PR #51655 完整报告

vllm-project/vllm

Add Muse Glimmer model support

合并时间：2026-08-14 12:28

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51655

执行摘要

- 一句话：新增 Muse Glimmer VLM 支持：ATEM 工具解析、DFlash 投机解码
- 推荐动作：值得精读。三个设计点有很强的可迁移性：(1) 对非 JSON 协议模型，选择"分段 + 选择"替代"正则减法"解析工具调用，并在流式场景用 holdback 不动点保证输出只增不减；(2) config 双布局归一化（flat vs nested）防止 checkpoint 静默错配；(3) DFlash 的 RoPE 布局从目标模型推断而非依赖 draft 检查点。同时建议跟踪社区未决问题：draft buffer 越界修复、is_reasoning_end_streaming 增量解码、无工具结构化输出缺口，这三项都有对应后续 PR 或框架修复的空间。

功能与动机

PR body 明确说明 Muse Glimmer 的协议特殊性：该模型不输出 JSON 工具调用、不包裹推理内容在 <think> 标签中，每一轮都是 channel-scoped 消息序列，因此必须配套 --tool-call-parser muse_glimmer 与 --reasoning-parser muse_glimmer 一起使用，并强制 skip_special_tokens=False，否则标记被剥离后两个通道会全部塌缩进 content。Draft head 复用现有 qwen3_dflash 实现（同架构、同张量），在目标模型 residual stream 的 [1, 13, 25, 37, 49] 层读取并每 forward 预测 16 槽 block。社区在 Issue 评论中表达了对该模型的高需求（Google 搜索词条、recipe 页面、官方 docker tag 均已被引用），同时暴露了 DFlash 在发布镜像中不可用的问题。

实现拆解

1. 模型与配置落地：vllm/transformers_utils/configs/muse_glimmer.py 新增 MuseGlimmerTextConfig / MuseGlimmerVisionConfig / MuseGlimmerConfig / MuseGlimmerAssistantConfig（文本栈为 Gemma2 派生 + QK-norm + iRoPE + 输出门 + logit softcapping），并在 vllm/transformers_utils/config.py 与 configs/__init__.py 注册。MuseGlimmerConfig 内置 flat（legacy converter）与 nested（canonical）两种 config 布局的归一化（_looks_flat / _normalize_flat），防止旧版扁平 config 静默落到全默认值导致错误形状加载。vllm/model_executor/models/muse_glimmer.py 原生移植文本解码器与视觉编码器（稀疏块注意力、2-D RoPE、pixel-shuffle 下采样），并在 registry.py 注册 MuseGlimmerForCausalLM。
2. 多模态处理器：vllm/transformers_utils/processors/muse_glimmer.py 新增 MuseGlimmerProcessor 及 Image/Video 子处理器。chat template 每个媒体项发出一个哨兵 token（<|image|> / <|video|>），__call__ 按 _grid_size 计算出的 patch 网格展开

成 <|patch|> 序列；视频走 torchcodec 训练一致解码路径，按 patch_temporal 分组并输出真实 PTS 时间戳。

3. 双解析器（核心设计）：vllm/reasoning/muse_glimmer_reasoning_parser.py 与 vllm/tool_parsers/muse_glimmer_tool_parser.py 均围绕 channel-scoped 消息工作，adjust_request 强制 skip_special_tokens=False；工具解析器声明 supports_required_and_named=False 以绕开 JSON guided decoding 路径。两者在流式场景均采用"先分段再选通道"策略，并用 holdback 机制（_trim_open_body / _safe_open_body）保证已发出内容前缀只增不减。
4. DFlash 投机解码适配：qwen3_dflash.py 新增 dflash_target_rope_is_neox_style（从目标模型第一个注意力层读 RoPE 布局，避免 Q/K 旋转不匹配导致的静默接受率崩塌），扩展权重映射（encoder.output_norm_enc. → hidden_norm.、encoder.fc. → fc.）；vllm/v1/spec_decode/dflash.py 的 load_model 在构造 draft 前注入该布局到 draft 配置。vllm/config/speculative.py 改为显式声明 Muse Glimmer draft head 的 causality，而非放宽所有 DFlash/DSpark drafter 的默认判定（后者破坏 test_dflash_causality.py）。
5. 测试与配套：新增 7 个测试文件，覆盖 ATEM 工具解析（含通道作用域、截断隔离、逐字符流式）、parse_delta 级回归（需要真实 checkpoint，用 MUSE_GLIMMER_CKPT 环境变量控制，缺失时跳过）、config 双布局归一化与 native/modular attention 数值收敛；tests/models/registry.py 补注册测试。测试文件在 review 后从独立脚本整合为 pytest 函数。

关键文件：

- vllm/model_executor/models/muse_glimmer.py（模块 模型实现；类别 source；类型 data-contract；符号 _text_config, _vision_config, _muse_glimmer_has_vision, MuseGlimmerImagePixelInputs）：模型主体（1649 行新增）：原生移植 MuseGlimmer 文本解码器（Gemma2 派生 + QK-norm + iRoPE + 输出门 + logit softcapping）与视觉编码器，定义多模态输入 TensorSchema 契约与 ProcessingInfo，是整 PR 的核心。
- vllm/transformers_utils/configs/muse_glimmer.py（模块 模型配置；类别 source；类型 dependency-wiring；符号 _default_no_rope_layers, MuseGlimmerTextConfig, MuseGlimmerVisionConfig, MuseGlimmerConfig）：HF 配置原生副本（389 行），核心价值在 flat/nested 双布局归一化：旧版扁平 converter 的 config 若未被识别会静默落到全默认值，造成错误形状加载且无报错，_normalize_flat 配合两个重命名表从根上消除该类静默正确性 bug。
- vllm/tool_parsers/muse_glimmer_tool_parser.py（模块 工具解析；类别 source；类型 dependency-wiring；符号 _decode_value, _iter_messages, _trailing_partial_marker_len, _safe_open_body）：ATEM 工具调用解析器（501 行）。核心设计是"分段 + 选择"替代 HF 参考实现的"正则减法"，后者在截断回合会把真实工具调用与未闭合推理段一起删掉；同时处理流式标记跨 chunk、通道作用域、裸工具名归一化。
- vllm/reasoning/muse_glimmer_reasoning_parser.py（模块 推理解析；类别 source；类型 dependency-wiring；符号 _current_assistant_turn, _trim_open_body, MuseGlimmerReasoningParser, init）：channel 推理解析器（327 行）：将 to=self 通道归类为推理、to=user 归类为内容，只有真实工具通道才切换 DelegatingParser 相位；包含对未终止推理段与流式 holdback 的精细处理，是模型协议正确性的关键。

- vllm/transformers_utils/processors/muse_glimmer.py（模块 媒体处理；类别 source；类型 dependency-wiring；符号 _grid_size, MuseGlimmerImageProcessor, init, _to_norm_tensor）：多模态处理器（582 行）：image/video 处理器 + Jinja chat template + sentinel 展开。视频解码特意选用 torchcodec（训练一致路径），并暴露标准 HF AutoVideoProcessor API，是模型多模态契约的入口。
- vllm/model_executor/models/qwen3_dflash.py（模块 投机解码；类别 source；类型 data-contract；符号 dflash_target_rope_is_neox_style）：共享 DFlash 路径的关键修正（+38/-1）：新增 dflash_target_rope_is_neox_style 从目标模型推断 RoPE 布局，解决 Muse Glimmer interleaved-RoPE 目标与 draft 头旋转不匹配导致的静默接受率崩塌；同时扩展权重映射以兼容 Muse-Glimmer-30B-assistant 的 encoder.* 命名。
- vllm/v1/spec_decode/dflash.py（模块 投机解码；类别 source；类型 core-logic；符号 load_model）：DFlash draft loader 入口（+15 行）：在 load_model 中先于 draft 构造注入目标模型的 RoPE 布局，是 qwen3_dflash 推断逻辑的调用侧配套，保证新模型接入不破坏既有 DFlash 流程。
- tests/tool_use/test_muse_glimmer.py（模块 单元测试；类别 test；类型 test-coverage；符号 _fresh_parsers, _FakeReq, _req, _call）：ATEM 解析器单元测试（386 行）：覆盖通道作用域（推理回显 invoke 不解析）、并行调用跨 eom 边界、JSON 参数解码、逐字符流式、reasoning→tool handoff 回归，不依赖 checkpoint，是解析器正确性的主要保障。
- tests/tool_use/test_muse_glimmer_parse_delta.py（模块 单元测试；类别 test；类型 test-coverage；符号 _Req, tok, parser_cls, _drive）：parse_delta 级回归测试（187 行）：驱动统一 parser 引擎（DelegatingParser.parse_delta）的真实 serving 路径，覆盖无工具回合的 reasoning 相位保持、framing 不泄漏、截断 CoT 不误解析，但需要真实 MuseGlimmer tokenizer（MUSE_GLIMMER_CKPT，缺失时跳过）。
- tests/transformers_utils/test_muse_glimmer_config.py（模块 单元测试；类别 test；类型 test-coverage；符号 Cfg, init, test_flat_config_values_respected, test_flat_config_no_silent_default）：config 归一化测试（174 行）：验证 flat/nested 双布局字段保留、无静默默认、native/modular attention 配置（QK-norm、输出门、query pre-scale）数值收敛，直接守护本 PR 最重要的防错设计。

关键符号：dflash_target_rope_is_neox_style, MuseGlimmerConfig._normalize_flat, MuseGlimmerToolParser.adjust_request, MuseGlimmerToolParser._iter_messages, MuseGlimmerToolParser._safe_open_body, MuseGlimmerReasoningParser._classify_bodies, MuseGlimmerReasoningParser._trim_open_body, MuseGlimmerReasoningParser.is_reasoning_end, MuseGlimmerReasoningParser.extract_reasoning_streaming, MuseGlimmerProcessingInfo.get_mm_max_tokens_per_item, MuseGlimmerImageProcessor.preprocess_image, MuseGlimmerVideoProcessor.decode_video, load_model

关键源码片段

vllm/model_executor/models/muse_glimmer.py

模型主体（1649 行新增）：原生移植 MuseGlimmer 文本解码器（Gemma2 派生 + QK-norm + iRoPE + 输出门 + logit softcapping）与视觉编码器，定义多模态输入 TensorSchema 契约与 ProcessingInfo，是整 PR 的核心。

```
# vllm/model_executor/models/muse_glimmer.py
# MuseGlimmer 检查点存在两种形态：多模态 `MuseGlimmerConfig` 把文本配置
# 嵌套在 `text_config` 下，纯文本形态直接暴露 `MuseGlimmerTextConfig`。
# 这两个小工具函数统一了访问路径，后续的 attention 装配、权重加载
# 都通过它们取配置，避免到处写 `getattr` 判断。
def _text_config(config):
    return getattr(config, "text_config", config)

def _vision_config(config):
    return getattr(config, "vision_config", config)

# 纯文本版检查点没有 vision_config，也没有显式 has_vision 标志，
# 此时多模态 registry 的 mm_limits 必须返回空，否则会误判为支持图像。
def _muse_glimmer_has_vision(config) -> bool:
    has_vision = getattr(config, "has_vision", None)
    if has_vision is not None:
        return bool(has_vision)
    return hasattr(config, "vision_config")


# 多模态像素输入契约：图像与视频都允许变分辨率、变长度，用 TensorSchema
# 描述维度以支持动态 shape 推断。视频比图像多一个 temporal group 维度，
# 因为模型按 patch_temporal 帧做通道拼接后送入编码器。
class MuseGlimmerImagePixelInputs(TensorSchema):
    type: Literal["image_pixels"]
    pixel_values: Annotated[
        torch.Tensor | list[torch.Tensor],
        TensorShape("bn", 3, "h", "w", dynamic_dims={"h", "w"}),
    ]
    feature_sizes: Annotated[torch.Tensor, TensorShape("bn")]

class MuseGlimmerVideoPixelInputs(TensorSchema):
    type: Literal["video_pixels"]
    pixel_values: Annotated[
        torch.Tensor | list[torch.Tensor],
        TensorShape("bn", "ng", "c", "h", "w",
                    dynamic_dims={"ng", "h", "w"}),
    ]
    feature_sizes: Annotated[torch.Tensor, TensorShape("bn")]
```

vllm/transformers_utils/configs/muse_glimmer.py

HF 配置原生副本（389 行），核心价值在 flat/nested 双布局归一化：旧版扁平 converter 的 config 若未被识别会静默落到全默认值，造成错误形状加载且无报错，_normalize_flat 配合两个重命名表从根上消除该类静默正确性 bug。

```
# vllm/transformers_utils/configs/muse_glimmer.py
# 野外存在两种 MuseGlimmer config 布局：
# * CANONICAL（新 HF converter）：嵌套 text_config / vision_config，
# 字段名规范（hidden_activation / final_logit_softcapping）。
# * FLAT（旧 converter）：全部字段平铺在顶层，且命名不同
# （hidden_act / output_soft_cap_temp / vision_latent_dim ...）。
# 若不归一化，扁平 config 会静默反序列化成"全默认"文本配置——checkpoint
# 里所有值都被忽略，架构与默认值不同时直接加载出错误形状，且没有任何报错。
class MuseGlimmerConfig(PretrainedConfig):
    model_type = "muse_glimmer"
    sub_configs = {
        "text_config": MuseGlimmerTextConfig,
        "vision_config": MuseGlimmerVisionConfig,
    }

    # 扁平字段名 -> 规范字段名
    _FLAT_TEXT_RENAMES = {
        "hidden_act": "hidden_activation",
        "output_soft_cap_temp": "final_logit_softcapping",
    }
    _FLAT_VISION_RENAMES = {
        "vision_latent_dim": "hidden_size",
        "vision_heads": "num_attention_heads",
        "vision_layers": "num_hidden_layers",
        "vision_output_dim": "output_dim",
        "vision_patch_size": "patch_size",
        "vision_patch_temporal": "patch_temporal",
        "vision_adapter_dim": "adapter_dim",
        "vision_pos_emb_grid_h": "pos_emb_height",
        "vision_pos_emb_grid_w": "pos_emb_width",
        "vision_downsample_factor": "merge_kernel_size",
    }

    @classmethod
    def _looks_flat(cls, kwargs: dict) -> bool:
        # 没有显式 text_config 却在顶层出现文本级字段（hidden_size 等），
        # 就判定为扁平布局，触发归一化。
        if kwargs.get("text_config") is not None:
            return False
        return any(k in kwargs for k in ("hidden_size", "num_hidden_layers"))

    @classmethod
    def _normalize_flat(cls, kwargs: dict) -> dict:
        kwargs = dict(kwargs)
        text: dict = {}
        # 顶层文本字段先抽走并改名
        for key in list(kwargs.keys()):
            if key in cls._FLAT_TEXT_KEYS:
                canon = cls._FLAT_TEXT_RENAMES.get(key, key)
```

```
                text.setdefault(canon, kwargs.pop(key))
        vision: dict = {}
        # vision 字段同理，并处理 vision_mlp_ratio / sparse_factor 派生字段
        for key in list(kwargs.keys()):
            if key in cls._FLAT_VISION_RENAMES:
                vision.setdefault(cls._FLAT_VISION_RENAMES[key], kwargs.pop(key))
        if text:
            kwargs["text_config"] = text
        if vision:
            kwargs["vision_config"] = vision
        return kwargs
```

vllm/tool_parsers/muse_glimmer_tool_parser.py

ATEM 工具调用解析器（501 行）。核心设计是“分段 + 选择”替代 HF 参考实现的“正则减法”，后者在截断回合会把真实工具调用与未闭合推理段一起删掉；同时处理流式标记跨 chunk、通道作用域、裸工具名归一化。

```
# vllm/tool_parsers/muse_glimmer_tool_parser.py
# MuseGlimmer 不输出 JSON 工具调用，而是 ATEM 风格 XML，且每条消息带
# 通道头：to=self（推理）、to=user（回答）、to=<tool>（工具调用）。
# 通道作用域是硬约束：推理回显的 <atem:invoke> 绝不能解析成真实调用。

def _iter_messages(text: str) -> Iterator[tuple[str | None, str, bool]]:
    """把整段输出切成助手消息，产出 (接收方, 正文, 是否闭合)。

    HF 参考实现用正则“减法”剔除推理/回答段，只在完整回合上安全；
    本解析器改为先分段、再按通道选择工具正文。截断或损坏的回合里，
    减法可能把未闭合推理段之后的真实工具调用一起删掉，选择不会。
    """
    pos = 0
    while pos < len(text):
        header = _MSG_HEADER_RE.search(text, pos)
        if header is None:
            return
        body_start = header.end()
        end = _MSG_END_RE.search(text, body_start)
        nxt = _MSG_HEADER_RE.search(text, body_start)
        body_end = end.start() if end is not None else len(text)
        closed = end is not None
        # 下一条消息头同样终止当前正文：推理段的 <|eom|> 丢失（截断或
        # chunk 在推理→工具切换处丢失）时，不能让它吞掉后面的工具调用
        if nxt is not None and nxt.start() < body_end:
            body_end = nxt.start()
            closed = False
            next_pos = nxt.start()
        else:
            next_pos = end.end() if end is not None else len(text)
        body = text[body_start:body_end]
        # 正文里不可能合法出现 <|start|>；一旦出现，说明下一条消息头
```

```
        # 只生成了一半（<|message|> 未到），正则还认不出它。先切断，
        # 否则流式正文会长大再缩回，客户端会看到已发出的内容被回退。
        start_tok = body.find("<|start|>")
        if start_tok != -1:
            body = body[:start_tok]
            closed = False
        yield header.group("rcpt"), body, closed
        pos = next_pos
```

评论区精华

评审与社区讨论的焦点集中在三处：

1. 测试整合（已解决）：DarkLight1337 要求合并独立测试脚本为 test 函数，作者随后在 commit fe0b872 将 9 个 case 转为 pytest 函数并合并纯单元测试文件。
2. ATEM 解析器健壮性（部分未决）：honziik 在生产中遇到两个边界问题——_decode_value 对 type: string 的参数也做 JSON 解码导致 SchemaError；模型偶发吞掉 <atem:parameter> 开标签导致参数丢失。作者在 fork 分支提供补丁，未合入本 PR。
3. DFlash 的系列问题（社区深度追踪）：himorishige 定位注册键不匹配根因——EAGLEConfig 的 dflash 分支会给架构名加 DFlash 前缀，而 MuseGlimmerAssistantModel 是首个不以 DFlash 开头的 draft 检查点；作者确认修复已包含。maxw1489 与 myfykris 进一步追踪到 kernel warmup 崩溃的根因是 draft 输入缓冲区按 max_num_batched_tokens（2048）定尺寸，而 DFlash 一次 forward 宽度为 max_num_seqs x (1 + num_speculative_tokens) = 256 x 16 = 4096，越界写后 positions 变成垃圾值；该问题属框架级，社区用 --max-num-batched-tokens 4096 绕过，本 PR 未修复。dharaghodasara 报告 is_reasoning_end_streaming 忽略 delta_ids 导致长上下文结构化输出时每步 O(context) 解码（115.8s → 7.0s），以及无工具调用时 structured outputs grammar 可能永不生效。stepnivlk 请求将 DFlash 的 target_inner 兼容逻辑同步到 dspark 路径。

- DFlash 注册键不匹配：DFlashMuseGlimmerAssistantModel 未注册 (correctness): 作者确认修复已包含在 PR 中（'the fix has been included now'），registry 注册补充了 resolved 名称；社区在后续镜像中确认 drafter 可加载。
- DFlash draft 缓冲区越界导致 kernel warmup 崩溃 (correctness): 属 spec decode 框架级 buffer sizing 问题，本 PR 未修复；社区以调高 token budget 绕过。合并后仍可能影响默认 recipe 用户。
- is_reasoning_end_streaming 长上下文 O(context) 热路径开销 (performance): 未在 PR 内修改；建议后续将 streaming 检查改为有界尾部解码并回退到全量解码。
- 结构化输出且无工具调用时 guided decoding 可能永不生效 (correctness): 作者未回应；是否把结构化输出门控与工具解析器交接语义分离，留待后续讨论。
- ATEM 解析器边界：string 参数被 JSON 解码与吞标签 (correctness): 未解决；解析器对损坏输出的鲁棒性仍需加固。
- 测试整合为 pytest 函数 (testing): 已解决：commit fe0b872 将 9 个 case 转为 test 函数并合并为 tests/tool_use/test_muse_glimmer.py。

- DFlash 的 target_inner 兼容逻辑应同步到 DSpark 路径 (design): 未在 PR 内处理；留待 DSpark 后续支持跟进。
- 测试文件硬编码开发者家目录路径 (security): 已解决：最终版本默认值改为空字符串，仅通过 MUSE_GLIMMER_CKPT 环境变量指定。

风险与影响

- 风险:
  1. DFlash draft 缓冲区越界（高）：社区在 MI355X、GB10、SM120（RTX PRO 6000）上均复现 draft 阶段崩溃，根因是 spec decode 框架按 max_num_batched_tokens 而非 DFlash 实际 token 宽度 max_num_seqs * (1 + num_speculative_tokens) 分配缓冲区。涉及 vllm/v1/spec_decode 与调度器配置，本 PR 未做框架级修复，默认 recipe（256 并发 x 16 槽）必然触发。
  2. 解析器复杂度（中高）：muse_glimmer_reasoning_parser.py 的 _STRIP_OPEN_REASONING_RE / _OPEN_REASONING_RE 等正则以互斥条件手工建模流式边界，任何未覆盖的 tokenizer 变体都可能造成通道塌缩；honziik 报告的 string 参数被 JSON 解码与吞标签问题均未被本 PR 修复。
  3. 长上下文热路径开销（中）：is_reasoning_end_streaming 每次 delta 都解码全序列，在 128K 上下文下进入调度器热路径，存在 GPU 饥饿风险（参照 KimiK3 同款问题）。
  4. 结构化输出缺口（中）：无工具调用的 structured outputs 请求可能因 is_reasoning_end 永不置真而跳过 guided decoding。
  5. 请求级全局副作用（低中）：adjust_request 无条件 skip_special_tokens=False，影响该模型所有输出流的 framing 保留。
  6. 流程 / 供应链风险（流程性）：mdierolf 质疑 fork 代码提前发布官方 docker tag 的供应链可追溯性；作者说明 Day 0 镜像基于本 PR 构建。 - 影响：用户侧：Muse Glimmer 30B 成为开箱可用模型（recipes.vllm.ai 已有推荐页），官方 docker tag vllm-openai:muse-glimmer 已被社区广泛使用；但 DFlash 投机解码在默认配置下不可用，需 workaround 或关闭 spec decode。系统侧：新增 5 个核心源码文件（约 3400 行）与 7 个测试文件，注册点覆盖 config / processor / model registry / tool parser / reasoning parser 五处。团队侧：这是与 Meta 模型发布节奏绑定的合作式 PR，测试质量要求高（DarkLight1337 的整合要求、多轮 CI 重跑）；共享 DFlash 路径的 qwen3_dflash.py 与 dflash.py 改动影响所有 DFlash 用户，但改动向后兼容。影响程度：中等到大——新模型影响面隔离，但 spec decode 共享路径的问题暴露了框架级 buffer sizing 缺陷。 - 风险标记：DFlash draft 缓冲区越界（多硬件复现），共享 spec-decode 路径变更，解析器依赖复杂正则与 holdback，长上下文调度热路径 O(context)，结构化输出无工具场景缺口

关联脉络

- PR #52223 [Bugfix] Reapply 50869: 同为 DFlash/DSpark 配置校验线：移除 DSpark 块大小校验。本 PR 社区讨论中 myfykris 定位的 draft token budget 越界正属于同一条 spec decode 配置逻辑，且 zixi-qi 的 causality 修复提交明确提及 test_dflash_causality.py 的回归。

- PR #50062 [Model Runner V2][Spec Decode] Add KV cache support for multi-layer MTP: spec decode 基础设施（调度器、KV cache、config）的演进 PR；Muse Glimmer 的 DFlash draft head 依赖这套框架支撑 16 槽 block 预测。
- PR #52171 [Bugfix] Declare SupportsEagle3 on KimiLinearForCausalLM: 同类"新模型 + 投机解码能力声明"模式：Kimi K3 通过在模型类上声明 SupportsEagle3 修复文本版启动崩溃；本 PR 同样修改 interfaces.py 的声明接口并注册新模型。