

# PR #28723 完整报告

sgl-project/sglang

[Intel XPU] Enable fused\_moe\_triton tuning on XPU and add tuned DeepSeek-OCR-2 configs

合并时间: 2026-09-16 10:20

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/28723>

## 执行摘要

- 一句话: 让 fused\_moe\_triton 调优器支持 XPU, 并新增 Arc Pro B60 的 DeepSeek-OCR-2 MoE 调优配置
- 推荐动作: 值得精读, 尤其是正在做多后端 Triton kernel 调优或维护 MoE 相关 benchmark 工具的工程师。三个值得关注的设计决策: 一是用 get\_device\_module 一次性收敛同步、事件、图捕获与播种, 避免在 kernel 里写 XPU 专用分支, 这是「最小持续成本」路线的具体落地; 二是 clear\_b\_tma\_desc\_cache 揭示的隐藏耦合——描述符缓存与图捕获内存争用, 在长生命周期服务的语境下为什么是 no-op; 三是把 topk\_ids 层布局从硬编码改为从模型配置解析, 代价是引入了对模型元数据字段的隐式依赖。阅读时建议同时对照 Intel Triton issue #7306 / #7691 与 PR body 中的 workload 结果表, 并留意 tuning\_fused\_moe\_triton.py 与 \_sep 版本仍需保持行为一致, 以及 review 中 airMeng 提出的下一步方向 (KV cache store 等 memory-bound kernel)。

## 功能与动机

动机来自 PR body: 调优器 \_sep 版本是 CUDA 专用的, 图捕获、事件、播种和计时全部直连 torch.cuda.\*, 且 Ray worker 把每个 actor 都钉在 device 0 上, 导致 XPU 上无法完成有意义的 MoE 调优。作者希望按 Intel Triton 团队文档推荐的写法 (TensorDescriptor + autotune) 在 XPU 上把 fused\_moe\_triton 调到最优, 并为 Arc Pro B60 上的 DeepSeek-OCR-2 留下可直接复用的配置数据。过程还暴露并推动了两个上游 Triton issue: intel/intel-xpu-backend-for-triton#7306 (host 侧 TensorDescriptor 丢掉 2D-block-I/O 快路径) 与 #7691 (TensorDescriptor 形状特化导致每个 batch size 编译一次), 两者均已关闭。

## 实现拆解

1. 设备抽象 (tuning\_fused\_moe\_triton\_sep.py): 新增 get\_device / get\_device\_module / is\_xpu 导入与模块级 device\_module、\_is\_xpu; KernelWrapper.cuda\_graph\_wrapper 与 forward\_cost 中的 torch.cuda.synchronize、torch.cuda.Event、torch.cuda.CUDAGraph、torch.cuda.graph 全部替换为 device\_module.\*, XPU 分支改用 torch.xpu.XPUGraph 作为图类。BenchmarkWorker.\_\_init\_\_ 的 torch.set\_default\_device("cuda") 改为 get\_device(), manual\_seed\_all 同样走设备模块; benchmark / tune (Round 1 与 Round 2) / cmp\_configs 中的设备上下文条件由 is\_hip() 放宽为 \_is\_xpu or \_is\_hip, 使 XPU 也能正确切换 device。改动方式刻意对齐同目

录已有的 `tuning_fused_moe_triton.py`，保持两条工具路径一致。

2. Ray worker 绑定设备 (`tuning_fused_moe_triton_sep.py`、`pyproject_xpu.toml`) :  
`self.device_id` 由固定 0 改为 `0 if not ray.is_initialized() else int(ray.get_gpu_ids()[0])`。  
commit message 说明原因是 XPU 上 Ray 的可见设备掩码生效太晚，若不显式使用 Ray 分配的 id，每个 worker 都会跑在 device 0 上。配套在 `pyproject_xpu.toml` 新增可选  
`extra ray = ["ray[default]>=2.55.1", "dpctl"]`，注释指出 `dpctl` 是 Ray 注册 Intel GPU 资源的前提；该 `extra` 为 opt-in、不并入 all，因此不影响服务化安装。
3. 层布局改为配置驱动 (`tuning_fused_moe_triton_sep.py`、  
`benchmark/kernels/fused_moe_triton/common_utils.py`) : `load_topk_ids` 删掉硬编码的  
`num_layers = 61 / dense_layers = 3`，改为通过 `sglang.srt.runtime_context` 的  
`get_model()` / `get_parallel()` 拿到模型路径、tp/ep，再调用 `get_model_config`，用返回的  
`num_layers`、`dense_layers` 推导 MoE 层数。为此 `get_model_config` 把  
`DeepseekOCRForCausalLM` 加入 DeepSeek 家族分支，并在返回值中新增 `num_layers` (  
`getattr(config, "num_hidden_layers", 0)`) 与 `dense_layers` (`getattr(config, "first_k_dense_replace", 0)`)，注释说明只有 DeepSeek 家族会用 `dense` 层替换前若干 MoE 层)。  
4. 描述符缓存清理 (`python/sglang/kernels/ops/moe/fused_moe_triton_kernels.py`) : 在  
模块级 `_B_DESC_CACHE` 旁新增 `clear_b_tma_desc_cache()`，`benchmark_config` 进入  
时立即调用。原因是 tuner 每个 config 都会重新分配 `w1/w2`，缓存里的旧  
`TensorDescriptor` 只会钉住已死权重并让图捕获内存吃紧；服务化时权重与进程同生命周期，因此该清理在线上是 no-op，CUDA 服务路径未受影响。
5. 配置数据与测试配套 (`configs/*.json`、`test_fused_moe_common_utils.py`) : 新增  
`configs/triton_3_8_0/E=64,N=896,device_name=Intel(R)_Arc(TM)_Pro_B60_Graphics.j`  
`son` (上投影，17 个 batch size) 与同名 `_down.json` (下投影，额外带 `USE_TMA: true`)  
，按 shape 键控因此也适用于 DeepSeek-OCR v1。测试侧更新 Kimi-VL 断言以包含新增  
字段，并新增 `test_get_model_config_reports_deepseek_ocr_layer_layout` 覆盖  
`DeepSeek-OCR-2` 的 12 层 / 1 dense 层布局。第 4 个 commit 额外移除了对已废弃  
`get_global_server_args` 的使用。

关键文件：

- `benchmark/kernels/fused_moe_triton/tuning_fused_moe_triton_sep.py` (模块 调优工具  
；类别 source；类型 core-logic；符号 `KernelWrapper.cuda_graph_wrapper`，  
`KernelWrapper.forward_cost`，`load_topk_ids`，`benchmark_config`) : 核心改造文件：把  
CUDA 专用的调优器改为设备无关 (sync/ 事件 / 图捕获 / 播种走 `device_module`)，修正  
Ray worker 设备绑定，并把 `topk_ids` 层布局从硬编码 61/3 改为从模型配置解析。
- `benchmark/kernels/fused_moe_triton/common_utils.py` (模块 模型配置；类别 source；  
类型 core-logic；符号 `get_model_config`) : 为调优器提供层布局元数据：把  
`DeepseekOCRForCausalLM` 纳入 DeepSeek MoE 分支，并在返回字典中新增  
`num_layers / dense_layers`，直接决定 `load_topk_ids` 能否找到录制文件。
- `test/registered/unit/layers/moe/test_fused_moe_common_utils.py` (模块 单元测试；类  
别 test；类型 test-coverage；符号 `test_get_model_config_reports_deepseek_ocr_layer`  
`_layout`，`test_get_model_config_supports_kimi_vl`) : 回归修复的落点：更新 Kimi-VL 断

言以包含新增字段，并新增 DeepSeek-OCR 层布局覆盖，正是 review 中要求补的测试。

- python/sglang/kernels/ops/moe/fused\_moe\_triton\_kernels.py (模块 内核缓存; 类别 infra; 类型 infrastructure; 符号 clear\_b\_tma\_desc\_cache) : 新增 clear\_b\_tma\_desc\_cache, 用于在 tuner 每次换 config 时释放被缓存的 B TensorDescriptor, 是修掉「缓存钉住死权重、饿死图捕获」的关键小改动。
- python/sglang/srt/layers/moe/moe\_runner/triton\_utils/configs/triton\_3\_8\_0/E=64,N=896,device\_name=Intel(R)\_Arc(TM)\_Pro\_B60\_Graphics.json (模块 内核配置; 类别 config; 类型 configuration) : DeepSeek-OCR-2 上投影的 Arc Pro B60 调优结果, 17 个 batch size, 是性能收益的数据载体; 按 shape 键控, 也可服务 DeepSeek-OCR v1。
- python/sglang/srt/layers/moe/moe\_runner/triton\_utils/configs/triton\_3\_8\_0/E=64,N=896,device\_name=Intel(R)\_Arc(TM)\_Pro\_B60\_Graphics\_down.json (模块 内核配置; 类别 config; 类型 configuration) : 下投影配置, 唯一携带 USE\_TMA: true 的一份, 是 untouched 基线 (无 down config、无 TMA) 与调优结果差距的主要来源。
- python/pyproject\_xpu.toml (模块 依赖声明; 类别 config; 类型 configuration) : 为 XPU 声明 opt-in 的 ray extra 及 dpctl, 使调优器在 XPU 上能让 Ray 正确识别并分配 Intel GPU, 同时不并入 all、不影响服务化安装。

关键符号: clear\_b\_tma\_desc\_cache, test\_get\_model\_config\_reports\_deepseek\_ocr\_layer\_layout, load\_topk\_ids, benchmark\_config, KernelWrapper.cuda\_graph\_wrapper, KernelWrapper.forward\_cost, BenchmarkWorker.init, get\_model\_config

## 关键源码片段

### benchmark/kernels/fused\_moe\_triton/tuning\_fused\_moe\_triton\_sep.py

核心改造文件: 把 CUDA 专用的调优器改为设备无关 (sync/ 事件 / 图捕获 / 播种走 device\_module), 修正 Ray worker 设备绑定, 并把 topk\_ids 层布局从硬编码 61/3 改为从模型配置解析。

设备抽象与图捕获 (重写后的相对完整实现) :

```
# benchmark/kernels/fused_moe_triton/tuning_fused_moe_triton_sep.py
_is_hip = is_hip()
_is_xpu = is_xpu()
# 统一出口: CUDA/HIP 下是 torch.cuda, XPU 下是 torch.xpu
device_module = get_device_module()
```

```
class KernelWrapper:
```

```
    """把一个 MoE 调用序列包装成可重放的设备图, 并在图外计时。"""
```

```
    def __init__(self, moe_inputs, use_cuda_graph=True, inner_iter=10, **kwargs):
        self.func = invoke_fused_moe_kernel
        self.use_cuda_graph = use_cuda_graph
        self.moe_inputs = moe_inputs
        self.inner_iter = inner_iter
        self.kwags = kwargs
```

```

self.graph = self.cuda_graph_wrapper() if use_cuda_graph else None

def cuda_graph_wrapper(self):
    moe_input = self.moe_inputs[0]
    self.func(
        **self.kwargs,
        topk_ids=moe_input.topk_ids,
        sorted_token_ids=moe_input.sorted_token_ids,
        expert_ids=moe_input.expert_ids,
        num_tokens_post_padded=moe_input.num_tokens_post_padded,
    )
    # 原来是 torch.cuda.synchronize(), 现在按设备派发
    device_module.synchronize()

    # 把 inner_iter 次调用捕获进一张可重放的图; XPU 需要用 XPUGraph 类
    graph_cls = torch.xpu.XPUGraph if _is_xpu else torch.cuda.CUDAGraph
    graph = graph_cls()
    with device_module.graph(graph):
        for k in range(self.inner_iter):
            moe_input = self.moe_inputs[k]
            self.func(
                **self.kwargs,
                topk_ids=moe_input.topk_ids,
                sorted_token_ids=moe_input.sorted_token_ids,
                expert_ids=moe_input.expert_ids,
                num_tokens_post_padded=moe_input.num_tokens_post_padded,
            )
    device_module.synchronize()

    # 预热 5 次, 把首次编译与首次执行排除在计时区间之外
    for _ in range(5):
        graph.replay()
    device_module.synchronize()
    return graph

def forward_cost(self, try_cnt=2):
    time_cost = float("inf")
    for _ in range(try_cnt):
        # 事件同样走设备抽象, 否则 XPU 上会创建 CUDA 事件
        start_event = device_module.Event(enable_timing=True)
        end_event = device_module.Event(enable_timing=True)
        start_event.record()
        if self.use_cuda_graph:
            self.graph.replay()
        else:
            for k in range(self.inner_iter):
                moe_input = self.moe_inputs[k]
                self.func(
                    **self.kwargs,

```

```

        topk_ids=moe_input.topk_ids,
        sorted_token_ids=moe_input.sorted_token_ids,
        expert_ids=moe_input.expert_ids,
        num_tokens_post_padded=moe_input.num_tokens_post_padded,
    )
    end_event.record()
    device_module.synchronize()
    time_cost = min(time_cost, start_event.elapsed_time(end_event))
    return time_cost

```

层布局解析、描述符缓存清理与 Ray 设备绑定：

```
def load_topk_ids(topk_ids_dir, i: int):
```

```
    """按 MoE 层索引读取录制好的 topk_ids。
```

层布局不再硬编码为 DeepSeek-V3 的 61 层 / 3 dense 层，而是从模型配置解析，因此 DeepSeek-OCR-2（12 层、首 1 层为 dense）才能取到正确文件。

```
    """
```

```

    model_config = get_model_config(
        get_model().model_path,
        tp_size=get_parallel().tp_size,
        ep_size=get_parallel().ep_size,
    )
    num_layers = model_config["num_layers"]
    dense_layers = model_config["dense_layers"]
    moe_layers = num_layers - dense_layers
    return torch.load(
        f"{topk_ids_dir}/topk_ids_layer{i % moe_layers + dense_layers}_idx{i // moe_layers}.pt"
    )

```

```
def benchmark_config(config: BenchmarkConfig, num_tokens: int, ...) -> float:
```

```
    ncu_enable = os.getenv("NCU_ENABLE", "0") == "1"
```

```
    if ncu_enable:
```

```
        num_iters = 1
```

```
    # 每个 config 都会重新分配 w1/w2，缓存里的旧 TensorDescriptor 只会钉住死权重，
```

```
    # 并让图捕获内存吃紧；服务化时权重与进程同生命周期，所以此清理在线上是 no-op。
```

```
    clear_b_tma_desc_cache()
```

```
    init_dtype = torch.float16 if use_fp8_w8a8 else dtype
```

```
    ...
```

```
class BenchmarkWorker:
```

```
    def __init__(self, seed: int, server_args: ServerArgs) -> None:
```

```
        torch.set_default_device(get_device())
```

```
        device_module.manual_seed_all(0)
```

```
        self.seed = seed
```

```
        # XPU 上 ray 的可见设备掩码生效太晚，必须显式使用 ray 分配的 device id，
```

```
        # 否则每个 actor 都落在 device 0 上，多卡调优结果全部作废。
```

```
        self.device_id = 0 if not ray.is_initialized() else int(ray.get_gpu_ids()[0])
```

```
set_global_server_args_for_scheduler(server_args)
```

## benchmark/kernels/fused\_moe\_triton/common\_utils.py

为调优器提供层布局元数据：把 DeepseekOCRForCausalLM 纳入 DeepSeek MoE 分支，并在返回字典中新增 num\_layers / dense\_layers，直接决定 load\_topk\_ids 能否找到录制文件。

get\_model\_config 的关键分支与返回值（重写后的完整尾部逻辑）：

```
# benchmark/kernels/fused_moe_triton/common_utils.py
elif architecture in [
    "DeepseekV2ForCausalLM",
    "DeepseekV3ForCausalLM",
    "DeepseekV32ForCausalLM",
    "DeepseekV4ForCausalLM",
    # 新增: DeepSeek-OCR-2 与 DeepSeek 家族共用同一套 MoE 字段布局
    "DeepseekOCRForCausalLM",
    "Glm4MoeForCausalLM",
    "Glm4MoeLiteForCausalLM",
    "GlmMoeDsaForCausalLM",
    "KimiVLForConditionalGeneration",
    "MistralLarge3ForCausalLM",
]:
    E = (config.n_routed_experts // ep_size) + (
        0
        if disable_shared_experts_fusion
        or architecture
        not in [
            "DeepseekV3ForCausalLM",
            "DeepseekV32ForCausalLM",
            "Glm4MoeForCausalLM",
            "Glm4MoeLiteForCausalLM",
            "GlmMoeDsaForCausalLM",
            "MistralLarge3ForCausalLM",
        ]
        else 1
    )
    topk = config.num_experts_per_tok + (
        0 if disable_shared_experts_fusion or topk_ids_dir is None else 1
    )
    intermediate_size = config.moe_intermediate_size

    shard_intermediate_size = calculate_shard_intermediate_size(
        intermediate_size, tp_size, ep_size
    )

    # text_config 可能不带 torch_dtype, 这里回退到 bf16
    torch_dtype = getattr(config, "torch_dtype", None) or torch.bfloat16
    num_layers = getattr(config, "num_hidden_layers", 0)
    # 只有 DeepSeek 家族会用 dense 层替换前若干层 MoE 层;
```

```

# 取不到时默认 0，调用方需自行承担 moe_layers 为 0 的后果。
dense_layers = getattr(config, "first_k_dense_replace", 0)

return {
    "num_experts": E,
    "topk": topk,
    "hidden_size": hidden_size,
    "shard_intermediate_size": shard_intermediate_size,
    "dtype": torch_dtype,
    "block_shape": block_shape,
    "architecture": architecture,
    # 下面两个字段是本次新增，被 tuning_fused_moe_triton_sep.load_topk_ids 消费
    "num_layers": num_layers,
    "dense_layers": dense_layers,
}

```

## test/registered/unit/layers/moe/test\_fused\_moe\_common\_utils.py

回归修复的落点：更新 Kimi-VL 断言以包含新增字段，并新增 DeepSeek-OCR 层布局覆盖，正是 review 中要求补的测试。

新增的 DeepSeek-OCR 层布局测试（完整用例）：

```

# test/registered/unit/layers/moe/test_fused_moe_common_utils.py
def test_get_model_config_reports_deepseek_ocr_layer_layout():
    """load_topk_ids 按 MoE 层索引取录制文件，层数由 num_layers - dense_layers 推出，
    任一字段为 0 都会让 tuner 读到错误（或读不到）的录制数据。

    这个用例同时锁定了「层数字段来自 text_config 而非外层 config」这一事实。
    """
    common_utils = _load_common_utils()
    text_config = SimpleNamespace(
        hidden_size=1280,
        n_routed_experts=64,
        num_experts_per_tok=6,
        moe_intermediate_size=896,
        num_hidden_layers=12, # 12 层中首 1 层为 dense
        first_k_dense_replace=1,
        torch_dtype=torch.bfloat16,
    )
    model_config = SimpleNamespace(
        architectures=["DeepseekOCRForCausalLM"],
        text_config=text_config,
        get_text_config=lambda: text_config,
    )

    with patch.object(common_utils, "get_config", return_value=model_config):
        tuned_config = common_utils.get_model_config(
            "deepseek-ai/DeepSeek-OCR-2", tp_size=1, ep_size=1
        )

```

```

assert tuned_config == {
    "num_experts": 64,
    "topk": 6,
    "hidden_size": 1280,
    "shard_intermediate_size": 1792,
    "dtype": torch.bfloat16,
    "block_shape": None,
    "architecture": "DeepseekOCRForCausalLM",
    "num_layers": 12,
    "dense_layers": 1,
}

```

## 评论区精华

- 路线投入之争 (design) : mingfeima 明确表态「our strategy is to keep triton path functional not necessarily performant」, 认为 XPU 上 Triton 只是 nice to have, 应保持最小投入; airMeng 补充「I am okay for tuning triton configs since it can improve performance with the least effort. But DeepSeek-OCR is not quite critical. You can switch to some other memory-bound triton kernels like kv cache store」, 即认可调 config 的性价比但质疑所选场景的重要性。作者引用 Intel Triton 官方「Improving performance of your Triton source code」指南回应, 并强调持续成本接近零: 不改 MoE kernel 源码、无 XPU 专用分支、tuner 改动是设备无关的 bug 修复。
- CI 回归 (testing) : mingfeima 以 CHANGES\_REQUESTED 指出 benchmark/kernels/fused\_moe\_triton/common\_utils.py 新增 num\_layers / dense\_layers 后未同步 test\_fused\_moe\_common\_utils.py, Kimi-VL 期望字典缺两个字段导致 base-a-test-cpu 失败, 并要求「Need update the existing assertion and add coverage for the DeepSeek-OCR layer layout used by load\_topk\_ids」。作者在 3a37051cb1 修复并补齐 DeepSeek-OCR 覆盖; mingfeima 仍留下「Please do self check next time. I am super busy right now...」。
- 搜索空间截断 (correctness) : gemini-code-assist 对 tuning\_fused\_moe\_triton.py 中 search\_space = search\_space[:4] 打出 high priority, 要求合并前移除。最终 configs 覆盖 18 个 batch size 的全量 sweep, 说明截断未进入最终版本。
- 设备绑定与缓存副作用: commit message 与 PR body 记录了 XPU 上 Ray mask 生效过晚导致所有 worker 落在 device 0, 以及 B 描述符缓存钉住死权重、饿死图捕获两个非显而易见的坑, 前者用 `ray.get_gpu_ids()` 修复, 后者用 `clear_b_tma_desc_cache()` 修复。
  - XPU 上 Triton 路线的投入策略与场景选择 (design): 方案被接受 (airMeng、mingfeima、polisettyvarma 均给出 approve), 但「DeepSeek-OCR 不是关键场景」的意见未被解决, 属于后续方向建议而非阻塞项。
  - get\_model\_config 新增字段导致 base-a-test-cpu CI 回归 (testing): 回归已在 3a37051cb1 修复, Kimi-VL 断言与 DeepSeek-OCR 层布局用例同时补齐; 流程层面留下「提交前先自检 CI」的要求。
  - 调优器搜索空间曾被临时截断为前 4 个配置 (correctness): 最终提交的配置覆盖 18 个 batch size 的全量 sweep, 截断未保留在最终版本; 该文件也未出现在最终变更文件列表中。

- XPU 上 Ray worker 的设备绑定与 dpctl 依赖 (correctness): self.device\_id 改为从 ray.get\_gpu\_ids() 推导, 并在 pyproject\_xpu.toml 新增 opt-in 的 ray extra ( ray[default]>=2.55.1 与 dpctl ), 不并入 all 以免影响服务化安装。
- B TensorDescriptor 缓存与图捕获的内存争用 (performance): 新增 clear\_b\_tma\_desc\_cache() 并在 benchmark\_config 入口调用, 服务路径行为不变。

## 风险与影响

- 风险:

1. 层数解析的零值陷阱: load\_topk\_ids 依赖 model\_config["num\_layers"] - dense\_layers, 而 get\_model\_config 用 getattr(..., 0) 取值。若某模型配置缺 num\_hidden\_layers, moe\_layers 为 0, i % moe\_layers 会直接抛 ZeroDivisionError ; 新增测试的 docstring 已点明这一失败模式, 但代码本身未做防御, 属于调优工具对模型元数据的隐式契约。
2. 配置按 device\_name 字符串键控: 新增配置文件名嵌入 Intel(R)\_Arc(TM)\_Pro\_B60\_Graphics, 设备命名变化 (驱动版本、B580 等兄弟型号、容器内改名) 会导致查找未命中并回退 get\_default\_config(), 最坏结果是性能退回基线而非报错; 同时 USE\_TMA: true 只出现在 \_down.json, 两侧配置字段并不对称。
3. 精度证据强度有限: PR body 自述 olmOCR-bench 总量 85.6% 对 85.8%, 但两臂并非同一测试集 (old\_scans 496 vs 472 个可评分样本), 作者也承认差异量级约 0.4% 以内, 只能读作「噪声内无回归」, 不能作为严格零回归证明; bf16 归约顺序确实随 BLOCK\_SIZE\_M / BLOCK\_SIZE\_K 改变。
4. 全局可变状态的导出: clear\_b\_tma\_desc\_cache 放在了 python/sglang/kernels/ops/moe/fused\_moe\_triton\_kernels.py, 是会被服务化代码导入的模块。当前唯一调用点在 benchmark 内, 但接口无访问约束, 若被误用到服务路径会清空 B 描述符缓存并带来重建开销。
5. CI 与可复现性: 本 PR 曾打断共享 base-a-test-cpu job, 修复虽已合入, 但 PR body 中的最新 PR Test (Base) 与 Extra 仍显示失败 / 未通过标记, 落地前需要确认是否还有残余失败; 此外 --configs 模式一次只对一个投影生效, 复现一组数据需要三次调用人工拼装 (PR body 已给出脚本), 测量结果依赖手工步骤, 误差控制较脆弱。
6. 安装路径未文档化: tuner 需要 ray 与 dpctl, 但只作为 opt-in extra 存在且不并入 all, 使用者在 XPU 上跑 tuner 时需自行安装 sglang[ray], PR 未附使用说明。- 影响: 对用户: 在 Intel Arc Pro B60 上以 --moe-runner-backend triton 跑 DeepSeek-OCR-2 的用户可拿到约 +13% 输出吞吐、约 +12.8%~13.6% 的 TPOT/ITL 改善 (eager 与 XPU graph 两种模式结论一致), 代价是新增两份 device 级配置; 不指定 triton 后端时 moe\_runner\_backend 会解析为 auto, 配置查找不会被触达, 收益为零。对系统: CUDA/HIP 服务化路径完全未改, clear\_b\_tma\_desc\_cache 在线上为 no-op; 配置目录新增按设备名键控的文件, 对其他设备无影响。对团队: 调优器从 CUDA 专用变为设备无关工具, 后续 AMD/NPU 调优可以直接复用同一套 device 抽象与 ray 绑定逻辑; 同时暴露出「改了 benchmark 工具的公共返回契约却没有同步测试」的协作问题, 维护者明确要求提交前自检 CI。- 风险标记: 调优工具隐式依赖模型元数据, 配置按 device\_name 字符串键控, 精度对比非同一测试集, 曾打断 base-a-test-cpu CI, 全局缓

存清理接口无访问约束，ray extra 安装路径未文档化

## 关联脉络

- PR #34712 [Fix] Spawn, don't fork, the benchmark server process: 同属 XPU 平台上的 benchmark 基础设施修复 (python/sglang/benchmark/endpoint.py)，与本 PR 一样解决「工具链在 XPU 上跑不通 / 跑不对」的问题，可对照看 XPU 工具链成熟度。
- PR #32618 [CPU] Add fp8\_per\_tensor\_scaled\_mm\_cpu kernel: 同为 Intel 平台 kernel 性能工作 (CPU GEMM 内核 + 量化路径)，与本 PR 的平台化性能优化脉络一致，且都涉及对设备能力的条件判断。
- PR #39631 [AMD] Prefer HIP Top-K for GLM-5.x on ROCm: 同为「按设备 / 平台选择内核实现或配置」的改动 (arg\_groups/model\_hook.py 设备门控)，可对照本 PR 的 device\_module/config 键控思路，观察多后端 kernel 选择策略的演进。