

PR #33939 完整报告

sgl-project/sglang

[AMD] Add gfx1151 (Strix Halo / Ryzen AI MAX+) Docker image

合并时间: 2026-09-11 13:46

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/33939>

执行摘要

- 一句话: 新增 gfx1151 (Strix Halo) ROCm 镜像、构建期补丁与 nightly 发布流水线
- 推荐动作: 值得精读, 但读者按角色区分: 负责 ROCm/ 发布基础设施的工程师应逐行看 `docker/patches/sgl-kernel-gfx1151.sh`, 它是本 PR 信息密度最高的部分——wave32 部件上 host/device 两遍编译 WARP_SIZE 不一致会引发 `hipErrorLaunchFailure` 并把报错“甩锅”给后续 kernel, 这个排障经验可迁移到其他非 gfx9 目标; 同时“先 grep 断言原文、再就地替换、上游改动即 build fail”的构建期打补丁模式, 对任何需要在固定上游源码上做临时适配的场景都值得借鉴。只做模型侧开发的读者可略读。需要重点盯的两个设计决策: 一是 AITER“装但不用” (`SGLANG_USE_AITER=0` + 不设 `PREBUILD_KERNELS`) 这一妥协, 其代价是 Triton 被换成 3.7.0, 目前无 CI 验证; 二是把 gfx1151 支持完全寄托于镜像构建期补丁而非源码, 删除条件已写进 PR body (PR#31137), 建议在后续 PR 合并时同步清理并与 PR#31403 一起收口。

功能与动机

PR body 明确指出: 既有的 `docker/rocm.Dockerfile` 面向 CDNA (gfx942/gfx950), 包含 gfx1151 不支持的组件, Strix Halo 这类统一内存的 iGPU 无法直接复用。作者把范围刻意收窄到“能在 gfx1151 上把 SGLang 跑起来”: “The scope is limited to building and running SGLang on gfx1151. General RDNA kernel support and performance tuning remain separate work.” 验证环境是 WSL2 下的 Ryzen AI MAX+ 395 / Radeon 8060S, 最终镜像 13.16 GB, `import sglang`、`import sgl_kernel`、`torch.cuda.is_available()` 均通过, 并用 Qwen2.5-7B-AWQ、Phi-4-mini、Mistral-7B、Qwen3.5-9B、gpt-oss-20b-bf16 各跑通三次串行对话请求。作者也提前列明后续工作: 把 gfx1151 / wave32 支持落到 `sgl-kernel` 源码 (PR#31137)、RDNA 默认选 Triton attention (PR#31259)、去掉 gfx1151 启动路径对 `aiter` 的可选依赖 (PR#31403)。

实现拆解

1. 镜像骨架与基础镜像固定 (`docker/rocm-gfx1151.Dockerfile`)。分 `rocm-torch` 与 `sglang` 两个 stage: `ARG BASE_IMAGE` 用 digest 锁定 `rocm/pytorch@sha256:7fe531fa185...`, 并加了一条自检 `RUN python3 -c "...assert torch.version.hip" && test -x /opt/rocm/bin/hipcc`, 让基础镜像或工具链变动时构建直接失败, 而不是静默换工具链。Stage 1 设置 `PYTORCH_ROCM_ARCH=gfx1151` 与 `HSA_ENABLE_DXG_DETECTION=1` (WSL 下 `ROCDXG` 需要, 无 `/dev/dxg` 时无副作用)

- ，可单独 `--target rocm-torch` 构建，便于隔离排查。
2. sgl-kernel 的构建期适配 (`docker/patches/sgl-kernel-gfx1151.sh` + `Dockerfile` 中的构建步骤)。仓库内 `python/sglang/kernels/aot/setup_rocm.py` 的架构门禁只放行 `{gfx942, gfx950, gfx1250}` 并 `sys.exit(1)`，补丁脚本把 `gfx1151` 追加进 `allowlist`，并为该架构注入 `-DSGL_ROCM_WARP_SIZE=32`；同时改写 `include/utils.h` 的 `WARP_SIZE` 宏，新增 `#if defined(SGL_ROCM_WARP_SIZE)` 前置分支。原因是 `host pass` 下 `__HIP_DEVICE_COMPILE__` 未定义使 `WARP_SIZE = 64`，而 `wave32` 设备的 `device pass` 得到 `32`，两遍不一致会让 `__launch_bounds__(WARPS_PER_CTA * WARP_SIZE) (128)` 与 `dim3 block_dim(WARP_SIZE, WARPS_PER_TB) (256)` 对不上，启动 `256` 线程进 `128` 上限会 `hipErrorLaunchFailure` 并污染队列，报错常落在下一个 kernel (`moe_align_block_size_kernel`) 上，极易误判。所有替换都先 `grep -qF` 断言原文存在，上游改写即构建失败。`BUILD_SGL_KERNEL=0` 可整体跳过 AOT kernel，退化为纯 Triton。
 3. 依赖装配：pyproject 切换与 extras 展平。先把 `python/pyproject.toml` 换成 `python/pyproject_other.toml`（取 `srt_hip` extra，与 `docker/rocm.Dockerfile` 同款做法），再用一段内联 Python 把 `runtime_base`、`runtime_common`、`srt_hip` 三组 extras 展平成一条依赖列表后 `pip install`，绕过 `srt_hip -> sglang[runtime_common] -> sglang[runtime_base]` 的自我引用循环，并显式把 `compressed-tensors` 钉在 `0.15.0`；最后 `pip install --no-deps -e python` 装 SGLang 本体。
 4. AITER 的“装但不用”策略。AITER 在 ROCm 上并非可选：`sglang/srt/layers/quantization/__init__.py` 会 `import quark`，后者在模块作用域 `import aiter.ops.triton`，缺了它连 `import sglang` 都过不去。因此按固定 `commit c16d44b9...` 浅装 AITER，但刻意不设 `PREBUILD_KERNELS`（AOT 编 CDNA 汇编才是 `gfx1151` 上真正失败的一步），JIT 模式按需为 `gfx1151` 构建 `module_aiter_core`；运行时用 `ENV SGLANG_USE_AITER=0` 把 `sglang` 留在 Triton 路径上，并要求调用方显式传 `--attention-backend triton`，因为 `ServerArgs` 在 ROCm 上默认选 `aiter`。这一开关不止影响注意力——`aiter` 的 `RMSNorm` 用了 CDNA 专属指令 `v_pk_mul_f32`。
 5. CI / 发布配套 (`.github/workflows/release-docker-amd-gfx1151-nightly.yml`)。新增每日 `14:00 UTC`（与 CDNA ROCm nightly 的 `12:00 UTC` 错峰）的定时任务与 `workflow_dispatch`，在 `amd-docker-scale` runner、`prod` environment、`120` 分钟超时下 `--no-cache --progress=plain` 构建并推送 `rocm/sgl-dev:v<VERSION>-rocm724-gfx1151-<YYYYMMDD>`，版本号来自 `scripts/release/get_version_tag.py --tag-only`，标签同时写入 `STEP_SUMMARY`。仓库内没有任何测试文件改动，镜像正确性完全依赖这一条发布流水线与人工验证。

关键文件：

- `docker/rocm-gfx1151.Dockerfile`（模块 容器镜像；类别 `infra`；类型 `infrastructure`；符号 `rocm-torch`, `sglang`, `BASE_IMAGE`, `GPU_ARCH`）：本 PR 的核心交付物：`gfx1151` 专用两阶段镜像定义，承载基础镜像 `digest` 固定与自检、`sgl-kernel` 构建期补丁调用、`pyproject extras` 展平、AITER 装配与 `SGLANG_USE_AITER=0` 运行时开关。
- `docker/patches/sgl-kernel-gfx1151.sh`（模块 构建补丁；类别 `infra`；类型 `build-patch`；符号 `GATE_OLD`, `GATE_NEW`, `FLAGS_OLD`, `FLAGS_NEW`）：让 `gfx1151` 能编出 `sgl-kernel` 的关键适配脚本：放开架构 `allowlist`、注入 `SGL_ROCM_WARP_SIZE=32`、改

写 include/utils.h 的 WARP_SIZE 宏，且全部改动带 grep 守卫。

- .github/workflows/release-docker-amd-gfx1151-nightly.yml（模块 发布流水线；类别 infra；类型 configuration；符号 publish, rocm/sgl-dev, amd-docker-scale）：新增 gfx1151 镜像的每日发布流水线，决定了该镜像的交付节奏、标签规范与构建成本，是这条新平台的运维入口。

关键符号：docker/patches/sgl-kernel-gfx1151.sh：带 grep 守卫的构建期补丁主流程，setup_rocm.py 架构 allowlist 门禁（GATE_OLD / GATE_NEW），include/utils.h WARP_SIZE 宏（新增 SGL_ROCM_WARP_SIZE 前置分支），rocm-gfx1151.Dockerfile：rocm-torch / sglang 两阶段构建目标，pyproject extras 展平脚本（runtime_base + runtime_common + srt_hip 去环），release-docker-amd-gfx1151-nightly.yml：publish job 的版本号与镜像标签生成

关键源码片段

docker/rocm-gfx1151.Dockerfile

本 PR 的核心交付物：gfx1151 专用两阶段镜像定义，承载基础镜像 digest 固定与自检、sgl-kernel 构建期补丁调用、pyproject extras 展平、AITER 装配与 SGLANG_USE_AITER=0 运行时开关。

```
# =====
==
# Stage 2: 在 gfx1151 ROCm 栈之上叠加 SGLang。
# Stage 1 (rocm-torch) 只负责校验基础镜像与工具链，可单独 --target rocm-torch 构建；
# sgl-kernel 与 SGLang 本体都在这一层。
# =====
==
FROM rocm-torch AS sglang

ARG GPU_ARCH=gfx1151
# sgl-kernel 的 ROCm 构建脚本只接受 gfx942/gfx950/gfx1250，其余架构直接 sys.exit(1)，
# 下面的构建期补丁会摘掉这道门；置 0 可完全跳过 AOT kernel，退化为纯 Triton 路径。
ARG BUILD_SGL_KERNEL=1
ARG MAX_JOBS=12

# 上游 pyproject.toml 钉死了 CUDA 栈（torch、flashinfer[cu13] 等），
# 换成 pyproject_other.toml 里的 srt_hip extra —— 与 docker/rocm.Dockerfile 同款做法。
RUN cd /sgl-workspace/sglang \
    && rm -f python/pyproject.toml \
    && mv python/pyproject_other.toml python/pyproject.toml

# 先 grep 断言待改文本存在，再打补丁：上游一旦改写源码，构建大声失败，
# 而不是静默产出一个 WARP_SIZE 用错的 kernel 镜像。
COPY docker/patches/sgl-kernel-gfx1151.sh /tmp/sgl-kernel-gfx1151.sh
RUN cd /sgl-workspace/sglang/python/sglang/kernels/aot \
    && if [ "${BUILD_SGL_KERNEL}" = "1" ]; then \
        rm -f pyproject.toml \
        && mv pyproject_rocm.toml pyproject.toml \
```

```

    && sh /tmp/sgl-kernel-gfx1151.sh setup_rocm.py \
    && AMDGPU_TARGET=${GPU_ARCH} MAX_JOBS=${MAX_JOBS} python3 setup_rocm.py
    install; \
else \
    echo "Skipping sgl-kernel build (BUILD_SGL_KERNEL=0)"; \
fi

# 当前 main 通过自我引用组合 extras (srt_hip -> sglang[runtime_common] -> sglang[runtime_
base]) ,
# pip 解析器在 editable 源码树下会在这个环里递归; 这里先把三组展平成一条依赖列表,
# 再 --no-deps 安装包本身, 并把 compressed-tensors 钉在最后一个兼容 torch 2.9 的版本。

# aiter 在 ROCm 上并非可选: quantization/__init__.py 会 import quark,
# 后者在模块作用域 import aiter.ops.triton, 缺了它连 import sglang 都过不去。
# 刻意不设 PREBUILD_KERNELS: AOT 编译 CDNA 汇编 kernel 才是 gfx1151 上真正失败的一步,
# JIT 模式下 aiter 会按需为 gfx1151 构建 module_aiter_core。
ARG AITER_REPO="https://github.com/ROCm/aiter.git"
ARG AITER_COMMIT="c16d44b93a528b2a4bfd6d8d3409116d465872a9"

RUN git clone --recursive ${AITER_REPO} /sgl-workspace/aiter \
    && cd /sgl-workspace/aiter \
    && git checkout ${AITER_COMMIT} \
    && git submodule update --init --recursive \
    && GPU_ARCHS=${GPU_ARCH} pip install --no-cache-dir --no-build-isolation \
        --config-settings editable_mode=compat -e .

# aiter 的注意力 / MoE kernel 面向 CDNA; ServerArgs 在 ROCm 上默认选 aiter,
# 因此调用方必须显式传 --attention-backend triton, 直到上游修好 RDNA 默认值。
# 这个开关不止管注意力: aiter 的 RMSNorm 用了 CDNA 专属指令 v_pk_mul_f32。
ENV SGLANG_USE_AITER=0

```

docker/patches/sgl-kernel-gfx1151.sh

让 gfx1151 能编出 sgl-kernel 的关键适配脚本: 放开架构 allowlist、注入 SGL_ROCM_WARP_SIZE=32、改写 include/utils.h 的 WARP_SIZE 宏, 且全部改动带 grep 守卫。

```

#!/bin/sh
# 教 python/sglang/kernels/aot/setup_rocm.py 为 gfx1151 (Strix Halo) 构建。
# 由 docker/rocm-gfx1151.Dockerfile 在构建期调用; 仓库文件本身保持不变,
# 因为 gfx1151 目前不是 SGLang 的正式支持目标。
set -e

FILE="${1:?usage: sgl-kernel-gfx1151.sh <path to setup_rocm.py>}"
UTILS="$(dirname "${FILE}")/include/utils.h"

# 改动 1: 摘掉 {gfx942, gfx950, gfx1250} 的架构 allowlist (原先会让构建 sys.exit(1)) 。
# 当前 main 给非 gfx942 目标 40KB 的 TopK 动态 LDS 预算, 已经装得进 gfx1151 的 64KB
workgroup 上限。
GATE_OLD='if amdgpu_target not in ["gfx942", "gfx950", "gfx1250"]:'

```

```
GATE_NEW='if amdgpu_target not in ["gfx942", "gfx950", "gfx1250", "gfx1151"]:'
```

```
# 改动 2: 为 gfx1151 单独追加编译宏, 把 host / device 两遍编译钉到同一 wave size。
```

```
FLAGS_OLD='          f"-DSGL_TOPK_DYNAMIC_SMEM_BYTES={topk_dynamic_smem_bytes}",'
FLAGS_NEW='          f"-DSGL_TOPK_DYNAMIC_SMEM_BYTES={topk_dynamic_smem_bytes}",
          # gfx1151 是 wave32 部件, 两遍编译都必须看到 32 (见下方 utils.h) 。
          *(["-DSGL_ROCM_WARP_SIZE=32"] if amdgpu_target == "gfx1151" else []),'
```

```
# include/utils.h 里 WARP_SIZE 在 host pass (__HIP_DEVICE_COMPILE__ 未定义) 解析为 64,
# 在非 __GFX9__ 的 device pass 解析为 32; CDNA 上两边都是 64, 所以上游从未暴露这个问题。
# 在 gfx1151 上两遍不一致会直接炸:
```

```
# moe_topk_softmax_kernels.cu __launch_bounds__(WARPS_PER_CTA * WARP_SIZE) -> device
侧 4*32 = 128
```

```
# moe_topk_softmax_kernels.cu dim3 block_dim(WARP_SIZE, WARPS_PER_TB) -> host 侧 64*4 =
256
```

```
# 256 线程启动进 128 线程上限会 hipErrorLaunchFailure、污染队列,
```

```
# 最终常表现为下一个 kernel (moe_align_block_size_kernel) 的 page fault, 极易误判。
```

```
# 同一套模式也在 moe_topk_sigmoid_kernels.cu 里; 只有 MoE 模型会命中, dense 模型不调用这些
kernel。
```

```
# 32 就是 gfx1151 的正确取值, 所以这里直接在宏定义处钉死, 而不是逐调用点改名。
```

```
WARP_OLD='#if defined(__GFX9__) || !defined(__HIP_DEVICE_COMPILE__)
```

```
#define WARP_SIZE 64'
```

```
WARP_NEW='#if defined(SGL_ROCM_WARP_SIZE)
```

```
#define WARP_SIZE SGL_ROCM_WARP_SIZE
```

```
#elif defined(__GFX9__) || !defined(__HIP_DEVICE_COMPILE__)
```

```
#define WARP_SIZE 64'
```

```
# 每次替换前先断言原文仍在: 上游改写就 fail, 而不是静默产出用错 launch bounds 的镜像。
```

```
for pattern in "${GATE_OLD}" "${FLAGS_OLD}"; do
```

```
    if ! grep -qF "${pattern}" "${FILE}"; then
```

```
        echo "ERROR: expected line not found in ${FILE}:" >&2
```

```
        echo " ${pattern}" >&2
```

```
        echo "setup_rocm.py changed upstream; re-check this patch before building." >&2
```

```
        exit 1
```

```
    fi
```

```
done
```

```
if ! grep -qF "${WARP_OLD}" "${UTILS}"; then
```

```
    echo "ERROR: expected WARP_SIZE block not found in ${UTILS}." >&2
```

```
    echo "utils.h changed upstream; re-check the wave32 fix before building." >&2
```

```
    exit 1
```

```
fi
```

```
# 断言通过后就地改写: utils.h 替换一次, setup_rocm.py 替换门禁 + 追加编译宏。
```

```
# ... (内联 Python 按 old/new 字符串执行 str.replace)
```

```
echo "Patched ${FILE} for gfx1151:"
```

```
grep -nF -e "${GATE_NEW}" -e "SGL_ROCM_WARP_SIZE" "${FILE}"
```

```
echo "Patched ${UTILS} for wave32:"
```

```
grep -nF "SGL_ROCM_WARP_SIZE" "${UTILS}"
```

评论区精华

讨论集中在 Dockerfile 第 146 行 (AITER 安装处) 的一条线程:

bingxche: “When installing AITER, if not set `AITER_USE_SYSTEM_TRITON=1`, AITER will reinstall Triton 3.7, is this expected?”

hubertlu-tw (首轮回应): “This was not intended. ... the ROCm 7.2.4 base provides `triton==3.5.1+rocm7.2.4.gita272dfa8`, but the pinned AITER installer replaces it with `triton==3.7.0+amd.rocm7.2.0.git89002410`. ... The PR has already merged, so I will handle this as a follow-up fix.”

hubertlu-tw (实测后撤回该方案): “I tested `AITER_USE_SYSTEM_TRITON=1` against the merged image ... Both `import aiter` and Qwen2.5 AWQ server startup fail with `RuntimeError: aiter gluon kernels require triton>=3.6.0`. Therefore, setting this variable alone is not a valid fix. ... A longer-term fix should remove the unconditional AITER import dependency on this path or use an AITER/Triton combination compatible with the ROCm 7.2.4 base.”

结论: Triton 被替换是当前 pin 下的预期行为且是启动必需, 但长期应解耦 AITER 的无条件 import (对应 PR body 里的 follow-up PR#31403)。HaiShaw 给出 APPROVED / “LGTM”, 其余 hubertlu-tw 的 COMMENTED 均为该线程内的追问与澄清, 没有第二处争议点。

- AITER 安装是否应保留基础镜像自带的 ROCm 匹配 Triton (question): 问题被确认成立但未在本 PR 内修复; 作者承诺以 follow-up PR 处理。
- Triton 版本回退方案实测失败, 长期解法指向解耦 AITER 无条件 import (design): 接受现状 (保留 Triton 3.7) 并记为已知技术债, 目录指向 PR body 中的 follow-up PR#31403。

风险与影响

- 风险:

1. 构建期文本补丁的脆弱性 (docker/patches/sgl-kernel-gfx1151.sh): 补丁以字面文本 `grep -qF` 匹配 `setup_rocm.py` 的架构门禁、`SGL_TOPK_DYNAMIC_SMEM_BYTES` 编译参数与 `include/utils.h` 的 `WARP_SIZE` 块。上游任何一次无关重排、改名或格式化都会让 nightly 构建直接失败 (设计上“大声失败”优于静默错配 kernel), 但也意味着这条流水线的可用性被上游源码文本稳定性绑架; 补丁只在 `gfx1151` 目标下注入 `-DSGL_ROCM_WARP_SIZE=32`, 对 CDNA 分支无影响。

1. AITER 覆盖 Triton 版本 (Dockerfile 第 146 行附近): 固定 commit 的 AITER 把基础镜像自带、与 ROCm 7.2.4 配套的 `triton==3.5.1+rocm7.2.4` 替换为 `triton==3.7.0+amd.rocm7.2.0`。 `AITER_USE_SYSTEM_TRITON=1` 已验证不可行 (`import aiter` 报 `aiter gluon kernels require triton>=3.6.0`, Qwen2.5 AWQ 启动同样失败), 说明镜像内 Triton 与 ROCm 7.2.4 的匹配关系已被破坏, 而 digest 固定基础镜像的初衷正是锁定工具链; 这对其他依赖 Triton 的路径 (`torch.compile`、其他 Triton 内核) 存在潜在 ABI 级别的不确定性, 且尚未有 CI 覆盖验证。

2. 运行时依赖调用方纪律: `ENV SGLANG_USE_AITER=0` 只关掉 SGLang 侧的 aiter 分派, 注意力后端仍需用户显式传 `--attention-backend triton` (ServerArgs 在 ROCm 上默认 aiter)。部署者照抄旧 ROCm 命令会直接踩坑; `--mem-fraction-static 0.5` 在 395 上约占 52 GB 共享内存, PR body 自己也强调这不是生产容量建议。
3. 自动化验证缺失: 仓库无任何测试改动, nightly workflow 只做 `docker build + docker push`, 不含容器内 `import sglang`、`import sglang_kernel` 或最小推理冒烟; PR 状态里“Latest PR Test (Extra)”为失败、“AMD ROCm 10”槽位显示该 commit 没有对应的 AMD PR run, 真实硬件验证来自作者 / 维护者手工执行。
4. 产物与成本: 镜像 13.16 GB, nightly 走 `--no-cache` 全量重建 (sglang-kernel+AITERJIT), `MAX_JOBS=12`、`CARGO_BUILD_JOBS=8`、timeout 120 分钟, 对 `amd-docker-scale` 的机时与 registry 存储是持续性开销。- 影响: 对用户: AMD Strix Halo / Ryzen AI MAX+ 这类共享内存 iGPU 首次有了官方镜像入口 (`rocm/sglang-dev:*gfx1151*`), 但可用级别是“功能跑通”——作者自述吞吐、延迟、并发、显存与 CUDA graph 调优另行测量, 且必须显式 `--attention-backend triton`。对系统: 改动 100% 落在构建产物侧, 仓库源码 (含 `setup_rocm.py`、`include/utils.h`) 保持原样, 既有 CDNA 镜像与 gfx942/gfx950 路径零回归; 镜像内部则引入了一条与基础镜像原配 Triton 不一致的依赖 (3.7.0 vs 3.5.1)。对团队: 新增一个长期维护面——nightly 流水线、`rocm/sglang-dev` 标签规则、以及必须回收的构建期补丁 (PR#31137 落地后即可删除) 和 AITER import 解耦 (PR#31403)。由于补丁的 grep 守卫会随上游源码漂移而炸, AMD 侧需要有人对这条流水线的红灯负责。- 风险标记: 构建期文本补丁易碎, Triton 版本被 AITER 覆盖, 镜像无自动化冒烟测试, 运行时常需手动指定 attention backend, nightly 全量重建成本高

关联脉络

- PR #31137 sglang-kernel 侧 gfx1151 / wave32 源码级支持 (PR body 引用的 follow-up): 本 PR 的构建期补丁正是为等待该源码级支持而存在; 一旦落地, `docker/patches/sglang-kernel-gfx1151.sh` 即可删除。
- PR #31403 移除 gfx1151 启动路径上的可选 aiter 依赖 (PR body 引用的 follow-up): 直接对应本次 review 中 AITER/Triton 版本冲突的根因——unconditional AITER import; 该 PR 落地后镜像可不再安装 AITER, 从而保留基础镜像的 Triton 3.5.1。
- PR #31259 在受支持的 RDNA 目标上默认选择 Triton attention (PR body 引用的 follow-up): 本镜像要求用户手工传 `--attention-backend triton`, 正是因为 ServerArgs 在 ROCm 上默认走 aiter; 该 PR 落地后这条使用约束才会消失。
- PR #38968 [NPU] Change npu.Dockerfile working directory to /sglang-workspace: 同为近期平台专用镜像的基础设施改动, 且同样使用 /sglang-workspace 作为工作目录, 可对照两者的镜像布局约定。
- PR #39044 [AMD][CI] Temporarily pause MI355X disaggregated nightly: 同为 AMD 侧 CI / nightly 流水线的近期调整, 反映 AMD 平台 CI 资源与稳定性维护的连续脉络。