PR #24253 完整报告

sgl-project/sglang

ci: combine H200 8-GPU warmup steps and surface server log on every path

合并时间：2026-05-14 11:09

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24253

执行摘要

- 一句话：合并 H200 8-GPU 预热步骤并统一日志输出
- 推荐动作：值得精读，展示 CI 优化的典型思路：识别非持久化工作（CUDA graphs）、利用持久化缓存（DeepGEMM JIT cache）、精确匹配启动参数（FALLBACK_ARGS）、处理孤儿进程。标记文件实现和版本键设计可作为类似场景的参考。

功能与动机

PR body 指出原来的 CUDA Graphs 预热步骤浪费（CUDA graphs 不跨进程持久），且只覆盖 V3-0324 和 Ring，其他模型在测试中承受首次 JIT 编译成本，导致超时（如 step 9 花费 14 分 52 秒）。同时 warmup_server.py 只在失败路径输出日志，成功时静默删除，导致调试困难。关联 Issue #24237 修复了 shard 验证问题作为辅助。

实现拆解

1. 工作流配置（pr-test.yml）：移除 Warmup Server CUDA Graphs 步骤，将原有 Warmup DeepGEMM JIT Compilation 的模型列表从 2 个扩展为 5 个（DeepSeek-V3-0324、DeepSeek-V3.2、GLM-5-FP8、MiMo-V2-Flash、MiMo-V2.5），超时从 25 分钟调整为 60 分钟。
2. 预热脚本重写（warmup_deep_gemm.py）：新增 _kill_pg_and_wait 函数处理子进程组超时与孤儿清理；新增 get_version_key 用于基于 Python/Triton/PyTorch 版本的缓存失效标记；新增 get_fallback_marker_path、check_fallback_marker、write_fallback_marker 实现持久化标记文件，避免冷启动重复执行；新增 _watch 线程监控子进程异常退出并触发 SIGKILL；为每个模型维护 FALLBACK_ARGS 字典以匹配测试启动参数。
3. 日志输出改进（warmup_server.py）：将服务器日志 tail 从失败路径移到 finally 块，确保无论成功、失败或异常均输出最后 30 行日志；在 kill_server 中追加通过 pkill -9 -f sglang::scheduler|detokenizer 清理孤儿进程。
4. 其他调整：多处 Merge branch 'main' 对齐分支；精简 stale 模型引用（V3.2-Exp、Ring-2.5-1T）。

关键文件：

- scripts/ci/cuda/warmup_deep_gemm.py（模块 CI 脚本；类别 infra；类型 infrastructure；符号 _kill_pg_and_wait, get_version_key, get_fallback_marker_path, check_fallback_marker）：核心文件，重写预热逻辑，新增多模型支持、fallback 参数、超

时控制、标记文件、孤儿进程清理等

- scripts/ci/cuda/warmup_server.py（模块 CI 脚本；类别 infra；类型 infrastructure）：修改日志输出到 finally 块并增加孤儿进程清理
- .github/workflows/pr-test.yml（模块 CI 工作流；类别 infra；类型 infrastructure）：工作流配置，移除 Warmup Server CUDA Graphs 步骤，扩展 DeepGEMM 预热模型列表并调整超时

关键符号：_kill_pg_and_wait, get_version_key, get_fallback_marker_path, check_fallback_marker, write_fallback_marker, _watch, kill_server, warmup_one_model, wait_for_server

关键源码片段

scripts/ci/cuda/warmup_deep_gemm.py

核心文件，重写预热逻辑，新增多模型支持、fallback 参数、超时控制、标记文件、孤儿进程清理等

def_kill_pg_and_wait(proc): """SIGTERM 子进程组，超时后升级为 SIGKILL。""" try: os.killpg(os.getpgid(proc.pid), signal.SIGTERM) except (ProcessLookupError, OSError): pass try: return proc.wait(timeout=10) except subprocess.TimeoutExpired: try: os.killpg(os.getpgid(proc.pid), signal.SIGKILL) except (ProcessLookupError, OSError): pass try: return proc.wait(timeout=5) except subprocess.TimeoutExpired: return -1 defget_version_key(): """基于 Python、Triton、PyTorch 版本生成哈希，用于标记文件失效。""" parts = [sys.version] try: importtriton parts.append(f"triton={triton.__version__}") except ImportError: parts.append("triton=none") try: importtorch parts.append(f"torch={torch.__version__}") except ImportError: parts.append("torch=none") return hashlib.sha256("|".join(parts).encode()).hexdigest()[:12] # 标记文件路径基于模型、TP 和启动参数哈希 defget_fallback_marker_path(model, tp, extra_args): key = hashlib.md5(f"{model}:{tp}:{sorted(extra_args)}".encode()).hexdigest()[:8] return os.path.join(MARKER_DIR, f"{key}.marker") # check/write 省略，但读写原子文件判断是否已完成 fallback ### `scripts/ci/cuda/warmup_server.py` 修改日志输出到 finally 块并增加孤儿进程清理 ```python def kill_server(proc): """ 终止服务器进程树，并清理残留的 scheduler/detokenizer 进程。""" if proc.poll() is None: try: os.killpg(os.getpgid(proc.pid), signal.SIGTERM) except (ProcessLookupError, OSError): pass try: proc.wait(timeout=15) except subprocess.TimeoutExpired: try: os.killpg(os.getpgid(proc.pid), signal.SIGKILL) except (ProcessLookupError, OSError): pass try: proc.wait(timeout=5) except subprocess.TimeoutExpired: pass # sglang 的 scheduler_TP* 和 detokenizer 通过多进程产生独立进程组， # 在 killpg launch_server 时无法被终止，它们会持留 GPU 内存。 # 此处按名称强制杀死以免影响后续步骤。 for pattern in ("sglang::scheduler", "sglang::detokenizer"): try: subprocess.run(["pkill", "-9", "-f", pattern], timeout=5, check=False, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL) except (FileNotFoundError, subprocess.TimeoutExpired):

pass time.sleep(2) # 等待 GPU 内存释放 # warmup_one_model 的 finally 块改动： # 之前只在失败时 dump log，现在统一在 finally 中输出 tail： try: # ... wait_for_server ... finally: try: log_file.flush() with open(log_path) as f: lines = f.readlines() print(f" --- server log tail ({len(lines)} lines, last 30) ---") for line in lines[-30:]: print(f" | {line.rstrip()}") print(" --- end server log ---") except Exception: pass kill_server(proc) log_file.close() # sglang 的 scheduler_TP* 和 detokenizer 通过多进程产生独立进程组， # 在 killpg launch_server 时无法被终止，它们会持留 GPU 内存。 # 此处按名称强制杀死以免影响后续步骤。 for pattern in ("sglang::scheduler", "sglang::detokenizer"): try: subprocess.run(["pkill", "-9", "-f", pattern], timeout=5, check=False, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL) except (FileNotFoundError, subprocess.TimeoutExpired): pass time.sleep(2) # 等待 GPU 内存释放 # warmup_one_model 的 finally 块改动： # 之前只在失败时 dump log，现在统一在 finally 中输出 tail： try: # ... wait_for_server ... finally: try: log_file.flush() with open(log_path) as f: lines = f.readlines() print(f" --- server log tail ({len(lines)} lines, last 30) ---") for line in lines[-30:]: print(f" | {line.rstrip()}") print(" --- end server log ---") except Exception: pass kill_server(proc) log_file.close() ```

评论区精华

评论中 Kangyan-Zhou 指出 MiMo 模型在 warmup 中不工作（“The mimo model seems not working”），作者通过多轮 rerun 调整参数（添加 --mm-enable-dp-encoder）最终解决。另询问运行时长（“is this expected to have 40 minutes runtime?”），作者解释冷缓存需约 45 分钟，热缓存 5 分钟内。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 冷缓存超时：首次运行或缓存失效时，累计 JIT 编译约 45 分钟，超时虽调至 60 分钟但仍有临界风险（需监控 runner 磁盘或并发干扰）。
  2. 孤儿进程残留：新增的 pkill -9 -f 清理可能误杀后续用户进程（若同一 runner 上运行并行任务）；sleep(2) 延迟不能完全保证 GPU 内存释放。
  3. 标记文件失效：get_version_key 只散列 Python/Triton/PyTorch 版本，若 CUDA、DeepGEMM 版本变化未触发重新编译。
  4. 破坏其他依赖：warmup_server.py 仍被 4-gpu-h100、B200 等工作流使用，日志改进安全，但清理逻辑新增的 pkill 可能影响共存脚本。 - 影响：影响范围：H200 8-GPU per-commit CI 阶段（stage-c-test-8-gpu-h200），预期减少测试因 JIT 编译超时的概率；其他使用 warmup_server.py 的工作流也受益于日志统一输出。影响程度：中等，大幅降低人工 rerun 和调试成本，但修改了 CI 耗时预算（总时长可能从约 30 分钟增至 45-60 分钟冷启动）。团队协作上，开发者无需 SSH 访问 runner 即可获得 warmup 日志。 - 风险标记：冷缓存超时风险，孤儿进程残留，标记文件失效可能

关联脉络

- PR #24237 fix: accept 0-indexed safetensors shard names in CI weight validator: PR body 引用其失败例子，且 warmup 改进有助于隔离类似验证问题