PR #34262 完整报告

sgl-project/sglang

[Feature] Add Muse Glimmer model support

合并时间：2026-08-12 06:41

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34262

执行摘要

- 一句话：新增 Muse Glimmer 模型原生支持，覆盖 CUDA 与 MLX 双后端
- 推荐动作：值得精读。重点看三处设计：一是 muse_glimmer_mlx.py::sanitize() 对三种 checkpoint 布局的判别与变换（offset norm 折叠、q/gate per-head interleave、位置性 norm 重命名），这是处理 vendor 私有格式的范例；二是 MuseGlimmerDetector 的 preserve_channels 机制，解决被引用的工具标记与真实工具调用的歧义，对流式解析器设计有借鉴意义；三是 remote_code_gate.py 在 mlx-lm 无 trust_remote_code 参数的情况下前置安全检查的思路。

功能与动机

PR body 明确说明目标是 "Adds native Muse Glimmer model support across the CUDA and MLX backends"，即让该模型无需依赖外部转换即可在 sglang 中部署。Issue 评论中 jasonge27 在 M4 Pro 上实测发现 MLX 端 sanitize() 无法加载公开的 mlx-community 预量化 checkpoint，Jiminator 随即澄清受支持的是 RadixArk 自研打包格式（Muse-Glimmer-q4-MLX 等）或原始 bf16 HF 导出，说明 checkpoint 兼容边界是社区最关心的问题。

实现拆解

实现按五层拆解：

1. CUDA 模型实现（新增 python/sglang/srt/models/muse_glimmer.py）：新增 MuseGlimmerForConditionalGeneration 全链路。MuseGlimmerAttention 处理 vendor 架构的特殊约定——四个逐层 norm 的权重是相对 1.0 的偏移量（加载时折叠为普通 RMSNorm）、非参数 QK-norm 在 RoPE 之前应用、attention scale 折叠为 qk_scale_factor / head_dim、attention output gate 在 o_proj 之前做 sigmoid 门控（CUDA 下用 fused_sigmoid_mul 核）。RoPE 默认 GPT-J 交错风格，no_rope_layers 标记的 NoPE 层跳过 RoPE，sliding window 层通过 RadixAttention(sliding_window_size=window-1) 表达。量化场景下退化为 unfused q/k/v 独立投影，并要求 num_key_value_heads >= tp_size。
2. MLX 模型实现（新增 python/sglang/srt/hardware_backend/mlx/models/muse_glimmer_mlx.py）：以 mlx-lm 自定义架构文件方式交付，文件必须保持独立可导入（不依赖 sglang）。sanitize() 识别三种权重布局：raw HF 导出、RC 多模态导出（model.language_model. 前缀 + 位置性 norm 重命名）、打包好的 MLX artifact（

muse_glimmer_mlx_format: 1 标记）。加载时完成 offset norm 折叠、output gate 与 q_proj 的 per-head interleave 融合、NeoX rotary 布局 permute。ModelArgs.from_dict 同时接受扁平 schema 与 RC 嵌套 schema，flatten_rc_config 处理字段映射与两个约定转换（qk_scale_factor 乘回 sqrt(head_dim)、layer_rope_theta 零值映射为 NoPE 层）。

3. 配置与多模态处理（新增 python/sglang/srt/configs/muse_glimmer.py、muse_glimmer_processing.py、multimodal/processors/muse_glimmer.py）：MuseGlimmerConfig 支持 from_dict（HF 嵌套 text_config schema 展平）与 from_gguf（从 GGUF 元数据重建，缺键回退到架构常量默认值）；MuseGlimmerImageProcessor 实现 aspect-ratio-preserving resize、patch/merge/temporal patch 展开并输出 pixel_values + image_grid_thw；MuseGlimmerProcessor 负责把 <|patch|> 占位符展开为 patch token 串。
4. 推理解析与 function call（修改 python/sglang/srt/parser/reasoning_parser.py，新增 function_call/muse_glimmer_detector.py、muse_glimmer_format.py）：MuseGlimmerDetector（reasoning parser 侧）按 recipient channel 分流 reasoning/answer/tool；关键设计是 preserve_channels 模式——当 tool_call_parser_active 时保留 to=user 通道的 framing，避免答案中被引用的工具标记被误判为真实调用；detect_and_parse 在非流式且含 ATEM 标记时回退重放一次。function call 侧 MuseGlimmerDetector 解析 <atem:invoke>/<atem:parameter> 结构，做工具名归一化与未知工具策略处理。
5. 加载、安全与投机解码配套：新增 python/sglang/srt/utils/hf_transformers/gguf_native.py，为 transformers 不支持的架构建立 GGUF_NATIVE_CONFIG_BUILDERS 注册机制，直接从 GGUF 构建 config、generation config 与 tokenizer（tokenizer 部分手动组装 tokenizers spec）；新增 hardware_backend/mlx/remote_code_gate.py，在 mlx-lm 执行 checkpoint 内 model_file 之前做安全检查，未加 --trust-remote-code 时拒绝加载；dflash.py 与 dflash_worker_v2.py 适配 Muse Glimmer 的 DFlash 草稿模型与量化 head 采样。测试配套新增 8 个以上文件，覆盖 config 展平、MLX 模型加载、detector 流式解析、remote code 门控、DFlash GSM8K 与 responses 接口 skip_special_tokens 行为。

关键文件：

- python/sglang/srt/models/muse_glimmer.py（模块 模型层；类别 source；类型 core-logic；符号 _vendor_weight_name, get_attention_sliding_window_size, MuseGlimmerMLP, MuseGlimmerAttention）：CUDA 后端主模型实现，包含 MuseGlimmerAttention/DecoderLayer/Model/ForConditionalGeneration 全套结构，处理 sandwich norm、QK-norm、attention output gate、iRoPE、sliding window 与量化退化路径。
- python/sglang/srt/hardware_backend/mlx/models/muse_glimmer_mlx.py（模块 MLX 模型；类别 source；类型 data-contract；符号 flatten_rc_config, ModelArgs, from_dict, post_init）：MLX 后端主模型文件，以 mlx-lm 自定义架构方式交付；sanitize() 兼容三种 checkpoint 布局并完成 norm 折叠与 gate 融合，ModelArgs 同时接受扁平与 RC 嵌套 schema。
- python/sglang/srt/parser/reasoning_parser.py（模块 推理解析；类别 source；类型 core-logic；符号 MuseGlimmerDetector, _consume, detect_and_parse,

parse_streaming_increment）：在既有推理解析器体系中新增 MuseGlimmerDetector，实现 recipient-channel 文本分流与 preserve_channels 防误判机制，是 function call 与 reasoning 正确性的核心所在。

- python/sglang/srt/configs/muse_glimmer.py（模块 配置层；类别 source；类型 data-contract；符号 MuseGlimmerAssistantConfig, MuseGlimmerVisionConfig, MuseGlimmerConfig, from_gguf）：定义 MuseGlimmerConfig/VisionConfig/AssistantConfig，同时支持 HF 嵌套 schema 展平与 GGUF 元数据重建，是所有加载路径的配置入口。
- python/sglang/srt/configs/muse_glimmer_processing.py（模块 图像处理；类别 source；类型 dependency-wiring；符号 get_aspect_ratio_preserving_size, MuseGlimmerImageProcessorKwargs, MuseGlimmerImageProcessor, _preprocess）：实现 Muse Glimmer 图像处理管线：aspect-ratio-preserving resize、temporal patch 展开、<|patch|> 占位符替换，是多模态输入的核心。
- python/sglang/srt/utils/hf_transformers/gguf_native.py（模块 格式适配；类别 source；类型 dependency-wiring；符号 read_gguf_architecture, has_native_gguf_support, build_gguf_config, build_gguf_generation_config）：为 transformers 白名单之外的架构建立 GGUF 原生加载机制，Muse Glimmer 是首个注册者，后续可扩展其他新架构。
- python/sglang/srt/hardware_backend/mlx/remote_code_gate.py（模块 代码门控；类别 source；类型 dependency-wiring；符号 RemoteCodeGateError, resolve_model_directory, ensure_remote_code_allowed）：填补 mlx-lm 无 trust_remote_code 参数的空白，在 checkpoint 内 Python 代码被执行前完成安全检查，属于安全关键路径。
- test/registered/unit/hardware_backend/mlx/test_muse_glimmer_mlx_model.py（模块 MLX 模型；类别 test；类型 test-coverage；符号 _raw_weights, _rc_weights, TestFlattenRcConfig, test_field_mapping_and_conversions）：用微小合成权重钉死 MLX 加载路径的三种格式变换与数值验证，弥补端到端测试依赖私有 artifact 的缺口。
- test/registered/unit/function_call/test_muse_glimmer_detector.py（模块 函数调用；类别 test；类型 test-coverage；符号 atem, TestMuseGlimmerDetector, setUp, parse）：覆盖 ATEM 工具调用的流式与非流式解析、未知工具策略与通道帧保留行为，是 function call 正确性的主要保障。

关键符号：MuseGlimmerForConditionalGeneration, MuseGlimmerModel, MuseGlimmerDecoderLayer, MuseGlimmerAttention, MuseGlimmerMLP, ModelArgs.from_dict, ModelArgs.post_init, flatten_rc_config, MuseGlimmerModel.sanitize, MuseGlimmerProcessor._preprocess, get_aspect_ratio_preserving_size, MuseGlimmerDetector.parse_streaming_increment, MuseGlimmerDetector._consume, MuseGlimmerDetector.detect_and_parse, ensure_remote_code_allowed, resolve_model_directory, build_gguf_tokenizer, build_gguf_config, muse_glimmer_config_kwargs_from_hf, muse_glimmer_config_kwargs_from_gguf

关键源码片段

python/sglang/srt/models/muse_glimmer.py

CUDA 后端主模型实现，包含 MuseGlimmerAttention/DecoderLayer/Model/ForConditional Generation 全套结构，处理 sandwich norm、QK-norm、attention output gate、iRoPE、sliding window 与量化退化路径。

```
class MuseGlimmerAttention(nn.Module):
    def __init__(self, config, layer_id, quant_config=None, prefix=""):
        super().__init__()
        # 关键设计：量化时 QKVParallelLinear 无法表达每模块单一量化格式，
        # 因此退化为三个独立 ColumnParallelLinear；同时要求 kv heads 不小于
        # tp_size，因为这里不复制 QKVParallelLinear 的 KV 头复制逻辑。
        self.unfused_qkv = quant_config is not None
        if self.unfused_qkv:
            if self.total_num_kv_heads < tp_size:
                raise ValueError(
                    "Muse Glimmer unfused q/k/v needs num_key_value_heads >= tp_size"
                )
            self.q_proj = ColumnParallelLinear(...)
            self.k_proj = ColumnParallelLinear(...)
            self.v_proj = ColumnParallelLinear(...)
        else:
            self.qkv_proj = QKVParallelLinear(...)

        # vendor 参考实现里 q 先乘 qk_scale_factor/sqrt(head_dim)，SDPA 再乘
        # 默认的 1/sqrt(head_dim)，两者在线性变换下等价于直接折叠成一个 scale。
        self.scaling = config.qk_scale_factor / self.head_dim

        # NoPE 层同时是 full_attention 层；sliding 层的窗口含 query 位置，
        # 与 HF 的 create_sliding_window_causal_mask 同带，避免 off-by-one。
        self.use_rope = config.no_rope_layers[layer_id] == 1
        self.is_sliding = config.layer_types[layer_id] == "sliding_attention"

        self.attn = RadixAttention(
            self.num_heads,
            self.head_dim,
            self.scaling,
            num_kv_heads=self.num_kv_heads,
            layer_id=layer_id,
            sliding_window_size=(
                get_attention_sliding_window_size(config) if self.is_sliding else -1
            ),
            quant_config=quant_config,
            prefix=add_prefix("attn", prefix),
        )

    def forward(self, positions, hidden_states, forward_batch):
        # 投影后先做非参数 QK-norm，再施加 RoPE，顺序与 vendor 一致。
        if self.qk_norm is not None:
            q, k = apply_qk_norm(q, k, q_norm=self.qk_norm, k_norm=self.qk_norm,
                                 head_dim=self.head_dim)
```

```
        if self.rotary_emb is not None:
            q, k = self.rotary_emb(positions, q, k)
        attn_out = self.attn(q, k, v, forward_batch)

        # attention output gate：sigmoid(gate) 逐元素作用到注意力输出上；
        # CUDA 下直接用融合核，避免一次额外的显存往返。
        if self.use_output_gate:
            gate, _ = self.output_gate_proj(hidden_states)
            if _is_cuda:
                attn_out = fused_sigmoid_mul(attn_out, gate, inplace=True)
            else:
                attn_out = torch.sigmoid(gate) * attn_out
        out, _ = self.o_proj(attn_out)
        return out
```

python/sglang/srt/hardware_backend/mlx/models/muse_glimmer_mlx.py

MLX 后端主模型文件，以 mlx-lm 自定义架构方式交付；sanitize() 兼容三种 checkpoint 布局并完成 norm 折叠与 gate 融合，ModelArgs 同时接受扁平与 RC 嵌套 schema。

```
def flatten_rc_config(config: dict) -> dict:
    """把 vendor RC 嵌套 schema 翻译成此文件的扁平 schema。"""
    text = config["text_config"]

    # 此移植只实现 silu，提前拒绝其他激活避免静默跑错。
    activation = text.get("hidden_activation", "silu")
    if activation != "silu":
        raise ValueError(f"RC config has hidden_activation={activation!r}; this port hardcodes
        silu")

    head_dim = int(text.get("head_dim", 128))
    rope_params = text.get("rope_parameters") or {}
    layer_rope_theta = text.get("layer_rope_theta")

    flat = {
        "model_type": "muse_glimmer",
        "hidden_size": text["hidden_size"],
        "num_hidden_layers": text["num_hidden_layers"],
        "num_attention_heads": text["num_attention_heads"],
        "num_key_value_heads": text["num_key_value_heads"],
        "head_dim": head_dim,
        "intermediate_size": text["intermediate_size"],
        "vocab_size": text["vocab_size"],
        "rms_norm_eps": text["rms_norm_eps"],
        "post_norm_eps": text["post_norm_eps"],
        "rope_theta": rope_params.get("rope_theta", text.get("rope_theta", 500_000.0)),
        "max_position_embeddings": text["max_position_embeddings"],
        # 关键约定转换：RC 的 qk_scale_factor 是相对 SDPA 标准 1/sqrt(head_dim)
        # 表达的，扁平 schema 则存最终折叠前的值，两者乘 sqrt(head_dim) 统一。
        "qk_scale_factor": text["qk_scale_factor"] * math.sqrt(head_dim),
```

```
        "output_multiplier": text["output_multiplier"],
        "output_soft_cap_temp": text.get("final_logit_softcapping"),
        # 当前 vendor 导出把 q/k 换成 NeoX rotary 布局，RC 路径固定按此读取。
        "rope_is_neox_style": True,
        "sliding_window": text["sliding_window"],
    }
    if "layer_types" in text:
        flat["layer_types"] = list(text["layer_types"])
    # RC 用 layer_rope_theta 的零值标记 NoPE 层，这里映射回二值列表。
    if layer_rope_theta is not None:
        flat["no_rope_layers"] = [0 if not theta else 1 for theta in layer_rope_theta]
    return flat


@dataclass
class ModelArgs(BaseModelArgs):
    # 所有默认值与 vendor 架构常量一致；muse_glimmer_mlx_format 只在打包
    # artifact 上由打包工具盖章，raw HF 导出永远不会带这个键。
    muse_glimmer_mlx_format: Optional[int] = None

    @classmethod
    def from_dict(cls, params):
        # RC 多模态 schema 的文本字段嵌套在 text_config 下，先展平再走默认逻辑。
        if "text_config" in params:
            params = flatten_rc_config(params)
        return super().from_dict(params)

    def __post_init__(self):
        # 与 vendor 配置保持一致的推导逻辑：config.json 缺省列表时也能
        # 构建出正确的 NoPE 与层类型；显式传入时则严格校验长度与取值。
        derived_no_rope = [
            0 if (self.num_hidden_layers - i - 1) % self.every_n_layers_nope == 0 else 1
            for i in range(self.num_hidden_layers)
        ]
        if self.no_rope_layers is None:
            self.no_rope_layers = derived_no_rope
        else:
            if len(self.no_rope_layers) != self.num_hidden_layers:
                raise ValueError(
                    f"no_rope_layers has {len(self.no_rope_layers)} entries but "
                    f"num_hidden_layers is {self.num_hidden_layers}"
                )
            # 非 0/1 的标记直接拒绝，避免后续把非法值当 NoPE 处理。
            bad_flags = sorted(set(self.no_rope_layers) - {0, 1})
            if bad_flags:
                raise ValueError(f"no_rope_layers contains non-binary entries {bad_flags}")

        # NoPE 层即 full_attention 层，其余都是 sliding_attention。
        derived_layer_types = [
```

```
            "full_attention" if rope_flag == 0 else "sliding_attention"
            for rope_flag in self.no_rope_layers
        ]
        if self.layer_types is None:
            self.layer_types = derived_layer_types
        else:
            # 显式传入时必须与 no_rope_layers 语义一致。
            if self.layer_types != derived_layer_types:
                raise ValueError("layer_types disagrees with no_rope_layers")
```

评论区精华

核心讨论集中在 MLX checkpoint 兼容性：

> jasonge27："I tested this PR on my M4 Pro with 48GB... the PR can't load the public mlx-community/Muse-Glimmer-30B-4bit checkpoint. muse_glimmer_mlx.py::sanitize() validates incoming weights against a raw, unquantized key schema... 627 missing keys ['lm_head.weight', ...], 1463 unexpected keys..."
>
> Jiminator："for the MLX backend on Sglang, use one of the three MLX checkpoints we host under RadixArk: Muse-Glimmer-q4-MLX, Muse-Glimmer-q4km-gs128-MLX, or Muse-Glimmer-q4k-dynamic-MLX... The mlx-community checkpoint's text weights are actually the same 4-bit quantization as our q4, but it is stored as a pre-quantized mlx-lm conversion of the full multimodal export, which isn't an input shape sanitize() supports. The loader accepts either the raw bf16 HF export... or our packaged..."

结论：这不是 bug 而是格式边界，sanitize() 只接受三种明确布局，社区预量化 mlx-lm 转换不在其中；使用指引已给出，代码未因该反馈再改动。

- MLX 后端无法加载 mlx-community 预量化 checkpoint (correctness): Jiminator 澄清：MLX 后端应使用 RadixArk 托管的三个打包 checkpoint（Muse-Glimmer-q4-MLX 等），或原始 bf16 HF 导出；mlx-community 的预量化转换不在 sanitize() 支持范围内。属于使用指引层面的解决，代码未因此改动。

风险与影响

- 风险：主要风险集中在格式兼容与平台差异：
- MLX checkpoint 兼容性：sanitize() 的严格 schema 校验会拒绝社区常见的 pre-quantized mlx-lm 导出（已在评论区实测复现），用户若未使用 RadixArk 打包件或原始 bf16 导出会直接加载失败，属预期但易踩坑。
- 旧版 RC 导出的 rope 布局歧义：muse_glimmer_mlx.py 文档明确说明旧一代 vendor 导出走该路径会得到错误的 rope 布局，且新旧 config 字节相同无法自动区分，只能依赖重新打包，存在静默错输出风险。
- CUDA SM120 MXFP8 启动风险：提交历史中修复了 MXFP8 checkpoint 在 SM120 prefill CUDA graph 捕获时的失败（CuTe-DSL 无 mm_mxfp8 核，FlashInfer autotune

掩盖了问题），最终把 swap 门控在 SM100；若用户用 MXFP8 混合精度模型在 SM120上跑，需确认该门控生效。

- remote_code_gate 行为变更：MLX 加载路径新增安全检查，凡 config.json 声明 model_file 的 checkpoint 若未加 --trust-remote-code 会被拒绝，可能影响既有自定义架构工作流。
- GGUF 自定义加载链路：gguf_native.py 绕过 transformers 的架构白名单，手工组装 tokenizer spec 与 config，依赖 gguf 包元数据完整性，代际差异靠可选键回退兜底，但缺失关键键时仍可能构造出错误配置。
- 测试覆盖边界：端到端测试依赖私有打包 artifact，公开环境只能覆盖 unit 层面的 schema 与解析逻辑，真实权重下的数值正确性验证有限。
- 影响：用户侧：CUDA 用户可直接部署 Muse Glimmer 的 HF/GGUF checkpoint（含多模态与 function call），MLX 用户必须使用 RadixArk 打包格式或原始 bf16 导出，社区预量化格式不可用。系统侧：新增可扩展的 GGUF 原生架构注册机制（GGUF_NATIVE_CONFIG_BUILDERS）与 MLX remote code 安全检查，后续其他 transformers 不支持的新架构可复用同一路径。团队侧：该 PR 为 RadixArk 合作交付，标记 release-highlight，新增大量注册到 CPU/MLX CI 的 unit 测试，扩大了多后端模型支持矩阵。
- 风险标记：新模型核心路径，MLX checkpoint 格式兼容性，remote_code 安全门控行为变更，SM120 MXFP8 启动风险，端到端测试依赖私有 artifact

关联脉络

- PR #34175 [VLM] Replace deprecated image processor use_fast: 同为多模态处理器基础设施演进：Muse Glimmer 新增的 MuseGlimmerProcessor 同样基于 transformers 处理管线，与 image processor 后端抽象改动同属一条技术线。
- PR #34405 Fix flaky decode cache-hit check in Inkling test: Inkling 与 Muse Glimmer 都是近期新增的原生模型支持，其 e2e 测试稳定性修复与本 PR 大量新增模型测试的配套思路可相互参照。
- PR #34257 [JIT Kernel] Migrate per-token FP8 quantization from AOT to JIT: Muse Glimmer 的 NVFP4/MXFP8 构建涉及 quantization 路径，JIT 量化内核迁移会影响该模型在 Blackwell 上的量化部署方式，属于同一量化技术线。