PR #33465 完整报告

sgl-project/sglang

[Kimi-K3][NPU] Support Kimi-K3 on NPU

合并时间：2026-08-12 21:20

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33465

执行摘要

- 一句话：Kimi-K3 全量支持 Ascend NPU，含 DSPARK 与 W4A8 量化
- 推荐动作：值得精读。核心看点：① kernel_dispatcher.extend_kernel 注入模式，让共享 KDA 后端零改动接入 Ascend 算子；② 数值等价 fallback 的工程化（显式开关 + unfused 路径，而非硬编码 if False）；③ ModelSlim 前缀解析把 checkpoint 层级差异收敛在量化边界；④ DSPARK 图折叠与 eager 采样的精度权衡。若要在此基础上维护 NPU 路径，建议优先补自动化回归测试并把硬编码常量配置化。

功能与动机

让 Kimi-K3 首次可在昇腾 NPU 上完整部署。PR body 说明这是在 upstream main 与 #32541 GPU 模型集成之上叠加 NPU 支持，原则是 'keeps shared/GPU behavior intact where possible and dispatches NPU-specific implementations through the Ascend backend'；算子层面把 Ascend 专属 Triton kernel 提取到 sgl-kernel-npu#658，避免仓库内平台分叉。精度上，MLA 的 torch.ops.npu.batch_matmul_transpose 与 fused split+RMSNorm 在 Ascend 上与参考实现数值不一致，因此需要 fallback 到 torch_npu.npu_transpose_batchmatmul 与 unfused 路径，这些数值问题只能通过 NPU 实机验证，也是本 PR 引入大量精度开关的原因。

实现拆解

变更入口：以显卡侧 #32541 的 Kimi-K3 集成为底座，通过 Ascend backend 分发 NPU 专属实现；NPU 算子整体外置到 sgl-kernel-npu#658，仓库内只保留调度与适配逻辑。

实现步骤：

1. KDA 注意力后端抽象（python/sglang/srt/hardware_backend/npu/attention/ascend_kda_backend.py，新增 634 行）：AscendKDAAttnBackend 继承共享的 KDAAttnBackend，仅覆盖 Ascend 的 cache 布局与算子调用；_AscendKDAExtendKernel.extend 用 sgl-kernel-npu 的 FLA 算子完成 prefill 分解（l2norm → chunk_local_cumsum → scaled_dot_kkt → solve_tril → recompute_w_u → chunk_gated_delta_rule → chunk_gla），decode 走 causal_conv1d_update_npu + packed_decode，target verify 复用 kda_target_verify_npu。kernel_dispatcher.extend_kernel 注入点让共享后端控制流不变。

2. MLA/KDA 数值等价 fallback：python/sglang/srt/hardware_backend/npu/modules/deepseek_v2_attention_mla_npu.py 为 Kimi-K3 关闭 fused split+RMSNorm（_disable_npu_fused_split_qk_norm），并以 torch_npu.npu_transpose_batchmatmul 替换数值发散的 torch.ops.npu.batch_matmul_transpose；只在激活该开关的模型上生效。
3. MoE 与模型布局适配：python/sglang/srt/models/kimi_k3.py 增加 SGLANG_K3_SHARED_EXPERTS_ATTN_TP / SGLANG_K3_DENSE_MLP_ATTN_TP，NPU 启动脚本用 attention-TP 分片 shared experts 与 dense MLP，_forward_shared_experts 负责 gather/reduce-scatter；moe_runner/ascend.py 在 activation == situ 时接入新增的 NPUSitu；modelslim.py 新增 _quant_prefix_candidates / _resolve_quant_prefix，抹平 mlp ↔ block_sparse_moe 与 language_model 前缀差异。
4. 推测解码配套：arg_groups/speculative_hook.py 放开 DSpark 对 NPU 的限制并允许 Ascend FuseEP；dspark_draft.py / dspark_worker_v2.py 引入 SGLANG_DSPARK_FOLDED_PROPOSAL / SGLANG_DSPARK_STACKED_CTX_KV / SGLANG_DSPARK_EMBED_IN_GRAPH 等 env，NPU 上贪心采样走图内 tail hook、随机采样保持 eager；dflash_utils.py 为 DFlash 验证增加 NPU 原生 top-k/top-p 重归一化并保留 torch 兜底。
5. 显存与部署配套：memory_pool_npu.py 增加 KDA conv state 的 is_kda 布局、set_kv_buffer_prefix_valid 的 NPU Triton store、FIA scatter 的 3-D view workaround；environ.py 注册全部新 env；新增 scripts/playground/launch_kimi_k3_npu_4node.sh 四节点启动脚本（TP64/DP4、W4A8、DeepEP、DSPARK）。

测试与 CI：本 PR 未新增单测文件；body 报告 GSM8K 94.6%、GPQA-Diamond 94.44% 与固定增量生成验证，static checks 通过；GPU CI（Base）通过，PR Test Extra 失败。运行强依赖 sgl_kernel_npu 新算子，需配套 kernel 包版本。

关键文件：

- python/sglang/srt/hardware_backend/npu/attention/ascend_kda_backend.py（模块 KDA 后端；类别 source；类型 dependency-wiring；符号 _AscendKDAExtendKernel, extend, AscendKDAAttnBackend, init）：新增的 Ascend KDA 注意力后端，是 NPU 执行 Kimi-K3 的核心入口：prefill 分解、decode、target verify 全部由此收口。
- python/sglang/srt/layers/quantization/modelslim/modelslim.py（模块 量化配置；类别 source；类型 data-contract；符号 _quant_prefix_candidates, _resolve_quant_prefix）：ModelSlim 量名前缀解析，解决 Kimi-K3 模型内部 mlp 层级与 checkpoint 中 block_sparse_moe 层级不一致的问题。
- python/sglang/srt/models/kimi_k3.py（模块 模型代码；类别 source；类型 data-contract；符号 _forward_shared_experts）：NPU 布局兼容：shared experts 与 dense MLP 支持 attention-TP 分片、sigmoid routing 契约修正，是模型侧最大的适配点。
- python/sglang/srt/speculative/dflash_utils.py（模块 推测解码；类别 source；类型 dependency-wiring；符号 _dflash_npu_top_k_top_p_renorm_prob, _dflash_top_k_renorm_prob, _dflash_top_p_renorm_prob）：为 DFlash 验证补充 NPU 原生 top-k/top-p 重归一化，并保留 torch 兜底，保证非 NPU 路径不受影响。

- python/sglang/srt/hardware_backend/npu/memory_pool_npu.py（模块 内存池；类别 source；类型 dependency-wiring；符号 set_kv_buffer_prefix_valid）：NPU 显存池适配：KDA conv state 布局、prefix KV cache 的 Triton store 以及 FIA scatter 的 3-D view workaround。
- python/sglang/srt/hardware_backend/npu/moe/activation.py（模块 激活函数；类别 source；类型 core-logic；符号 NPUSitu, init, _apply_activation）：新增 NPUSitu 激活，Kimi-K3 共享专家的 SiTU 激活与 INT8 重量化由此接入 DeepEP/Ascend runner。
- python/sglang/srt/hardware_backend/npu/modules/deepseek_v2_attention_mla_npu.py（模块 MLA 注意力；类别 source；类型 core-logic）：MLA 精度 fallback：Kimi-K3 关闭 fused split+RMSNorm，改用数值一致的 torch_npu.npu_transpose_batchmatmul。
- python/sglang/srt/arg_groups/speculative_hook.py（模块 启动校验；类别 source；类型 core-logic）：放开 DSpark 仅 CUDA 的限制并允许 NPU + FuseEP 组合，是 DSPARK 在 NPU 上可启动的开关。

关键符号：_AscendKDAExtendKernel.extend, AscendKDAAttnBackend.forward_decode, AscendKDAAttnBackend.forward_extend, ModelSlimConfig._quant_prefix_candidates, ModelSlimConfig._resolve_quant_prefix, KimiK3MoE._forward_shared_experts, _dflash_npu_top_k_top_p_renorm_prob, NPUMHATokenToKVPool.set_kv_buffer_prefix_valid, NPUSitu._apply_activation, _handle_dspark

关键源码片段

python/sglang/srt/hardware_backend/npu/attention/ascend_kda_backend.py

新增的 Ascend KDA 注意力后端，是 NPU 执行 Kimi-K3 的核心入口：prefill 分解、decode、target verify 全部由此收口。

```
class _AscendKDAExtendKernel:
    """Ascend 专属 KDA prefill 分解，底层算子来自 sgl-kernel-npu。

    共享 KDAAttnBackend 的 control flow 保持不变，这里只做布局与算子差异。
    """

    def extend(
        self,
        q, k, v, g, beta,
        *,
        ssm_states, cache_indices, query_start_loc,
        return_intermediate_states=False,
        **kwargs,
    ):
        # 分块大小固定为 64：prefill 按 chunk 递推，chunk 内做三角求解，
        # chunk 间用 delta rule 更新 SSM 状态。
        chunk_size = 64
        # q/k 先做 L2 归一化；NPU 算子要求输入为 contiguous
        q = l2norm_fwd(q.contiguous())
```

```
    k = l2norm_fwd(k.contiguous())
    v = v.contiguous()
    beta = beta.contiguous()
    chunk_indices = prepare_chunk_indices(query_start_loc, chunk_size)
    # gate 的 chunk 内累积和按 log2(e) 缩放，对应 kernel 内 exp2 语义
    g = chunk_local_cumsum(
        g.contiguous(), chunk_size=chunk_size, scale=_LOG2_E,
        cu_seqlens=query_start_loc, chunk_indices=chunk_indices,
    )
    # 1) 计算 chunk 内 QK^T（带 gated 衰减），再对三角矩阵做线性求解
    triangular, query_key = chunk_kda_scaled_dot_kkt_fwd(
        q=q, k=k, gk=g, beta=beta, scale=k.shape[-1] ** -0.5,
        cu_seqlens=query_start_loc, output_dtype=torch.float32,
    )
    triangular = solve_tril_npu(
        A=triangular, cu_seqlens=query_start_loc, output_dtype=k.dtype,
    )
    # 2) 由三角矩阵重新计算 w/u，得到 gated_k 供 delta rule 递推
    w, u, gated_k = recompute_w_u_fwd_npu(
        k=k, v=v, beta=beta, A=triangular, gk=g,
        cu_seqlens=query_start_loc, chunk_indices=chunk_indices,
    )
    del triangular
    # 3) delta rule 前向：更新 SSM 状态，产出 chunk 内新 values
    chunk_states, new_values = chunk_gated_delta_rule_fwd_h_npu(
        k=gated_k, w=w, u=u, gk=g,
        initial_state=ssm_states, initial_state_indices=cache_indices,
        cu_seqlens=query_start_loc, chunk_indices=chunk_indices,
        use_exp2=True,
    )
    del w, u, gated_k
    # 4) GLA 风格输出：直接复用 v 作为输出缓冲，避免额外分配
    out = chunk_gla_fwd_o_gk_npu(
        q=q, v=new_values, g=g, A=query_key, h=chunk_states,
        out=v, scale=k.shape[-1] ** -0.5,
        cu_seqlens=query_start_loc, chunk_size=chunk_size,
        chunk_indices=chunk_indices,
    )
    del query_key, new_values
    if return_intermediate_states:
        # 状态跟踪需要 (B, T, K, V) 布局的中间态，这里转置后返回
        return out, chunk_states.transpose(-1, -2).contiguous()
    return out
```

python/sglang/srt/speculative/dflash_utils.py

为 DFlash 验证补充 NPU 原生 top-k/top-p 重归一化，并保留 torch 兜底，保证非 NPU 路径不受影响。

```
def _dflash_npu_top_k_top_p_renorm_prob(probs, *, top_ks=None, top_ps=None):
```

```
    """NPU 上优先用 torch_npu 原生算子做 top-k/top-p 重归一化。

    返回 None 表示当前平台不可用，调用方会 fallback 到 torch 实现，
    从而保证 GPU/CPU 路径的行为完全不变。
    """
    if not is_npu() or probs.device.type != "npu":
        return None
    try:
        import torch_npu
    except ImportError:
        return None
    if not hasattr(torch_npu, "npu_top_k_top_p"):
        return None
    # npu_top_k_top_p 消费 logits，这里从概率取对数还原
    logits = probs.log()
    npu_top_ps = (
        top_ps.reshape(-1).to(device=probs.device, dtype=probs.dtype)
        if top_ps is not None else None
    )
    npu_top_ks = (
        top_ks.reshape(-1).to(device=probs.device, dtype=torch.int32)
        if top_ks is not None else None
    )
    # Ascend 算子的 top_k 上限为 1024，超限则放弃该路径
    if npu_top_ks is not None and not bool(
        torch.all((npu_top_ks >= 1) & (npu_top_ks <= 1024)).item()
    ):
        return None
    filtered_logits = torch_npu.npu_top_k_top_p(logits, npu_top_ps, npu_top_ks)
    return filtered_logits.softmax(dim=-1)


def _dflash_top_k_renorm_prob(probs, top_ks):
    # 优先复用 CUDA/ROCm 已编译的 sgl_kernel 或 triton 实现
    if top_k_renorm_prob is not None:
        return top_k_renorm_prob(probs, top_ks)
    # 其次走 NPU 原生算子
    npu_probs = _dflash_npu_top_k_top_p_renorm_prob(probs, top_ks=top_ks)
    if npu_probs is not None:
        return npu_probs
    # 兜底：torch topk + scatter 重建稠密概率，语义与 top-k-first 一致
    vocab_size = probs.shape[-1]
    top_ks = top_ks.reshape(-1).to(device=probs.device, dtype=torch.int64)
    top_ks = top_ks.clamp(min=1, max=vocab_size)
    max_top_k = int(top_ks.max().item())
    topk_probs, topk_indices = torch.topk(probs, k=max_top_k, dim=-1)
    ranks = torch.arange(max_top_k, device=probs.device)[None, :]
    topk_probs.masked_fill_(ranks >= top_ks[:, None], 0.0)
    topk_probs.div_(topk_probs.sum(dim=-1, keepdim=True))
```

```
    return torch.zeros_like(probs).scatter_(1, topk_indices, topk_probs)
```

评论区精华

- Hexq0210 在 kda_backend.py 第 347 行评论 'abstract a ascend_kda_backend'，主张把 Ascend 差异收口为独立后端；作者最终新增 ascend_kda_backend.py（+634 行）。
- MLA 精度上，Hexq0210 指出 'if Flase ...' 死代码，最终移除 if False，改为 _disable_npu_fused_split_qk_norm 显式开关。
- 平台判断被要求统一：'Use is_npu to determine if it is necessary' / 'Review all such judgments'，dspark_verify_window.py 直接加 is_npu() 分支被要求 'delete, change judgement in inputs_on_cuda'。
- 命名纠正：'The activation function has no relation to deepep' 促成 NPUSituDeepEPKernel → NPUSitu；'ascend to npu' 促成 dflash_ascend→ dflash_npu。
- 分支带入的 serving_chat.py 改动被追问 'why add this change?' 并要求删除。
- DSPARK 图折叠 env 被质疑（'why add this env' / 'delete it'），作者以 NPU eager fallback 精度诉求回应，最终保留。
- 将 Ascend KDA 抽象为独立后端 (design): 新增 python/sglang/srt/hardware_backend/npu/attention/ascend_kda_backend.py，把 Ascend 布局与算子差异全部收口到 AscendKDAAttnBackend。
- MLA 精度 fallback 的 if False 死代码 (correctness): 移除 if False，改为 _disable_npu_fused_split_qk_norm 显式开关，Kimi-K3 走 unfused 路径，其他模型保留 fused 路径。
- 平台判断统一使用 is_npu (design): 共享 kernel 入口改为 _is_npu / is_npu() 显式判断；dspark_verify_window.py 中直接加 is_npu 分支被要求删除，改为在 inputs_on_cuda 内判断。
- NPUSituDeepEPKernel 命名 (style): 类改名 NPUSitu，与 DeepEP 解耦。
- serving_chat.py 无关改动 (design): 该批改动从合入范围剥离。
- DSPARK 图折叠开关必要性 (design): 作者保留 SGLANG_DSPARK_FOLDED_PROPOSAL / STACKED_CTX_KV / EMBED_IN_GRAPH 等 env，说明 NPU 贪心采样需 eager fallback 保证精度，review 接受并合入。

风险与影响

- 风险：
  1. 测试缺口：整个 PR 无自动化测试文件，精度保障依赖 GSM8K/GPQA 人工验证与外部硬件 CI；PR Test Extra 目前失败，合入前未闭环。
  2. 数值等价性：MLA/KDA 的 unfused fallback 与 NPU 算子的数值行为仅在 4 节点实测中验证，若 CANN 或 sgl-kernel-npu 版本漂移可能回归。
  3. 硬编码参数：ascend_kda_backend.py 中 chunk_size = 64 等常量，review 已建议 'please read from config.json'，尚未落实。

4. 平台判断分散：add3.py、mla_output_gate.py、chunk_delta_h.py 等共享 kernel 入口引入了 is_npu 分支，后续改动容易漏掉平台分支。
5. 共享默认值：SGLANG_DSPARK_FOLDED_PROPOSAL 等 env 默认值对 GPU 同样生效，若 GPU 侧依赖旧行为需要确认。
6. 部署依赖：运行强依赖 sgl_kernel_npu 新算子，缺版本时 ImportError 报错不够友好。 - 影响：影响范围以 NPU 路径为主：Ascend backend 首次获得完整混合线性注意力（KDA）能力，并打通 DSPARK/DFLASH 推测解码与 ModelSlim 量化加载，后续 Mamba/GDN 类模型可复用 move_intermediate_cache_kda、prefix KV store 等设施。对用户，NPU 用户可按 scripts/playground/launch_kimi_k3_npu_4node.sh 四节点部署 Kimi-K3 W4A8（TP64/DP4），body 实测平均 TPOT 约 15.37 ms、吞吐约 43-45 tok/s、GSM8K 94.6%。对 GPU/ 共享路径，改动被刻意隔离，GPU CI 通过，回归风险低。对团队，' 共享后端 + 平台子类 + kernel 外置 ' 的适配模式可作为其他硬件后端接入的样板。总体影响程度中高，但风险集中在 NPU 侧。 - 风险标记：NPU 后端无自动化测试，数值等价依赖人工验证，Extra CI 失败，硬编码 chunk_size=64，平台判断分散于共享 kernel

关联脉络

- PR #32541 Kimi-K3 GPU model integration: PR body 明确基于其 GPU 集成叠加 NPU 支持，共享 kimi_k3.py 模型与 KDA 后端。
- PR #33997 Bump FlashInfer to 0.6.17 and remove Kimi K3 workarounds: GPU 侧持续移除 Kimi-K3 workaround，本 PR 的 NPU 路径需与调整后的 GPU kernel 契约保持一致。
- PR #34524 Fix DFlash sliding attention causality defaults: DFlash 推测解码链路联动，本 PR 为其补充了 NPU top-k/top-p 重归一化。