PR #33460 完整报告

sgl-project/sglang

[CI] Build the Rust extensions on the 5090 pool and seed the cache from main

合并时间：2026-08-04 14:33

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33460

执行摘要

- 一句话：Rust 扩展构建迁移至 5090 池，新增 main 缓存种子机制
- 推荐动作：值得精读。重点看 actions/cache 的 key/ 压缩工具版本语义、跨 job 共享 CARGO_TARGET_DIR 与磁盘保护、以及 seed 工作流如何用 concurrency 和最小权限控制成本。对维护大型 CI 流水线的团队，这是一个 " 缓存生命周期管理 " 的典型案例。

功能与动机

PR body 说明了问题根源："A cache entry only reaches every PR when it was written from the default branch's ref, and pr-test.yml has no push trigger - so before this, a merge that moved the key left every PR compiling for itself until the next scheduled run, up to 12h later"。即：只有 default branch 写入的缓存条目才能覆盖所有 PR，而 pr-test.yml 没有 push 触发器，导致缓存键变化后每个 PR 都要等到下一次计划运行才能恢复命中。另一个动机是把构建节点切到 5090 池，与测试阶段保持同一镜像与 ABI 标签。

实现拆解

1. 切换构建节点：在 .github/workflows/pr-test.yml 与 pr-test-extra.yml 中，将 Rust 扩展构建 job 的 runs_on 从 x64-kernel-build-node 改为 1-gpu-5090。pr-test-extra.yml 的 artifact_name 使用 rust-ext-x86_64-extra 后缀，避免两个调用方的产物名冲突，而 cache_key_prefix 保持共享，让两次运行复用同一份构建。
2. 磁盘压力保护：在 _pr-test-rust-ext-build.yml 中，为 CARGO_TARGET_DIR 增加 85% 使用率检查。因为 5090 池的 runner 容器只编译不清理，CARGO_TARGET_DIR 无人修剪，直接复用 ci_install_dependency.sh 中已有的保护逻辑：超过阈值就删除重建目录。
3. 新增缓存种子工作流：新增 seed-rust-ext-cache.yml，在 main 分支推送且变更路径命中 rust/**或 python/setup.py 时触发（与缓存键哈希的路径一致），复用 _pr-test-rust-ext-build.yml 执行构建并保存缓存。concurrency 设为 cancel-in-progress: true，只保留最新合并的 seed 任务；permissions 声明 contents: read、issues: read，最小化权限。
4. zstd 兼容性保障：由于 actions/cache 的条目由 key 与压缩工具版本共同标识，缺少 zstd 的 runner 保存的缓存无法被恢复任务读取。构建 job 新增 " 确保 zstd" 步骤，在非 root 时通过 sudo 安装，失败只打 warning 不阻断（冷构建仍可工作）。

5. 模块加载断言：在 scripts/ci/cuda/ci_install_dependency.sh 中，用 importlib.import_module 依次导入 sglang.srt.server._core、sglang.srt.grpc._core、sglang.srt.multimodal._core。注释说明不用 find_spec 的原因：finder 只定位而不 dlopen，无法加载的 .so 会通过检查、直到某个测试套件运行时才崩溃。
6. 清理冗余属性：在 rust/sglang-server/src/utils/regex.rs 删除两个 #[allow(dead_code)]（其中一个在 pattern() getter 上重复标注），在 rust/sglang-server/src/fsm.rs 删除 RequestState 枚举上的一个。作者用 " 把所有 allow 换成 expect" 的方式验证，剩余八个 allow 仍承担作用，且其中一个只在 --lib 构建中满足，单查 --all-targets 会误删。

关键文件：

- .github/workflows/seed-rust-ext-cache.yml（模块 缓存种子；类别 infra；类型 infrastructure）：新增的缓存种子工作流，是避免 PR 冷编译的核心机制。
- .github/workflows/_pr-test-rust-ext-build.yml（模块 构建工作流；类别 infra；类型 infrastructure）：构建 job 的关键改动都在这里：磁盘压力保护、zstd 检查、构建产物验证。
- scripts/ci/cuda/ci_install_dependency.sh（模块 安装脚本；类别 infra；类型 infrastructure）：增加模块加载断言，确保 Rust 扩展 .so 能实际被 import。
- rust/sglang-server/src/utils/regex.rs（模块 正则模块；类别 source；类型 cleanup；符号 RegexPattern, pattern）：清理两个失效的 allow(dead_code)，并暴露 RegexPattern 字段的 lint 状态。
- rust/sglang-server/src/fsm.rs（模块 状态机；类别 source；类型 cleanup；符号 RequestState）：删除 RequestState 枚举上一个失效的 allow(dead_code)。
- .github/workflows/pr-test.yml（模块 主流水线；类别 infra；类型 infrastructure）：Rust 扩展构建的 runs_on 从 x64-kernel-build-node 切到 1-gpu-5090。
- .github/workflows/pr-test-extra.yml（模块 扩展流水线；类别 infra；类型 infrastructure）：与 pr-test.yml 同步切换 runs_on，并在 artifact_name 上加后缀避免与主流程碰撞。

关键符号：未识别

关键源码片段

.github/workflows/_pr-test-rust-ext-build.yml

构建 job 的关键改动都在这里：磁盘压力保护、zstd 检查、构建产物验证。

```
# 与 ci_install_dependency.sh 共用同一条缓存路径，
# 在同时跑测试阶段的 runner 上保持一份热缓存。
export CARGO_TARGET_DIR="${HOME}/.cache/sglang-cargo-target"
mkdir -p "${CARGO_TARGET_DIR}"

# 5090 池的 runner 容器只编译不清理，CARGO_TARGET_DIR 无人修剪，
# 因此重复 85% 磁盘压力保护，防止磁盘写满。
used="$(df --output=pcent "${CARGO_TARGET_DIR}" 2>/dev/null | tr -dc '0-9')"
if [ "${used:-0}" -ge 85 ]; then
  echo "cargo target dir filesystem at ${used}%; dropping ${CARGO_TARGET_DIR}"
  rm -rf "${CARGO_TARGET_DIR}"
```

```
    mkdir -p "${CARGO_TARGET_DIR}"
fi
```

scripts/ci/cuda/ci_install_dependency.sh

增加模块加载断言，确保 Rust 扩展 .so 能实际被 import。

```
# 用 importlib 导入，而非 find_spec。
# finder 只定位扩展而不 dlopen，无法加载的 .so 会通过 find_spec，
# 直到某个测试套件运行时才崩溃。
import importlib

for mod in ("server", "grpc", "multimodal"):
    name = f"sglang.srt.{mod}._core"
    try:
        importlib.import_module(name)
    except Exception as exc:
        raise SystemExit(f"{name} is present but does not load: {exc!r}")
    print(f"{name} loads")
```

评论区精华

本 PR 没有人工 review 评论，review_comments 为 0。仅有的两条 issue 评论分别是 Gemini Code Assist bot 的下线声明和作者的 /tag-and-rerun-ci 触发命令，无实质技术讨论。设计权衡全部记录在 PR body 中：缓存必须由 default branch 写入、artifact_name 按调用方加后缀而 cache_key_prefix 共享、zstd 版本差异会导致缓存不可读等。

- 暂无高价值评论线程

风险与影响

- 风险：风险：
- 缓存污染：若某个 merge 在验证前就把坏产物写入缓存，后续 PR 会静默复用。缓解：保存步骤在 verify 之后执行，该 PR 自己也验证了 glibc <= 2.34 与编译成功。
- zstd 缺失：安装步骤失败只打 warning，缓存保存仍继续，但恢复任务读不到条目，所有 PR 退回冷编译。
- 5090 池可用性：1-gpu-5090 同时服务 sgl-kernel 与 docker 构建，队列繁忙时 Rust 扩展构建可能排队；本 PR 实测排队 2 秒。
- 加载断言误报：若 _core 扩展在无 GPU 环境导入失败，会把安装阶段误判为失败；当前断言只 import 扩展本体，不依赖设备上下文，风险较低。
- 影响：对开发者：PR 的 CI 从 " 缓存键变化后冷编译 12 小时 " 收敛为缓存命中后的秒级恢复，反馈循环显著缩短。对 CI 系统：构建节点与测试阶段共用同一镜像，减少环境漂移；新增的 seed 工作流会消耗 main 分支推送后的额外 GPU runner 分钟数。对代码质量：模块加载断言为构建产物增加了一道 " 能真正加载 " 的健康检查。
- 风险标记：缓存键变化触发冷编译，zstd 缺失导致缓存不可恢复，加载断言可能误报，5090 池排队延迟

关联脉络

- PR #33384 [CI] Build the Rust extension modules once per run instead of in every CUDA job: 本 PR 的直接前身，在其基础上迁移构建节点并新增缓存种子机制。
- PR #33361 [CI] Persist the cargo build cache across CUDA CI jobs: 与 cargo 构建缓存的持久化相关，本 PR 延续并增强了缓存生命周期管理。
- PR #33437 [CI] Build the Rust extensions with the pinned toolchain instead of the image default: 同为 Rust 扩展构建链路的 CI 优化，固定工具链后迁移节点。
- PR #33441 [CI] Remove the orphaned site-packages sglang skeleton that shadows the checkout: 同为 Rust 扩展相关 CI 清理，确保构建产物解析到 checkout。