PR #33913 完整报告

sgl-project/sglang

[inkling] Let Anthropic thinking=disabled map to reasoning effort "none"

合并时间：2026-08-08 21:59

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33913

执行摘要

- 一句话：修复 Inkling 的 Anthropic thinking=disabled 400 报错
- 推荐动作：值得精读：这是一个小而完整的 bug 修复范本——先厘清「effort 条件式 vs toggle」的语义差异，再选择在写侧做特殊映射并保留读侧不动，同时用带注释的回归测试锁定双向行为。对理解 SGLang 推理配置的写入路径（apply_reasoning_enabled）与读取路径（_get_reasoning_from_request）如何保持同步很有帮助。

功能与动机

PR body 明确指出：/v1/messages 对任何设置 thinking: {"type": "disabled"} 的 Inkling 请求返回 400，错误为 Reasoning parser 'inkling' is always-on and cannot be disabled via Anthropic thinking。但 Inkling 的推理是 effort 条件式而非 toggle 式，_parse_inkling_reasoning_effort 已接受 "none"；更关键的是 Anthropic 层在 _get_oai_reasoning_effort 中已经为 disabled 设置了 reasoning_effort="none"，随后 40 行后的 apply_reasoning_enabled 却把配置好的请求拒绝掉。这阻止了任何在小型 token 预算下需要非思考回答的调用方（例如用 max_tokens: 64 的命令安全分类器）。

实现拆解

1. 定位根因：serving_chat.py 的 apply_reasoning_enabled 中，is_always_on 分支会对 enabled=False 抛出 ValueError，而 InklingDetector 声明 reasoning_default="always"，因此 Inkling 请求天然落入该拒绝分支。
2. 修改宿主函数：在 apply_reasoning_enabled 的 hunyuan 分支之后、template_manager.reasoning_config 读取之前，插入 reasoning_parser == "inkling" 分支：enabled=False 时将 request.reasoning_effort 设为 "none" 并 return；enabled=True 时故意 fall through，既不修改显式传入的 effort，也允许后续 always-on 分支正常返回，从而保留 output_config.effort 的语义。
3. 测试配套：在 test/registered/unit/entrypoints/openai/test_serving_chat.py 的 InklingReasoningEffortTest 中新增 test_thinking_disabled_maps_to_no_thinking_effort，用 object.__new__(OpenAIServingChat) + Mock 构造最小服务对象，双向断言 disabled 映射为 "none"、enabled 保留显式 effort，覆盖了 PR 说明中「禁用映射 + 启用不覆盖」两个方向。
4. 端到端验证：PR 作者在真实 Inkling checkpoint 上验证了 effort 与推理字符数 / 回答完整性的关系（none 输出完整回答且推理 0 字符，high 在 64 token 预算下回答为空），并

确认修复前 400、修复后 200 的完整链路。CI 侧由维护者 ispobock 触发 test_serving_chat.py 重跑并通过。

关键文件：

- python/sglang/srt/entrypoints/openai/serving_chat.py（模块 推理配置；类别 source；类型 core-logic；符号 apply_reasoning_enabled）：核心修复位置：在 apply_reasoning_enabled 中为 inkling 解析器增加 effort 条件分支，把 disabled 映射为 none，并保持启用时 fall-through 以保留显式 effort。
- test/registered/unit/entrypoints/openai/test_serving_chat.py（模块 服务测试；类别 test；类型 test-coverage；符号 test_thinking_disabled_maps_to_no_thinking_effort）：新增回归测试 test_thinking_disabled_maps_to_no_thinking_effort，双向验证 disabled 映射 none 与 enabled 保留显式 effort。

关键符号：apply_reasoning_enabled, test_thinking_disabled_maps_to_no_thinking_effort

关键源码片段

python/sglang/srt/entrypoints/openai/serving_chat.py

核心修复位置：在 apply_reasoning_enabled 中为 inkling 解析器增加 effort 条件分支，把 disabled 映射为 none，并保持启用时 fall-through 以保留显式 effort。

```
# python/sglang/srt/entrypoints/openai/serving_chat.py
# apply_reasoning_enabled 方法：按启用开关把请求强制转换为开启 / 关闭模式。
def apply_reasoning_enabled(
    self, request: ChatCompletionRequest, enabled: bool
) -> None:
    # 没有注册推理解析器时，开启请求直接拒绝。
    if not self.reasoning_parser:
        if enabled:
            raise ValueError(
                "Anthropic thinking is not supported for models without "
                "a reasoning parser"
            )
        return

    # hunyuan 与 mistral 都是先例：用 effort 字符串表达开关语义。
    if self.reasoning_parser == "hunyuan":
        request.reasoning_effort = "medium" if enabled else "no_think"
        return

    if self.reasoning_parser == "inkling":
        # Inkling 是 effort 条件式推理："none" (0.0) 就是关闭开关，
        # 因此必须先于 always-on 检查处理，否则会抛 400。
        if not enabled:
            request.reasoning_effort = "none"
            return
        # 启用时故意 fall through：保留调用方显式设置的 reasoning_effort，
        # 避免进入通用 always-on 分支后被覆盖；也允许默认 effort 语义生效。
```

```
    config = self.template_manager.reasoning_config
    # 后续分支处理 mistral 特例与 always-on 拒绝逻辑。Inkling 走到这里时
    # config 为 None 且 _reasoning_default_mode() == "always"，因此
    # enabled=True 会在 always-on 分支正常 return，不会改动已设置的 effort。
```

test/registered/unit/entrypoints/openai/test_serving_chat.py

新增回归测试 test_thinking_disabled_maps_to_no_thinking_effort，双向验证 disabled 映射 none 与 enabled 保留显式 effort。

```
# test/registered/unit/entrypoints/openai/test_serving_chat.py
def test_thinking_disabled_maps_to_no_thinking_effort(self):
    """Bug 回归：Inkling 是 always-on 解析器，之前 Anthropic 的
    thinking={"type": "disabled"} 会被直接拒绝，尽管 effort "none" (0.0)
    能表达完全相同的语义。"""
    serving = object.__new__(OpenAIServingChat)
    serving.reasoning_parser = "inkling"
    serving.template_manager = Mock(reasoning_config=None)
    serving._reasoning_detector = Mock(reasoning_default="always")
    request = ChatCompletionRequest(
        model="test-model", messages=[{"role": "user", "content": "hi"}]
    )

    # 关闭方向：disabled 应映射为 "none"，而不是抛 400。
    serving.apply_reasoning_enabled(request, False)
    self.assertEqual(request.reasoning_effort, "none")

    # 开启方向：fall through 不应覆盖显式 effort。
    request.reasoning_effort = "low"
    serving.apply_reasoning_enabled(request, True)
    self.assertEqual(request.reasoning_effort, "low")
```

评论区精华

本 PR 没有 review 评论，reviewer ispobock 直接 APPROVED。核心设计决策都写在 PR body 中：作者论证了「写侧加特殊映射、读侧零改动」的理由——读侧 _get_reasoning_from_request 对 always-on parser 返回 True 是无害的，因为 InklingDetector 按 <|content_thinking|> token 做结构化解析，模型不输出思考 token 时自然找不到。另一个值得注意的点是启用方向「故意 fall through」的设计：避免覆盖 output_config.effort 传入的显式 effort。

- test_serving_chat.py CI 重跑 (other): 重跑通过，验证 PR 引入的测试在 CI 上可稳定通过。

风险与影响

- 风险：
  1. API 行为变更：thinking=disabled 的 Inkling 请求从固定 400 变为 200，依赖 400 错误进行降级 / 熔断的客户端理论上会受影响，但现实场景极少。

2. fall-through 语义依赖：启用方向依赖 Inkling 始终被声明为 always-on（_reasoning_default_mode() == "always"），若未来 Inkling 引入真正的 toggle 机制，这段 fall-through 可能导致 effort 被错误保留；当前配置下无实际风险。
3. 无性能影响：仅增加一次字符串比较，且显式选择 "none" 的请求会得到更短 prompt、零推理 token。 - 影响：影响范围集中在 Anthropic 兼容入口（/v1/messages）服务 Inkling 模型的用户：现在可以合法请求「不思考」的回答，解锁小 token 预算场景（如分类器、安全过滤）。对系统其他路径（OpenAI 入口、tokenizer、CUDA graph 等）无影响。团队维护成本极低，测试已覆盖双向行为。 - 风险标记：API 返回码行为变更，effort 映射语义依赖

关联脉络

- PR #33898 [inkling] Render tool-result media instead of coercing content to str: 同为 Inkling 解析 / 渲染路径的 bug 修复，共享 inkling_renderer 与相关测试文件所在的模型线。
- PR #33903 [Inkling] silu_and_mul: replace helion kernels with plain Triton: Inkling MoE 推理内核优化，同属 Inkling 模型支持的功能演进。
- PR #34045 Add registered short-conv tests and backend extensions: 扩展了 Inkling 短卷积后端与注册表，体现 Inkling 模型线在 SGLang 中的持续集成。