_ms.legacy.model.RotaryEmbedding

class mindspeed_ms.legacy.model.**RotaryEmbedding**(*kv_channels*, *rotary_percent=1.0*, *rotary_interleaved=False*, *seq_len_interpolation_factor=None*, *rotary_base=10000*)

用于语言模型的旋转位置嵌入。

参数：

- **kv_channels** (int) - 多头注意力中的投影权重维度。从 transformer 的配置中获取。
- **rotary_percent** (float, 可选) - 旋转位置编码中旋转维度的使用比例。默认值：`1.0` 。
- **rotary_interleaved** (bool, 可选) - 是否以交错方式将旋转位置编码应用于输入维度。默认值：`False` 。目前暂不支持设置为 `True` 。
- **seq_len_interpolation_factor** (float, 可选) - 对更长序列进行线性插值的比例。如果设置非 None，则该值必须是大于 1.0 的浮点数。默认值：`None` 。
- **rotary_base** (int, 可选) - 旋转位置嵌入编码的基期。默认值：`10000` 。

输入：

- **max_seq_len** (int) - 输入的最大序列长度。
- **offset** (int) - 位置编码偏移量。

输出：

- **emb** (Tensor) - 应用旋转位置编码后的嵌入向量。

异常：

- **NotImplementedError** - 当 *rotary_interleaved* 为 `True` 。

支持平台：

`Ascend`

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import numpy as np
>>> import mindspore.nn as nn
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> from mindspore.communication import init
>>> from mindspeed_ms.legacy.model.rotary_pos_embedding import RotaryEmbedding
>>> from mindspeed_ms.core.config import ModelParallelConfig, TrainingConfig,␣
↪TransformerConfig
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> class MyNet(nn.Cell):
...     def __init__(self, config: TransformerConfig):
...         super(MyNet, self).__init__()
...         self.rotary_embedding = RotaryEmbedding(config.seq_length)
...     def construct(self, x: Tensor):
...         emb = self.rotary_embedding(x.shape[1])
...         return emb
>>> ms.set_context(device_target="Ascend", mode=ms.PYNATIVE_MODE, deterministic='ON')
>>> init()
>>> initialize_model_parallel(tensor_model_parallel_size=1, context_parallel_size=1)
>>> parallel_config = ModelParallelConfig(tensor_model_parallel_size=1)
>>> training_config = TrainingConfig(parallel_config=parallel_config)
>>> config = TransformerConfig(seq_length=16,
...                            vocab_size=1,
...                            num_layers=1,
...                            num_attention_heads=16,
...                            hidden_size=256,
...                            ffn_hidden_size=256,
...                            parallel_config=parallel_config,
...                            training_config=training_config)
>>> bs = 2
>>> seq_len = 16
>>> hidden_size = 256
>>> input_shape = (bs, seq_len, hidden_size)
>>> net = MyNet(config)
>>> input = Tensor(np.random.standard_normal(input_shape).astype(np.float32))
>>> output = net(input)
>>> print(output.shape)
(16, 1, 1, 16)
```

1.1.30 mindspeed_ms.legacy.model.transformer.CoreAttention

class mindspeed_ms.legacy.model.transformer.**CoreAttention**(*layer_number*, *config*, *attn_mask_type=AttnMaskType.padding*)

核心注意力机制，用于计算查询、键和值层之间的注意力权重和上下文表示。

参数：

- **layer_number** (int) - 该 transformer 层在整个 transformer 块中的索引。
- **config** (dict) - Transformer 模型的配置，详情请参考 TransformerConfig 类。
- **attn_mask_type** (int，可选) - Attention mask type，支持 [AttnMaskType::padding = 1, AttnMaskType::causal = 2]。默认为 `1` 。

输入：

- **query_layer** (Tensor) - 查询层。形状为 (S, B, N, D) 。

- **key_layer** (Tensor) - 键层。形状为 (S, B, N, D) 。
- **value_layer** (Tensor) - 值层。形状为 (S, B, N, D) 。
- **attention_mask** (Tensor) - 注意力掩码。形状为 (B, S_q, S_k) 。

输出：

- **context_layer** (Tensor) - 形状为 (S, B, H) 。

异常：

- **NotImplementedError** - 如果 *config* 中的 *masked_softmax_fusion* 为 `True` 。

支持平台：

`Ascend`

样例：

说明：

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> from mindspore import Tensor
>>> from mindspore.communication.management import init
>>> from mindspeed_ms.legacy.model.transformer import CoreAttention
>>> from mindspeed_ms.core.config import ModelParallelConfig, TransformerConfig
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> class MyNet(nn.Cell):
...     def __init__(self, config):
...         super(MyNet, self).__init__()
...         self.core_attn = CoreAttention(layer_number=1, config=config)
...     def construct(self, x, mask):
...         out = self.core_attn(x, x, x, mask)
...         return out
>>> ms.set_context(device_target="Ascend", mode=ms.PYNATIVE_MODE, deterministic='ON')
>>> init()
>>> initialize_model_parallel(tensor_model_parallel_size=2)
>>> parallel_config = ModelParallelConfig(tensor_model_parallel_size=2)
>>> config = TransformerConfig(vocab_size=1,
...                            num_layers=1,
...                            num_attention_heads=8,
...                            num_query_groups=4,
...                            hidden_size=256,
...                            ffn_hidden_size=256,
...                            parallel_config=parallel_config,
...                            training_config=None)
>>> input_shape = (1024, 1, 4, 32)
>>> input = Tensor(np.random.standard_normal(input_shape).astype(np.float32))
>>> mask = np.ones((1, 1024, 1024), dtype=np.uint8)
>>> mask = Tensor(np.expand_dims(mask, axis=1))
>>> out = MyNet(config=config)(input, mask)
>>> print(out.shape)
(1024, 1, 128)
```

1.1.31 mindspeed_ms.legacy.model.TransformerLanguageModel

class mindspeed_ms.legacy.model.**TransformerLanguageModel**(*config*, *encoder_attn_mask_type*, *num_tokentypes=0*, *add_encoder=True*, *add_decoder=False*, *decoder_attn_mask_type=AttnMaskType.causal*, *add_pooler=False*, *pre_process=True*, *post_process=True*, *visual_encoder=None*, ***kwargs*)

Transformer 语言模型。

参数：

- **config** (TransformerConfig) - Transformer 模型配置，包括初始化函数和并行参数配置等。
- **encoder_attn_mask_type** (int) - 编码器注意力掩码类型。
- **num_tokentypes** (int，可选) - 如果大于 0，则使用 tokentype 嵌入。默认值：`0` 。
- **add_encoder** (bool，可选) - 如果为 `True` ，使用编码器。默认值：`True` 。
- **add_decoder** (bool，可选) - 如果为 `True` ，使用解码器。默认值：`False` 。
- **decoder_attn_mask_type** (int，可选) - 解码器注意力掩码类型。默认值：`AttnMaskType.causal` 。
- **add_pooler** (bool，可选) - 如果为 `True` ，使用池化层。默认值：`False` 。
- **pre_process** (bool，可选) - 使用流水线并行时，标记它是否为第一阶段。默认值：`True` 。
- **post_process** (bool，可选) - 使用流水线并行时，标记它是否为最后的阶段。默认值：`True` 。
- **visual_encoder** (nn.Cell，可选) - 视觉编码器。默认值：`None` 。
- **kwargs** (dict) - 其他输入。

输入：

- **enc_input_ids** (Tensor) - 编码器输入索引。形状为 (B, S) 。
- **enc_position_ids** (Tensor) - 编码器位置偏移量。形状为 (B, S) 。
- **enc_attn_mask** (Tensor) - 编码器注意力掩码。形状为 (B, S) 。
- **dec_input_ids** (Tensor，可选) - 解码器输入索引。形状为 (B, S) 。默认值：`None` 。
- **dec_position_ids** (Tensor，可选) - 解码器输入位置索引。形状为 (B, S) 。默认值：`None` 。
- **dec_attn_mask** (Tensor，可选) - 解码器注意力掩码。形状为 (B, S) 。默认值：`None` 。
- **retriever_input_ids** (Tensor，可选) - 检索器输入标记索引。默认值：`None` 。
- **retriever_position_ids** (Tensor，可选) - 检索器输入位置索引。默认值：`None` 。
- **retriever_attn_mask** (Tensor，可选) - 检索器注意力掩码，用于控制在检索器中计算注意力时的注意范围。默认值：`None` 。
- **enc_dec_attn_mask** (Tensor，可选) - 编码器-解码器注意力掩码，用于在编码器和解码器之间计算注意力时使用。默认值：`None` 。
- **tokentype_ids** (Tensor，可选) - 给模型输入的标记类型索引列表。形状为 (B, S) 。默认值：`None` 。
- **inference_params** (InferenceParams，可选) - 推理参数，用于在推理过程中指定特定设置，如最大生成长度、最大批处理大小等。默认值：`None` 。
- **pooling_sequence_index** (int，可选) - 池化序列索引。默认值：`0` 。
- **enc_hidden_states** (Tensor，可选) - 编码器隐藏层。默认值：`None` 。

- **output_enc_hidden** (bool，可选) - 是否输出编码器隐藏层。默认值：`False` 。
- **input_image** (Tensor，可选) - 输入图像的张量。形状为 $(N, C_{in}, H_{in}, W_{in})$ 或 $(N, H_{in}, W_{in}, C_{in},)$ 。默认值：`None` 。
- **delimiter_position** (Tensor，可选) - 分隔符位置张量。形状为 (B, N) ，其中 N 表示分隔符数量。默认值：`None` 。
- **image_embedding** (Tensor，可选) - 图像嵌入张量，维度依赖于图像嵌入的维数。默认值：`None` 。

输出：

- **encoder_output** (Tensor) - 形状为 (B, S, H) 或 (S, B, H) 的张量。

异常：

- **ValueError** - 如果 *config.untie_embeddings_and_output_weights* 且 *add_decoder* 为 `True` 。
- **RuntimeError** - 如果 *input_tensor* 长度为 *1* 。
- **NotImplementedError** - 如果 *config.retro_add_retriever* 为 `True` 。
- **NotImplementedError** - 如果 *visual_encoder* 或者 *add_decoder* 为 `True` 。
- **NotImplementedError** - 如果 *dec_input_ids* 、*dec_position_ids* 、*dec_attn_mask* 、*retriever_input_ids* 、*retriever_position_ids* 、*retriever_attn_mask* 、*enc_dec_attn_mask* 、*input_image* 、*delimiter_position* 或者 *image_embedding* 不为 `None` 。
- **NotImplementedError** - 如果 *output_enc_hidden* 为 `True` 。

支持平台：

`Ascend`

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import os
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> from mindspore.communication import init
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> from mindspeed_ms.legacy.model import TransformerLanguageModel
>>> from mindspeed_ms.core.config import (
...     init_configs_from_yaml,
...     TrainingConfig,
...     ModelParallelConfig,
...     TransformerConfig,
...     DatasetConfig,
... )
>>> os.environ['HCCL_BUFFSIZE'] = "1"
>>> CONFIG_PATH = "test_language_model.yaml"
>>> training_config, parallel_config, dataset_config, model_config = init_configs_from_yaml(
...     CONFIG_PATH, [TrainingConfig, ModelParallelConfig, DatasetConfig, TransformerConfig]
... )
>>> ms.set_context(device_target="Ascend", mode=ms.PYNATIVE_MODE, deterministic="ON")
```

(下页继续)

(续上页)

```
>>> init()
>>> initialize_model_parallel()
>>> batch_size = dataset_config.batch_size
>>> sq = model_config.seq_length
>>> language_model = TransformerLanguageModel(model_config, encoder_attn_mask_type=None)
>>> input_data = Tensor(np.random.random((batch_size, sq)).astype(np.int32))
>>> attention_mask = Tensor(np.zeros((batch_size, 1, sq, sq)).astype(np.int32))
>>> hidden_states = language_model(input_data, None, attention_mask)
>>> print(hidden_states.shape)
(8, 32, 64)
```

1.1.32 mindspeed_ms.training.loss_func.LossWithMask

class mindspeed_ms.training.loss_func.**LossWithMask**(*loss_func*, **args*, ***kwargs*)

使用掩码和均值计算损失。

参数：

- **loss_func** (Function) - 损失函数。
- **args** (tuple) - 位置参数。
- **kwargs** (dict) - 其他的关键字配置参数。

输入：

- **logits** (Tensor) - 主干网络的输出。形状为 (N, C) 的张量。数据类型必须为 float16 或 float32。
- **label** (Tensor) - 样本的真值。形状为 $(N,)$ 或与 *logits* 相同维度的张量。
- **input_mask** (Tensor) - *input_mask* 表示是否有填充的输入，对于填充的输入，它不会被计算到损失中。形状为 $(N,)$ 的张量。

输出：

- 对应的交叉熵。

样例：

说明**:**

- 运行样例之前，需要配置好环境变量。
- 针对 Ascend 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 msrun 启动 。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore.communication.management import init
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, nn
>>> from mindspeed_ms.core.parallel_state import initialize_model_parallel
>>> from mindspeed_ms.training.loss_func import LossWithMask
>>> ms.set_context(device_target='Ascend', mode=ms.PYNATIVE_MODE)
>>> ms.set_seed(2024)
>>> init()
>>> initialize_model_parallel()
>>> loss = LossWithMask(nn.CrossEntropyLoss())
```

(下页继续)

(续上页)

```
>>> logits = Tensor(np.array([[3, 5, 6, 9, 12, 33, 42, 12, 32, 72]]),
...                 mstype.float32)
>>> labels = Tensor(np.array([1]).astype(np.int32))
>>> input_mask = Tensor(np.ones(1).astype(np.float32))
>>> output = loss(logits, labels, input_mask)
>>> print(output)
67.0
```

1.1.33 mindspeed_ms.training.training.pretrain

mindspeed_ms.training.training.**pretrain**(*train_valid_test_datasets_provider*, *model_provider_func*, *model_type*, *forward_step_func=None*, *process_non_loss_data_func=None*, ***kwargs*)

预训练接口。用户通过此接口传入数据迭代器、模型定义、配置项等即可开始模型训练。

参数：

- **train_valid_test_datasets_provider** (function) - 一个输入为 train/valid/test 数据集大小，输出为对应的数据集的函数。当前未使能。
- **model_provider_func** (function) - 一个会返回模型的函数。
- **model_type** (enum) - 表示训练模型的类型的枚举类。当前未使能。
- **forward_step_func** (function，可选) - 前向函数，当前未使能。默认值：`None` 。
- **process_non_loss_data_func** (function，可选) - 对网络的输出进行后处理的函数。当前未使能。默认值：`None` 。
- **kwargs** (dict) - 其他输入。数据迭代器、配置项 *all_config* 等从这里传入。

异常：

- **ValueError** - *all_config* 没有从 kwargs 中传入。

支持平台：

`Ascend`

样例：

```
>>> from mindspeed_ms.training.training import pretrain
>>> from mindspeed_ms.core.config import TransformerConfig
>>> def model_provider_func(pre_process=True, post_process=True):
...     network_with_loss = GPTModel(
...         all_config.model_config,
...         pre_process=pre_process,
...         post_process=post_process
...     )
...     return network_with_loss
>>> all_config = TransformerConfig()
>>> pretrain(
...     train_valid_test_datasets_provider=None,
...     model_provider_func=model_provider_func,
...     model_type=None,
...     all_config=all_config
... )
```