PR #33498 完整报告

sgl-project/sglang

Build and release sgl-deep-ep wheels

合并时间：2026-08-06 17:07

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33498

执行摘要

- 一句话：新增 sgl-deep-ep wheel 构建与发布流水线
- 推荐动作：该 PR 值得发布负责人和 DevOps 精读，重点学习多 CUDA 版本、多架构 wheel 的构建与发布策略（auditwheel 排除规则、PyPI 本地版本剥离、PEP 503 索引维护）。普通开发者只需了解 sgl-deep-ep 的安装方式。在合并前应确认 DeepEP#3 已合并、secrets 已配置，并在真实环境跑一次 0.1.0rc0 验证。

功能与动机

PR body 明确指出：为 sgl-deep-ep 二进制发行版添加发布流水线，让 CUDA 13 构建可通过 pip install sgl-deep-ep 安装，CUDA 12.9 构建通过 SGLang wheel 索引发布。关联 issue（sgl-project/DeepEP#3）进一步说明需要在独立分支上维护共享的 sgl_deep_ep 打包 overlay，保持三个实现分支无重复打包文件。

实现拆解

1. 构建编排脚本（scripts/build_sgl_deepep.sh）：将原构建脚本改造为 Docker 编排器，接收 Python 版本、CUDA 版本、DeepEP 源码路径、打包 overlay 路径和架构参数。内部校验输入合法性、选择对应 manylinux 基础镜像（x86_64 用 pytorch/manylinux2_28-builder，aarch64 用 pytorch/manylinuxaarch64-builder），构建并运行容器，在容器内调用 overlay 的构建脚本，再通过 auditwheel 修复为 manylinux_2_28 wheel，并为 CUDA 13 额外生成去掉本地版本号的 PyPI 专用 wheel。
2. 构建容器（docker/sgl-deep-ep.Dockerfile）：基于 PyTorch manylinux 镜像，安装 RDMA 依赖（libibverbs、rdma-core、libfabric 等）、GDRCopy 2.5.1 用户态库，并配置 CUDA stub 链接。使用 BuildKit 缓存加速 pip 安装，固定 PyTorch 2.11.0。
3. 发布工作流（.github/workflows/release-whl-deepep.yml）：定义手工触发的 workflow_dispatch，输入 version、target（all/cu129/cu130）、packaging-ref（默认 sgl-deepep-packaging）。按矩阵构建 cu129（Python 3.10/3.12 × x86_64/aarch64）和 cu130（Python 3.10–3.13 × x86_64/aarch64），对应源码分支为 sgl-deepep-x86、sgl-deepep-cu12-arm、sgl-deepep-arm。构建产物上传至 sgl-project/whl 的 GitHub Release，并克隆 gh-pages 分支更新 PEP 503 索引；cu130 wheel 同时去除 +cu130 标签后上传 PyPI。
4. 索引更新器（scripts/update_deepep_whl_index.py）：实现幂等的 PEP 503 索引更新，支持 cu129/cu130 两个 CUDA 版本，生成带 SHA256 的下载链接，并确保根索引包含

sgl-deep-ep 子目录入口。

5. 配套说明：PR 未包含测试或文档（文档改动已在评审中移除）。验证以静态检查为主（shellcheck、bash -n、预提交、字节编译等），真实 wheel 编译和 GPU/ 传输执行推迟到下一阶段。

关键文件：

- scripts/update_deepep_whl_index.py（模块 索引工具；类别 source；类型 dependency-wiring；符号 _sha256, _ensure_root_link, update_wheel_index, main）：核心的 PEP 503 索引更新逻辑，是发布流水线的关键组成部分，决定了用户能否通过 wheel 索引正确发现并下载不同 CUDA 版本的 sgl-deep-ep。
- .github/workflows/release-whl-deepep.yml（模块 发布流水线；类别 infra；类型 infrastructure）：发布工作流是流水线的中枢，定义了构建矩阵、发布目标、索引更新和 PyPI 发布的全过程。
- scripts/build_sgl_deepep.sh（模块 构建脚本；类别 other；类型 core-logic）：构建脚本是整个流程的编排核心，负责参数校验、Docker 镜像构建与运行、auditwheel 修复以及 cu130 PyPI 版本剥离。
- docker/sgl-deep-ep.Dockerfile（模块 构建镜像；类别 infra；类型 infrastructure）：容器镜像定义构建环境，包括 RDMA 依赖、GDRCopy 和 CUDA 环境配置，是跨架构一致构建的基础。

关键符号：update_wheel_index, main, _sha256, _ensure_root_link

关键源码片段

scripts/update_deepep_whl_index.py

核心的 PEP 503 索引更新逻辑，是发布流水线的关键组成部分，决定了用户能否通过 wheel 索引正确发现并下载不同 CUDA 版本的 sgl-deep-ep。

```
"""PEP 503 索引更新：为 sgl-deep-ep 发布 wheel 生成并维护索引页面。"""

import hashlib
import pathlib
import re

SUPPORTED_CUDA_VERSIONS = ("129", "130")
WHEEL_PATTERN = re.compile(
    r"^sgl_deep_ep-(?P<version>[0-9][^-]*)-[^-]+-[^-]+-[^-]+\.whl$"
)
ANCHOR_PATTERN = re.compile(r'^<a href="[^"]+">(?P<filename>[^<]+)</a><br>$')


def _sha256(path: pathlib.Path) -> str:
    """分块计算 wheel 文件的 SHA256 摘要，避免一次性读入大文件。"""
    digest = hashlib.sha256()
    with path.open("rb") as wheel_file:
        for chunk in iter(lambda: wheel_file.read(1024 * 1024), b""):
```

```
            digest.update(chunk)
    return digest.hexdigest()


def _ensure_root_link(cuda_root: pathlib.Path) -> None:
    """在 cuXXX 根索引中添加指向 sgl-deep-ep 子目录的链接，幂等地追加。"""
    root_index = cuda_root / "index.html"
    lines = root_index.read_text().splitlines() if root_index.exists() else []
    doctype = [line for line in lines if line.startswith("<!DOCTYPE")]
    anchors = [line for line in lines if line.startswith("<a ")]
    link = '<a href="sgl-deep-ep/">sgl-deep-ep</a>'
    if link not in anchors:
        anchors.append(link)
    content = [*(doctype or ["<!DOCTYPE html>"]), *sorted(set(anchors))]
    root_index.write_text("\n".join(content) + "\n")


def update_wheel_index(
    cuda_version: str, wheel_dir: pathlib.Path, repository: pathlib.Path
) -> None:
    """更新指定 CUDA 版本的 PEP 503 索引页。

    只处理带对应 +cuXXX 本地版本号的 wheel，并为每个 wheel 生成
    GitHub Release 下载链接与 SHA256 校验值。
    """
    if cuda_version not in SUPPORTED_CUDA_VERSIONS:
        raise ValueError(f"Unsupported CUDA version: {cuda_version}")

    cuda_root = repository / f"cu{cuda_version}"
    index_dir = cuda_root / "sgl-deep-ep"
    index_dir.mkdir(exist_ok=True, parents=True)
    _ensure_root_link(cuda_root)

    # 读取现有条目，保证索引可重复更新（幂等）
    index_path = index_dir / "index.html"
    entries = {}
    if index_path.exists():
        for line in index_path.read_text().splitlines():
            match = ANCHOR_PATTERN.match(line)
            if match:
                entries[match.group("filename")] = line

    # 为每个匹配的 wheel 生成带 SHA256 的下载 URL
    suffix = f"+cu{cuda_version}"
    release_base = "https://github.com/sgl-project/whl/releases/download"
    for path in sorted(wheel_dir.glob("*.whl")):
        match = WHEEL_PATTERN.match(path.name)
        if not match or suffix not in match.group("version"):
            continue
```

```
        public_version = match.group("version").split("+", 1)[0]
        url = f"{release_base}/v{public_version}/{path.name}#sha256={_sha256(path)}"
        entries[path.name] = f'<a href="{url}">{path.name}</a><br>'

    content = ["<!DOCTYPE html>", *(entries[name] for name in sorted(entries))]
    index_path.write_text("\n".join(content) + "\n")
```

scripts/build_sgl_deepep.sh

构建脚本是整个流程的编排核心，负责参数校验、Docker 镜像构建与运行、auditwheel 修复以及 cu130 PyPI 版本剥离。

```
# 在构建容器内执行的命令：调用 overlay 构建脚本、repair wheel、生成 PyPI 版本
bash /packaging/build_sgl_deep_ep.sh \
    /deepep /packaging "${raw_dir}" "${CUDA_VERSION}" "${ARCHITECTURE}"

# 使用 auditwheel 将 wheel 重打为 manylinux_2_28，并排除宿主提供的动态库
# 这些库（如 libcuda.so.1、libtorch.so）在运行时由用户环境提供，不应打入 wheel
auditwheel repair \
    --plat "manylinux_2_28_${ARCHITECTURE}" \
    --wheel-dir /output/dist \
    --exclude libcuda.so.1 \
    --exclude libcudart.so.12 \
    --exclude libcudart.so.13 \
    --exclude libtorch.so \
    --exclude libtorch_cpu.so \
    --exclude libtorch_cuda.so \
    --exclude libnccl.so.2 \
    --exclude libgdrapi.so.2 \
    "${raw_wheels[0]}"

# CUDA 13 构建额外生成 PyPI 版本：去掉 +cu130 本地版本号，改写 METADATA 并重打包
if [[ "${CUDA_TAG}" == cu130 ]]; then
    tagged_wheels=(/output/dist/sgl_deep_ep-*+cu130-*.whl)
    if [[ ${#tagged_wheels[@]} -ne 1 ]]; then
        echo "Expected exactly one CUDA 13 wheel, found ${#tagged_wheels[@]}" >&2
        exit 1
    fi
    unpack_root="$(mktemp -d -t sgl-deep-ep-pypi.XXXXXX)"
    python -m wheel unpack "${tagged_wheels[0]}" --dest "${unpack_root}"
    unpacked="$(find "${unpack_root}" -mindepth 1 -maxdepth 1 -type d | head -1)"
    dist_info="$(find "${unpacked}" -maxdepth 1 -type d -name "*.dist-info" | head -1)"
    metadata="${dist_info}/METADATA"
    original_version="$(sed -n "s/^Version:[[:space:]]*//p" "${metadata}" | head -1)"
    public_version="${original_version%+cu130}"
    if [[ "${original_version}" == "${public_version}" ]]; then
        echo "CUDA 13 wheel metadata lacks the +cu130 local version" >&2
        exit 1
    fi
    # 改写 METADATA 中的 Version 行，并重命名 dist-info 目录以匹配新版本
```

```
    sed -i "s/^Version:.*/Version: ${public_version}/" "${metadata}"
    old_dist_info="$(basename "${dist_info}")"
    new_dist_info="${old_dist_info/${original_version}/${public_version}}"
    mv "${dist_info}" "$(dirname "${dist_info}")/${new_dist_info}"
    python -m wheel pack "${unpacked}" --dest-dir /output/dist-pypi
fi
```

评论区精华

评审中作者（Fridge003）提出了两条自审意见：

- 针对 docs_new/docs/developer_guide/release_sgl_deep_ep.mdx：明确“No need to change docs folder in this PR”，最终提交中移除了文档改动，将 PR 范围收敛为工作流、Dockerfile、构建脚本和索引更新器。
- 针对 .github/workflows/release-whl-deepep.yml：将默认 packaging-ref 分支从 sgl-deepep-x86 改为 sgl-deepep-packaging，确保共享 overlay 默认使用独立维护分支，与 DeepEP#3 的设计一致。
- 移除文档改动 (documentation): 后续提交移除了文档文件，PR 收敛为工作流、Dockerfile、构建脚本和索引更新器。
- 默认 packaging-ref 分支 (design): 最终提交将默认值改为 sgl-deepep-packaging，保证工作流默认使用共享打包 overlay。

风险与影响

- 风险：
  1. 依赖未合并的上游 PR：工作流默认引用 sgl-deepep-packaging 分支，该分支上的共享 overlay 由 sgl-project/DeepEP#3 提供，若该 PR 未合并或分支不存在，构建会失败。
  2. 真实构建未验证：PR 仅通过静态检查，未实际编译 wheel。CUDA 13.0 与 aarch64 组合可能存在编译错误或运行时库缺失，尤其是 GDRCopy、libfabric 等 RDMA 依赖的兼容性。
  3. secrets 依赖：发布依赖 SGL_DEEP_EP_PYPI_TOKEN 和 GH_PAT_FOR_WHL_RELEASE 两个 secret，若未配置，发布步骤会失败。
  4. auditwheel 排除规则风险：build_sgl_deepep.sh 中排除了 libtorch.so、libnccl.so.2 等大量动态库，若这些库在目标环境缺失，wheel 运行时可能链接失败。
  5. 发布竞态：工作流设置了 concurrency 组 release-sgl-deep-ep 且 cancel-in-progress: false，但多版本并发发布时，PEP 503 索引更新可能因 git 推送竞争产生不一致。 - 影响：对用户：CUDA 13 用户可直接 pip install sgl-deep-ep，CUDA 12.9 用户通过 sgl-project/whl 索引安装，大幅简化 DeepEP 的安装步骤。对系统：新增一套完整的发布基础设施，包含 Docker 构建镜像、CI 工作流和索引更新工具，后续维护成本提高。对团队：需要维护三个 DeepEP 实现分支与共享打包 overlay 的同步，发布流程依赖两个 secret，且需按版本手动触发，建议制定发布检查单。 - 风险标记：依赖未合并的上游 PR，真实构建未验证，secrets 未配置风险，aarch64 兼容性风险，发布竞态

关联脉络

- 暂无明显关联 PR