PR #7442 完整报告

PaddlePaddle/FastDeploy

[Speculative Decoding] Add MTP logprob support for PD disaggregation

合并时间：2026-04-17 21:37

原文链接：http://prhub.com.cn/PaddlePaddle/FastDeploy/pull/7442

执行摘要

- 一句话：为 PD 分离部署下的 MTP 投机解码新增 prefill 节点首 token 的 logprob 支持。
- 推荐动作：该 PR 值得精读，重点关注以下设计决策：
  1. 消息结构体抽取：将 msgdata 和 batch_msgdata 抽取到共享头文件，减少了代码重复，但宏命名存在冲突风险，可学习其重构思路。
  2. 输出保存逻辑重构：save_output_specualate 函数新增参数以区分 prefill 和 decode 节点，体现了 PD 分离架构下的模块化设计，但需注意数据一致性和参数传递的边界条件。
  3. 算子语义设计：mtp_save_first_token_with_topk 算子的实现展示了如何扩展现有功能（添加 logprob 支持），但 cur_token_num 计算与上游语义的冲突揭示了接口设计的重要性。 建议结合 review 评论，在实际部署前验证非 MTP 投机方法的兼容性，并补充单元测试。

功能与动机

根据 PR body 描述，动机是“Enable logprob return for MTP speculative decoding under PD disaggregation architecture, particularly for handling the first token at prefill nodes.”，即在 PD 分离部署场景下，MTP 投机解码缺少对 logprob 的支持，尤其是 prefill 节点的首 token。

实现拆解

1. 新增 C++ 算子与共享头文件：
   - 新增 mtp_save_first_token_with_topk.cc 算子，支持在 prefill 节点保存带 logprob 的首 token 信息，通过 SysV 消息队列传递数据。
   - 新增 speculate_logprob_msg.h 头文件，抽取了消息结构体 msgdata 和 batch_msgdata，以及相关宏定义（如 SPEC_LOGPROB_MAX_BSZ、SPEC_LOGPROB_K），供多个算子复用，减少代码重复。
2. 重构 Python 端输出保存逻辑：
   - 修改 fastdeploy/model_executor/pre_and_post_process.py 中的 save_output_specualate 函数，新增参数 proposer_share_inputs、local_rank、tensor_parallel_rank 和 is_mtp_prefill，以区分 prefill 和 decode 节点的处理逻辑。
   - 在 prefill 分支中，根据是否有 logprob 数据，分别调用 mtp_save_first_token 或 mtp_save_first_token_with_topk 算子，并调整数据恢复逻辑，确保正确传递 token 和

logprob 信息。

3. 迁移调用路径并清理冗余代码：
   - 修改 fastdeploy/spec_decode/mtp.py，将 mtp_save_first_token 的调用从 MTP proposer 内部移动到 pre_and_post_process.py，仅保留 XPU 平台的调用，以统一 GPU 平台的逻辑。
   - 修改 fastdeploy/worker/gpu_model_runner.py，在 _save_model_output 中调整 save_output_specualate 的调用参数，传入 proposer 输入和 rank 信息，以支持 MTP prefill 首 token 保存。
4. 更新相关算子以使用共享头文件：
   - 修改 speculate_get_output_with_topk.cc 和 speculate_save_output_with_topk.cc，移除内联的消息结构体定义，改为引入 speculate_logprob_msg.h，统一宏定义（如将 K 替换为 SPEC_LOGPROB_K），提高代码可维护性。
5. 测试与配置配套：
   - 根据 review 评论，本 PR 未添加单元测试，存在回归风险；代码覆盖率报告显示 patch 覆盖率为 26.32%，有 14 行代码缺少覆盖。

关键文件：

- custom_ops/gpu_ops/speculate_decoding/draft_model/mtp_save_first_token_with_topk.cc（模块 自定义算子；类别 source；类型 core-logic；符号 MTPSaveFirstTokenWithTopK）：新增的核心 C++ 算子，负责在 prefill 节点保存带 logprob 的首 token 信息，通过 SysV 消息队列传递数据，是实现功能的关键组件。
- fastdeploy/model_executor/pre_and_post_process.py（模块 模型执行器；类别 source；类型 core-logic；符号 save_output_specualate）：重构的输出保存函数，新增参数和逻辑分支以支持 MTP prefill 节点的 logprob 处理，是 Python 端的核心调度点。
- custom_ops/gpu_ops/speculate_decoding/speculate_logprob_msg.h（模块 自定义算子；类别 source；类型 data-contract；符号 msgdata, batch_msgdata）：新增的共享头文件，抽取了消息结构体和宏定义，供多个算子复用，提高了代码可维护性。
- fastdeploy/spec_decode/mtp.py（模块 投机解码；类别 source；类型 refactor；符号 _post_process）：修改 MTP proposer 的后处理逻辑，将 mtp_save_first_token 调用迁移到 pre_and_post_process.py，仅保留 XPU 平台调用，统一了 GPU 平台路径。
- fastdeploy/worker/gpu_model_runner.py（模块 工作器；类别 source；类型 entrypoint；符号 _save_model_output）：调整模型运行器的输出保存调用，传入 proposer 输入和 rank 信息，以支持 MTP prefill 首 token 保存，但存在兼容性风险。

关键符号：MTPSaveFirstTokenWithTopK, save_output_specualate, _post_process, _save_model_output

关键源码片段

custom_ops/gpu_ops/speculate_decoding/draft_model/mtp_save_first_token_with_topk.cc

新增的核心 C++ 算子，负责在 prefill 节点保存带 logprob 的首 token 信息，通过 SysV 消息队列传递数据，是实现功能的关键组件。

```
void MTPSaveFirstTokenWithTopK(const paddle::Tensor& sampled_token_ids,
                               const paddle::Tensor& logprob_token_ids,
                               const paddle::Tensor& logprob_scores,
                               const paddle::Tensor& logprob_ranks,
                               const paddle::Tensor& token_num_per_batch,
                               const paddle::Tensor& cu_batch_token_offset,
                               const paddle::Tensor& not_need_stop,
                               const paddle::Tensor& seq_lens_decoder,
                               const paddle::Tensor& prompt_lens,
                               const paddle::Tensor& preempted_idx,
                               int message_flag, // Target: 3, Draft: 4
                               int64_t rank_id,
                               bool save_each_rank) {
    if (!save_each_rank && rank_id > 0) {
        return; // 仅当save_each_rank为true或rank_id为0时执行，控制消息发送范围
    }

    int max_draft_tokens = sampled_token_ids.shape()[1];
    int bsz = token_num_per_batch.shape()[0];

    // 将输入张量拷贝到CPU，以便进行消息队列操作
    auto sampled_token_ids_cpu = sampled_token_ids.copy_to(paddle::CPUPlace(), false);
    auto logprob_token_ids_cpu = logprob_token_ids.copy_to(paddle::CPUPlace(), false);
    auto logprob_scores_cpu = logprob_scores.copy_to(paddle::CPUPlace(), false);
    auto logprob_ranks_cpu = logprob_ranks.copy_to(paddle::CPUPlace(), false);
    auto token_num_per_batch_cpu = token_num_per_batch.copy_to(paddle::CPUPlace(), false);
    auto cu_batch_token_offset_cpu = cu_batch_token_offset.copy_to(paddle::CPUPlace(), false);
    auto seq_lens_decoder_cpu = seq_lens_decoder.copy_to(paddle::CPUPlace(), true);
    auto prompt_lens_cpu = prompt_lens.copy_to(paddle::CPUPlace(), true);

    // 获取数据指针
    int64_t* sampled_token_ids_data = sampled_token_ids_cpu.data<int64_t>();
    int64_t* logprob_token_ids_data = logprob_token_ids_cpu.data<int64_t>();
    float* logprob_scores_data = logprob_scores_cpu.data<float>();
    int64_t* logprob_ranks_data = logprob_ranks_cpu.data<int64_t>();
    int* token_num_per_batch_data = token_num_per_batch_cpu.data<int>();
    int* cu_batch_token_offset_data = cu_batch_token_offset_cpu.data<int>();
    int* seq_lens_decoder_data = seq_lens_decoder_cpu.data<int>();
    int64_t* prompt_lens_data = prompt_lens_cpu.data<int64_t>();
    const int32_t* preempted_idx_data = preempted_idx.data<int32_t>();

    static struct msgdata msg_sed; // 使用共享头文件中定义的消息结构体
    // 后续逻辑：初始化消息队列，填充msg_sed结构，并发送消息
    // 注意：cur_token_num = token_num_per_batch_data[i] + 1，可能存在越界风险
}
```

fastdeploy/model_executor/pre_and_post_process.py

重构的输出保存函数，新增参数和逻辑分支以支持 MTP prefill 节点的 logprob 处理，是 Python 端的核心调度点。

```
def save_output_specualate(
    sampler_output: SamplerOutput,
    model_output: ModelOutputData,
    share_inputs: InputBatch,
    proposer_share_inputs: ProposerInputBatch, # 新增参数：proposer的输入数据
    local_rank: int, # 新增参数：本地rank
    tensor_parallel_rank: int, # 新增参数：张量并行rank
    save_each_rank: bool = False,
    is_mtp_prefill: bool = False, # 新增参数：标识是否为MTP prefill节点
):
    if is_mtp_prefill:
        if tensor_parallel_rank == 0: # 仅张量并行rank 0执行保存操作
            skip_chunk_prefill = bool(int(envs.ENABLE_V1_KVCACHE_SCHEDULER))
            if sampler_output.logprobs_tensors is None:
                # 无logprob时，恢复proposer输入并调用原始保存函数
                recover_proposer_share_inputs_map = recover_batch_index_for_output(
                    proposer_share_inputs,
                    proposer_share_inputs.index_to_batch_id,
                    proposer_share_inputs.enable_pd_reorder,
                    ["base_model_draft_tokens", "seq_lens_decoder", "prompt_lens", "step_idx"],
                )
                mtp_save_first_token(
                    recover_proposer_share_inputs_map["base_model_draft_tokens"],
                    proposer_share_inputs["not_need_stop"],
                    recover_proposer_share_inputs_map["seq_lens_decoder"],
                    recover_proposer_share_inputs_map["prompt_lens"],
                    recover_proposer_share_inputs_map["step_idx"],
                    local_rank,
                    save_each_rank,
                    skip_chunk_prefill,
                )
            else:
                # 有logprob时，恢复share_inputs和proposer输入，并调用带topk的保存函数
                recover_share_inputs_map = recover_batch_index_for_output(
                    share_inputs,
                    model_output.index_to_batch_id,
                    model_output.enable_pd_reorder,
                    ["sampled_token_ids", "accept_tokens_cpu", "accept_num_cpu",
                     "seq_lens_decoder_cpu", "prompt_lens_cpu", "last_preempted_idx"],
                )
                recover_batch_index_for_sampler_output(
                    sampler_output, model_output.index_to_batch_id, model_output.enable_pd_
                    reorder
                )
```

```
                recover_proposer_share_inputs_map = recover_batch_index_for_output(
                    proposer_share_inputs,
                    proposer_share_inputs.index_to_batch_id,
                    proposer_share_inputs.enable_pd_reorder,
                    ["base_model_draft_tokens"],
                )
                mtp_save_first_token_with_topk(
                    recover_proposer_share_inputs_map["base_model_draft_tokens"],
                    sampler_output.logprobs_tensors.logprob_token_ids,
                    sampler_output.logprobs_tensors.logprobs,
                    sampler_output.logprobs_tensors.selected_token_ranks,
                    recover_share_inputs_map["accept_num_cpu"], # 作为token_num_per_batch传入
                    sampler_output.cu_batch_token_offset,
                    model_output.not_need_stop,
                    recover_share_inputs_map["seq_lens_decoder_cpu"],
                    recover_share_inputs_map["prompt_lens_cpu"],
                    recover_share_inputs_map["last_preempted_idx"],
                    3, # message_flag: Target类型
                    model_output.mp_rank,
                    save_each_rank,
                )
            else:
                # 非prefill分支，保持原有逻辑但使用proposer_share_inputs作为数据源
                # 注意：此处可能存在数据一致性问题，需验证batch排列顺序
                pass
```

评论区精华

review 中主要讨论了以下问题：

- 兼容性风险：PaddlePaddle-bot 指出 gpu_model_runner.py 中条件从 self.speculative_decoding 收窄为 self.speculative_decoding and self.spec_method == SpecMethod.MTP，可能导致非 MTP 投机方法（如 NAIVE、NGRAM、SUFFIX）不再走 save_output_specualate，丢失专用输出处理。建议保持对非 MTP 方法的原有处理。
- 数据一致性疑问：PaddlePaddle-bot 质疑 pre_and_post_process.py 中非 prefill 分支使用 proposer_share_inputs 作为数据源，但 index_to_batch_id 来自 model_output，两者的 batch 排列顺序可能不一致，需要确认。
- 代码重复与命名冲突：
  - PaddlePaddle-bot 建议将 mtp_save_first_token_with_topk.cc 中的环境变量解析和消息队列初始化逻辑抽取为公共 helper 函数，减少重复代码。
  - PaddlePaddle-bot 指出 speculate_logprob_msg.h 中的宏命名（如 K）过于通用，存在命名冲突风险，建议添加项目前缀（如 SPECULATE_TOPK）。
- 潜在 bug 与语义冲突：
  - PaddlePaddle-bot 和 Copilot 均指出 gpu_model_runner.py 中无条件传入 self.proposer.model_inputs，在非 MTP 方法（如 NAIVE、NGRAM、SUFFIX）或 proposer 为 None 时会触发 AttributeError，建议延迟求值或条件传参。

  - Copilot 详细分析了 mtp_save_first_token_with_topk.cc 中 cur_token_num 计算（token_num_per_batch_data[i] + 1）与 accept_num_cpu 语义的冲突，可能导致越界写入或下游解析不一致，建议统一语义。
- 测试缺失：Copilot 建议补充单测以覆盖新算子路径，减少回归风险。
- 未使用参数：PaddlePaddle-bot 提到 save_each_rank 参数在算子中声明但未使用，与原始逻辑不同，需确认设计意图。 决策结论：大部分问题被标记为建议或疑问，未在 review 中看到明确修复；PR 最终被合并，但遗留了兼容性、数据一致性和测试覆盖等风险。
- 非 MTP 投机方法的兼容性风险 (correctness): 建议保持对非 MTP 方法的原有处理，但未在 review 中看到明确修复。
- 数据一致性与参数传递疑问 (correctness): 需要确认 proposer_share_inputs 中是否包含所需 key 且映射关系正确，但未在 review 中看到验证。
- 代码重复与命名冲突 (design): 建议添加项目前缀或抽取公共函数，但未在 review 中看到修改。
- 潜在 bug 与语义冲突 (correctness): 建议统一语义，明确 token_num_per_batch 的期望值，但未在 review 中看到调整。
- 测试缺失与回归风险 (testing): 未添加单元测试，存在较高回归风险。

风险与影响

- 风险：1. 兼容性风险：gpu_model_runner.py 中条件收窄可能导致非 MTP 投机方法（NAIVE、NGRAM、SUFFIX）的输出保存逻辑回退到 save_output_normal，丢失 accept_tokens 等投机解码专用处理，影响这些方法的正确性。 2. 运行时崩溃风险：gpu_model_runner.py 中无条件访问 self.proposer.model_inputs，在 proposer 为 None（如 SpecMethod.NAIVE）或非 MTP 方法时抛出 AttributeError，导致服务崩溃。 3. 数据越界与语义冲突风险：mtp_save_first_token_with_topk.cc 中 cur_token_num 计算（token_num_per_batch_data[i] + 1）可能超过 MAX_DRAFT_TOKEN_NUM（当前为 6），导致数组越界写入；且与 accept_num_cpu 语义（接受的 token 总数）冲突，可能引发下游解析错误。 4. 测试覆盖不足风险：根据 review 评论和 codecov 报告，新增算子缺少单元测试，patch 覆盖率仅 26.32%，回归风险较高。 5. 平台兼容性风险：pre_and_post_process.py 顶层无条件导入 GPU ops，可能影响非 GPU 平台（如 XPU）的导入或执行。
- 影响：1. 用户影响：使 PD 分离部署下的 MTP 投机解码能够返回 prefill 节点的首 token logprob，提升调试和监控能力，但若未解决兼容性问题，可能影响非 MTP 投机方法的正常使用。 2. 系统影响：
  - 新增算子和重构可能引入性能开销（如数据拷贝、消息队列通信），但属于必要功能扩展。
  - 统一 GPU 平台调用路径到 pre_and_post_process.py，简化了代码结构，便于维护。
    3. 团队影响：开发者需注意新引入的参数和逻辑分支，特别是在使用非 MTP 投机方法或混合部署场景时，需验证兼容性。
- 风险标记：非 MTP 方法兼容性风险，运行时崩溃风险，数据越界风险，测试覆盖不足

关联脉络

- PR #7412 [PD Disaggregation] Enable PD deployment without Router: 同属 PD 分离部署相关功能，可能共享配置或测试逻辑，本 PR 的 logprob 支持是该架构下的扩展。
- PR #7438 [BugFix] Fix real token exceeding max_batched_tokens limit: 同属投机解码（Speculative Decoding）相关修复，涉及调度器和资源管理，可能与本 PR 的 logprob 处理有交互。
- PR #7180 [XPU] Unify Spec and non-spec branch.(#6947): 同属投机解码平台统一工作，本 PR 中 mtp.py 保留了 XPU 平台的 mtp_save_first_token 调用，与此 PR 相关。