PR #34581 完整报告

sgl-project/sglang

[Diffusion] Optimizing MiniMax-H3 for consumer-level GPUs: INT8 Linear + pluggable DiT attention backends

合并时间：2026-08-18 21:44

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34581

执行摘要

- 一句话：H3 新增 INT8 量化与可插拔注意力，24GB 单卡提速 2.48×
- 推荐动作：值得精读。本 PR 的决策质量体现在三处：一是用实测而非猜测定位瓶颈（Nsight 数据 + 排除 PCIe/ 框架开销），二是"在线量化而非离线 checkpoint"的布局论证（qkv 分组重排导致离线量化为静默错误），三是对 CUTLASS Stream-K 配置选择的深入分析（21k CTA 已能填满 GPU，Stream-K 反而是纯开销）。这些分析比最终的 kernel 加速更有复用价值。

功能与动机

PR body 用 Nsight Systems 数据定位瓶颈：FlashAttention 与 DiT Linear GEMM 分别占 GPU 活动 41.8% 与 34.4%，合计约 76%；而 PCIe 带宽仅用到实测上限的 12%（3.34 / 27.0 GB/s），计算已接近 roofline（140-148 TFLOP/s vs 约 165 峰值），框架开销约 0。因此可优化的杠杆只有两个：先量化 Linear（纯数值层），再重审注意力——INT8 落地后 attention 占比升至约 51%，于是同一 PR 中同时改进两者。

实现拆解

1. 新增 kitchen_int8 量化方法与配置：在 python/sglang/multimodal_gen/runtime/layers/quantization/ 下新增 kitchen_int8.py 与 configs/kitchen_int8_config.py。KitchenInt8Config 支持 group_size（16/64/256）、ignored_layers、packed_modules_mapping；get_quant_method 对输入维度不能被 group_size 整除的层回退 UnquantizedLinearMethod（H3 的 AdaLN 投影 in=2688 走此路径）。KitchenInt8LinearMethod.create_weights 刻意与 UnquantizedLinearMethod 保持一致，使 H3 自定义 qkv loader 与 MergedColumnParallelLinear 分片逻辑无需改动；process_weights_after_loading 逐层在 CPU→GPU→CPU 之间往返量化，避免在 24 GB 卡上一次性搬移全部 60 GiB。
2. 接入量化注册表与加载器：在 quantization/__init__.py 的 QuantizationMethods Literal 与 _CUSTOMIZED_METHOD_TO_QUANT_CONFIG 中注册 kitchen_int8；在 runtime/loader/transformer_load_utils.py 的 _resolve_quant_config 中把 kitchen_int8 纳入在线量化约定（无参构造 → 加载后量化），并透传 quantization_ignored_layers。
3. 大 M INT8 GEMM 行切分：kitchen_int8.py 顶层定义 _row_split，当 rows > 8192 且 out_features >= 8192 时把激活按行切成 8192 的块多次调用

torch.ops.comfy_kitchen.int8_linear。原因是 comfy_kitchen 的 CUTLASS 配置阈值树在 M=32700 时给 qkv_proj 选了 Stream-K 调度，而该形状已能填满 GPU（21k CTA / 128 SM），Stream-K 的 workspace 与归约纯粹是开销；行切分是 bit-exact 的，因为每行的算术不变。阈值可用 SGLANG_KITCHEN_INT8_MAX_ROWS / SGLANG_KITCHEN_INT8_MIN_SPLIT_N 覆盖。

4. sol_attn 可插拔 dense backend：在 layers/attention/backends/sol_attn.py 中为 _get_sol_attn_runtime_config 新增 dense_backend 键（fa 默认 / sage_attn），forward 与 forward_varlen 在 dense 阶段按配置分发到 _dense_fa 或新增的 _dense_sage（后者用 sageattn 逐 sequence 处理）。同时在 _run_sol_attn_thd 中通过 inspect.signature 缓存 Sol 的形参集合，自动过滤不支持的参数（如 kv_splits），并在 Ada 上自动开启 int8_qk，兼容 Wan2GP 的 Ada Triton 移植版与官方 NVlabs API。
5. 测试、文档与基准：test/unit/test_sol_attn_backend.py 新增 test_dense_backend_aliases 覆盖 dense_backend 别名归一化与非法值报错；docs/cookbook/diffusion/MiniMax/MiniMax-H3.mdx、docs/docs/sglang-diffusion/quantization.mdx、attention_backends.mdx、environment_variables.mdx 补充 kitchen_int8、Sage→Sol hybrid 用法、24 GB offload 配方及两个环境变量说明。

关键文件：

- python/sglang/multimodal_gen/runtime/layers/quantization/kitchen_int8.py（模块 量化层；类别 source；类型 core-logic；符号 _row_split, _load_comfy_kitchen, KitchenInt8LinearMethod, process_weights_after_loading）：本 PR 的核心实现：新增 KitchenInt8LinearMethod，实现加载后在线 INT8 ConvRot 量化、逐层 CPU→GPU→CPU 往返避免 OOM，以及大 M GEMM 的行切分分支。
- python/sglang/multimodal_gen/runtime/layers/quantization/configs/kitchen_int8_config.py（模块 量化层；类别 source；类型 configuration；符号 KitchenInt8Config, get_name, get_supported_act_dtypes, get_min_capability）：新增 KitchenInt8Config，负责 CLI 名称注册、group_size 校验、层筛选（输入维度不可整除时回退 BF16）以及量化完成后的汇总日志。
- python/sglang/multimodal_gen/runtime/layers/attention/backends/sol_attn.py（模块 注意力；类别 source；类型 core-logic；符号 _get_sol_attn_runtime_config, SolAttnImpl._dense_sage, SolAttnImpl._dense_fa, SolAttnImpl._run_sol_attn_thd）：sol_attn 的 dense 阶段改造为可插拔后端（fa / sage_attn），并加入 Sol API 兼容性 shim（inspect.signature 过滤参数、Ada 自动 int8_qk）。
- python/sglang/multimodal_gen/runtime/loader/transformer_load_utils.py（模块 模型加载；类别 source；类型 core-logic；符号 _resolve_quant_config）：把 kitchen_int8 纳入在线量化约定（与 fp8/mxfp4 相同路径），透传 quantization_ignored_layers。
- python/sglang/multimodal_gen/runtime/layers/quantization/__init__.py（模块 量化层；类别 source；类型 dependency-wiring）：在 QuantizationMethods Literal 与 _CUSTOMIZED_METHOD_TO_QUANT_CONFIG 中注册 kitchen_int8，使 CLI 可解析。
- python/sglang/multimodal_gen/test/unit/test_sol_attn_backend.py（模块 测试；类别 test；类型 test-coverage；符号 test_dense_backend_aliases）：新增

test_dense_backend_aliases 覆盖 dense_backend 别名归一化与非法值 ValueError，是本 PR 唯一的单元测试新增。

- docs/cookbook/diffusion/MiniMax/MiniMax-H3.mdx（模块 文档；类别 docs；类型 documentation）：新增 24 GB 单卡 RTX 4090 的 kitchen_int8 + layerwise offload 配方，并补充完整基准表与精度对比。
- docs/docs/sglang-diffusion/quantization.mdx（模块 文档；类别 docs；类型 documentation）：补充 kitchen_int8 在线量化说明、适用硬件、依赖与 Warning 提示。
- docs/docs/sglang-diffusion/attention_backends.mdx（模块 文档；类别 docs；类型 documentation）：新增 dense_backend 配置项说明与 Sage→Sol hybrid 示例。
- docs/docs/sglang-diffusion/environment_variables.mdx（模块 文档；类别 docs；类型 documentation）：记录 SGLANG_KITCHEN_INT8_MAX_ROWS 与 SGLANG_KITCHEN_INT8_MIN_SPLIT_N 两个环境变量。

关键符号：KitchenInt8LinearMethod.process_weights_after_loading, KitchenInt8LinearMethod.apply, _row_split, KitchenInt8Config.get_quant_method, KitchenInt8Config.note_quantized, _get_sol_attn_runtime_config, SolAttnImpl._dense_sage, SolAttnImpl._dense_fa, SolAttnImpl._run_sol_attn_thd

关键源码片段

python/sglang/multimodal_gen/runtime/layers/quantization/kitchen_int8.py

本 PR 的核心实现：新增 KitchenInt8LinearMethod，实现加载后在线 INT8 ConvRot 量化、逐层 CPU→GPU→CPU 往返避免 OOM，以及大 M GEMM 的行切分分支。

```
# kitchen_int8.py 的核心：加载后在线量化 + 大 M 行切分前向

def _row_split(rows: int, out_features: int) -> int | None:
    """决定每次 int8_linear 调用承载多少行；返回 None 表示一次调用处理全部。"""
    if _MAX_ROWS_PER_CALL <= 0 or rows <= _MAX_ROWS_PER_CALL:
        return None # 行数未超阈值，或显式禁用切分
    if out_features < _MIN_SPLIT_OUTPUT:
        return None # 窄输出（如 out_proj/fc2 的 N=5376）切分反而更慢
    return _MAX_ROWS_PER_CALL


class KitchenInt8LinearMethod(LinearMethodBase):
    def process_weights_after_loading(self, layer: torch.nn.Module) -> None:
        from comfy_kitchen.tensor.int8 import TensorWiseINT8Layout

        weight = layer.weight.data
        if weight.dtype == torch.int8: # 已量化过，幂等保护
            return

        # 量化必须在 CUDA 上跑，但模型此刻可能还整体暂存在 CPU 上等待
        # layerwise offload。因此逐层做 CPU -> GPU -> CPU 往返，而不是
        # 依赖 loader 的整体设备搬移——后者在 24 GB 显存上会 OOM。
        home = weight.device
```

```
    qdata, params = TensorWiseINT8Layout.quantize(
        weight.to("cuda", non_blocking=True),
        is_weight=True,
        per_channel=True,
        convrot=True, # 分组 Hadamard 旋转
        convrot_groupsize=self.quant_config.group_size,
        stochastic_rounding=0, # 数据无关，结果可逐位复现
    )
    # 量化发生在 H3 自定义 qkv reorder 之后，因此布局无歧义；这也是
    # 拒绝离线预量化 checkpoint 的根因（布局不匹配是静默错误）。
    layer.weight = Parameter(qdata.to(home), requires_grad=False)
    layer.register_parameter(
        "weight_scale",
        Parameter(params.scale.to(device=home, dtype=torch.float32),
                  requires_grad=False),
    )
    self.quant_config.note_quantized(weight.numel() * weight.element_size())
    del qdata, params
    torch.cuda.empty_cache()

def apply(self, layer, x, bias=None) -> torch.Tensor:
    orig_shape = x.shape
    if x.dim() != 2:
        x = x.reshape(-1, orig_shape[-1])
    x = x.contiguous()

    def run(rows: torch.Tensor) -> torch.Tensor:
        # 单次 fused op：Hadamard 旋转 -> 动态逐行激活量化 ->
        # INT8 tensor-core GEMM -> dequant epilogue -> bias
        return torch.ops.comfy_kitchen.int8_linear(
            rows, layer.weight, layer.weight_scale, bias,
            _OUT_DTYPE_CODE[x.dtype], True, self.quant_config.group_size,
        )

    n_rows, n_out = x.shape[0], layer.weight.shape[0]
    split = _row_split(n_rows, n_out)
    if split is None:
        out = run(x)
    else:
        # 连续 2D 张量的行切片天然连续，切分不复制激活。目的：把
        # M=32700 的大形状压回 comfy_kitchen 的 data-parallel CUTLASS
        # 配置，避开 Stream-K 的 workspace 与归约开销（17.94 vs 26.5 ms）。
        out = torch.empty(n_rows, n_out, dtype=x.dtype, device=x.device)
        for start in range(0, n_rows, split):
            out[start:start + split] = run(x[start:start + split])

    if len(orig_shape) != 2:
        out = out.reshape(*orig_shape[:-1], out.shape[-1])
    return out
```

python/sglang/multimodal_gen/runtime/layers/quantization/configs/kitchen_int8_config.py

新增 KitchenInt8Config，负责 CLI 名称注册、group_size 校验、层筛选（输入维度不可整除时回退 BF16）以及量化完成后的汇总日志。

```
# kitchen_int8_config.py：层筛选与回退策略

class KitchenInt8Config(QuantizationConfig):
    def get_quant_method(self, layer, prefix) -> QuantizeMethodBase | None:
        if not isinstance(layer, LinearBase):
            return None
        if is_layer_skipped(prefix, self.ignored_layers,
                            fused_mapping=self.packed_modules_mapping):
            self.skipped.append(prefix)
            return UnquantizedLinearMethod()

        # 旋转把输入维切成固定大小的组，因此输入维不能整除 group_size 的
        # 层直接留在 BF16，而不是让整个模型加载失败。H3 的 AdaLN 投影
        #  （in=2688）正是为此存在，它们只占单步耗时的 0.2%。
        if layer.input_size % self.group_size:
            self.skipped.append(f"{prefix}(in={layer.input_size})")
            return UnquantizedLinearMethod()
        self.selected.append(prefix)
        return KitchenInt8LinearMethod(self)

    def note_quantized(self, saved_bytes: int) -> None:
        # 全部选中层量化完毕后再打一条汇总日志：静默回退 BF16 看起来
        # 就像 kernel 变慢，必须明确列出哪些层没被量化。
        self._processed += 1
        self._quantized_bytes += saved_bytes
        if self._processed == len(self.selected):
            logger.info(
                "kitchen_int8: quantized %d linear layers (%.2f GiB of BF16 "
                "weights -> %.2f GiB INT8), left %d in BF16",
                self._processed,
                self._quantized_bytes / 1024**3,
                self._quantized_bytes / 2 / 1024**3,
                len(self.skipped),
            )
```

python/sglang/multimodal_gen/runtime/layers/attention/backends/sol_attn.py

sol_attn 的 dense 阶段改造为可插拔后端（fa / sage_attn），并加入 Sol API 兼容性 shim（inspect.signature 过滤参数、Ada 自动 int8_qk）。

```
# sol_attn.py：dense 阶段的后端选择 + Sol API 兼容性 shim

_DENSE_BACKENDS = {"fa", "sage_attn"}
```

```
def _get_sol_attn_runtime_config() -> dict:
    server_args = get_global_server_args()
    cfg = getattr(server_args, "attention_backend_config", None) or {}
    dense_backend = (
        str(cfg.get("dense_backend", "fa")).strip().lower().replace("-", "_")
    )
    # 接受用户惯用别名：sage / sageattention 统一为 sage_attn
    if dense_backend in {"sage", "sageattention"}:
        dense_backend = "sage_attn"
    if dense_backend not in _DENSE_BACKENDS:
        raise ValueError(
            f"Unsupported sol_attn dense_backend={dense_backend!r}; "
            f"expected one of {sorted(_DENSE_BACKENDS)}"
        )
    ...


class SolAttnImpl(AttentionImpl):
    def _run_sol_attn_thd(self, query, key, value) -> torch.Tensor:
        from sol_attn import sol_attn
        ...
        if self._sol_params is None:
            # 缓存 Sol 版本的实际形参，适配官方 NVlabs API 与 Wan2GP 的
            # Ada Triton 移植版（后者有 int8_qk，前者没有）
            self._sol_params = frozenset(inspect.signature(sol_attn).parameters)
        kwargs = {
            "tau": cfg["tau"],
            "thresh_type": cfg["thresh_type"],
            "kv_splits": _resolve_kv_splits(q, cfg["kv_splits"]),
            "sink_start": cfg["sink_start"],
            "sink_tokens": cfg["sink_tokens"],
        }
        # Ada 上自动开启 INT8-QK 稀疏注意力
        if "int8_qk" in self._sol_params and tuple(
            torch.cuda.get_device_capability(q.device)
        ) >= (8, 9):
            kwargs["int8_qk"] = True
        # 过滤当前实现不接受的参数，避免 TypeError
        kwargs = {k: v for k, v in kwargs.items() if k in self._sol_params}
        return sol_attn(q, k, v, **kwargs).squeeze(0)
```

评论区精华

review 全程由维护者 mickqian 发起两次评论：第一次指出量化配置类更合适的位置是 quantization.configs 子包（作者回复 fix 并在 commit 7c15915 中把 KitchenInt8Config 拆分到 configs/kitchen_int8_config.py）；第二次在 APPROVED 时附带条件 please also update related docs and cookbook，作者随后补齐了 cookbook 与四份 diffusion 文档。两条评论都属结构收尾，没有针对量化数值或注意力近似的算法级争论，说明实现本身在合入前

已获得充分信任。

- 量化配置类应迁移至 quantization.configs 子包 (design): 作者回复 fix，并在 commit 7c15915 中将 KitchenInt8Config 拆分至 configs/kitchen_int8_config.py。
- 要求同步更新文档与 cookbook (documentation): 作者完成 MiniMax-H3 cookbook 与 quantization / attention_backends / environment_variables 文档更新后合入。

风险与影响

- 风险：
  - kitchen_int8 缺失独立单元测试：PR body 的 checklist 声明新增了 test_comfy_int8_linear_method.py 与 test_comfy_int8_row_split.py，但最终变更集（10 个文件）中只有 test_sol_attn_backend.py 一个测试文件。量化方法本身（含行切分分支）没有任何测试覆盖，行切分正确性依赖"逐行算术不变"的论证，缺少回归保障。
  - 数值精度改变：kitchen_int8 对 Linear 是 weight-only INT8 + 动态激活量化，PSNR 24.81 dB；sol_attn / sage_attn 改变注意力算法，PSNR 降至 23-24 dB。文档已声明两者都不是 consistency ground-truth 模式，但如果用户用 approximate 后端跑精度敏感任务，产出与 BF16 有可感知差异。
  - 外部依赖：comfy_kitchen 是第三方 abi3 扩展，不随 SGLang 分发。若未安装，KitchenInt8LinearMethod.__init__ 会抛 ImportError；BF16 默认路径不受影响，但 --quantization kitchen_int8 的失败信息依赖 ImportError 的引导质量。
  - TP>1 约束：create_weights 在 input_size_per_partition 不能被 group_size 整除时抛 ValueError；分布式 row-parallel 切分旋转分组维度时可能触发，当前文档与基准仅验证了单卡 24 GB 场景。
  - 行切分阈值是硬编码启发式：默认值 8192 基于 RTX 4090 测量得到，其他 GPU 上是否最优未验证，仅靠环境变量提供覆盖口。
- 影响：
  - 用户：把 MiniMax-H3 从"需要数据中心多卡"下沉到"单张 24 GB 消费卡可跑"，显著扩大适用硬件范围；同时引入 --attention-backend 的近似注意力选项目录，用户在速度与精度的取舍上多了两个杠杆。
  - 系统：量化注册表、transformer 加载器、注意力后端框架均有小改动，但 BF16 默认路径不改，非 diffusion 的 SRT 主链路零影响。
  - 团队：为 diffusion 子系统新增了一条"在线量化 + 后端可插拔"的优化范式，后续模型（如 Cosmos3）可复用 sol_attn 的 dense/sage 组合思路；row-split 的启发式也可能推广到其他大 M 场景。
  - 风险标记：kitchen_int8 缺独立单元测试，数值精度非 bit-exact，外部依赖 comfy-kitchen，近似注意力为 opt-in，TP>1 约束未验证

关联脉络

- PR #34986 [diffusion] feat: load quantized H3 text encoder checkpoints: 同一模型（MiniMax-H3）的量化演进线：从文本编码器的离线 FP8 checkpoint 扩展到 DiT 的在线 INT8 量化。

- PR #35107 [diffusion] Filter transformer safetensors by index.json to drop duplicate shard variants: 改动同一文件 transformer_load_utils.py，属于模型加载器行为的相邻演进。
- PR #34933 [diffusion] Per-section LoRA adapters on fused linear layers: 同在 diffusion fused linear layers 上做文章；本 PR 的 create_weights 保持与 UnquantizedLinearMethod 一致正是为了兼容这类自定义权重布局。