PR #31300 完整报告

sgl-project/sglang

[Build] Add srt_empty extra group for device-agnostic install

合并时间：2026-08-06 04:17

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/31300

执行摘要

- 一句话：新增 srt_empty 安装组，支持非 NVIDIA 平台免 torch 安装
- 推荐动作：值得精读。核心看点有两个：一是用"配置拆组 + 契约测试"替代 vLLM 动态 setup.py 方案的依赖治理模式；二是核心模块顶层 CUDA/Triton 导入的惰性化改造（TYPE_CHECKING、try/except、None 守卫）如何在保持运行时行为不变的前提下扩展安装面。建议关注两项已记录的遗留事项：uv pip compile 传递依赖 CI、FLA_CHUNK_SIZE 抽取共享常量文件；未来合入涉及 scheduler.py、server_args.py 的 PR 时注意与本 PR 导入区域的交互。

功能与动机

PR body 明确指出：runtime_common 包含 torchao 和 timm，会拉入特定版本 torch，与厂商 PyTorch 构建（torch_npu、torch_musa 等）冲突，阻碍 OOT 插件在非 NVIDIA 平台把 SGLang 当作纯 Python 基座使用。vLLM 通过 VLLM_TARGET_DEVICE=empty 解决同类问题（vLLM PR #4773），而 SGLang 采用声明式 pyproject.toml 而非动态 setup.py，因此选择新增 extra 组。该 PR 同时是 sglang-plugin-FL 生态（FlagGems 统一 Triton 算子库 + FlagCX 统一通信库，基于 PR #21388 引入的 hook 系统）落地的基础设施。

实现拆解

1. 依赖契约拆分（python/pyproject_other.toml）：把原 runtime_common 的平铺列表拆为 runtime_base（约 48 个纯 Python 包，不依赖 torch）与 runtime_common（引用 sglang[runtime_base] 并叠加 compressed-tensors、outlines==0.1.11、timm==1.0.16、torchao==0.9.0、xgrammar==0.2.1 六个会传递拉入 torch/triton 的包），新增 srt_empty = ["sglang[runtime_base]"]。评审中 alexnails 用 uv pip compile 验证：移出这三个传递拉 torch 的包后，安装图从 163 包（含 torch、triton 与 15 个 nvidia wheel）收敛为 128 包零 torch/triton/nvidia。设计上保证 pip install -e ".[runtime_common]" 与拆分前安装结果一致。
2. 源码顶层导入去 CUDA/Triton 化：python/sglang/srt/server_args.py 删除模块顶部 from sglang.kernels.ops.attention.fla.chunk_delta_h import CHUNK_SIZE as FLA_CHUNK_SIZE（该模块会 import triton），改为在 mamba_cache_chunk_size 属性内惰性导入，ImportError 时回退 64 并加注释必须与上游一致；python/sglang/srt/managers/scheduler.py 把 torch.cuda.Stream 移入 if TYPE_CHECKING（纯注解用途、运行期不求值），把 sglang.kernels.ops.mamba.triton

_ops 导入包 try/except 并在失败时置 None，init_mamba_backend 增加 None 守卫，避免无 triton 环境下 mamba 后端初始化崩溃。

3. 契约测试与 CI 配套（test/registered/core/test_srt_empty_deps.py）：_parse_runtime_base 用 tomllib/tomli 解析 pyproject_other.toml 的 runtime_base 包名（剥离版本与 extras），test_runtime_base_no_torch_deps 断言与 TORCH_PULLING_PACKAGES 黑名单无交集，test_runtime_base_not_empty 做数量 sanity check。该测试最初注册到 NPU CI 导致失败，后续改为 register_cpu_ci(est_time=10, suite="base-a-test-cpu") 归属 CPU 套件。
4. 硬件验证：作者在 NVIDIA H20（8×144GB）上用 sglang-plugin-FL + FlagGems 验证 Qwen3.6-27B（Dense + FLA）与 Qwen3.6-35B-A3B（MoE 256 experts）的离线与 16 并发推理（文本 + VL），全部算子走 FlagGems Triton 或 reference 实现，零依赖 sgl_kernel/flashinfer。

关键文件：

- python/pyproject_other.toml（模块 依赖配置；类别 config；类型 configuration；符号 runtime_base, runtime_common, srt_empty）：本 PR 的核心契约所在：将 runtime_common 拆分为 runtime_base 与向后兼容的 runtime_common，并新增 srt_empty 安装组，直接决定哪些包会传递拉入 torch/triton；评审中的依赖解析与修复全部围绕此文件展开。
- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 dependency-wiring；符号 init_mamba_backend）：核心调度模块：移除顶层 CUDA/Triton 导入（CudaStream 移入 TYPE_CHECKING，triton_ops 加 try/except），并在 init_mamba_backend 增加 None 守卫，是无 triton 环境下调度器可导入的关键，评审专门讨论过此处。
- python/sglang/srt/server_args.py（模块 服务参数；类别 source；类型 dependency-wiring；符号 mamba_cache_chunk_size）：移除顶层 FLA_CHUNK_SIZE 导入（原会触发 import triton），改为 mamba_cache_chunk_size 属性内惰性导入 + 64 回退；涉及 mamba 缓存 chunk 计算正确性，评审专门讨论漂移风险。
- test/registered/core/test_srt_empty_deps.py（模块 依赖测试；类别 test；类型 test-coverage；符号 _parse_runtime_base, test_runtime_base_no_torch_deps, test_runtime_base_not_empty）：新增的依赖契约测试，守护 runtime_base 不混入 torch 系依赖；经历了 NPU CI 失败后重新注册到 CPU 套件，是评审建议的直接落地。

关键符号：mamba_cache_chunk_size, init_mamba_backend, _parse_runtime_base, test_runtime_base_no_torch_deps, test_runtime_base_not_empty

关键源码片段

python/pyproject_other.toml

本 PR 的核心契约所在：将 runtime_common 拆分为 runtime_base 与向后兼容的 runtime_common，并新增 srt_empty 安装组，直接决定哪些包会传递拉入 torch/triton；评审中的依赖解析与修复全部围绕此文件展开。

```
[project.optional-dependencies]
```

```
# runtime_base: torch-free 子集，供 srt_empty 在非 NVIDIA 平台做与设备无关的安装。
# 注意：不要在此追加依赖 torch / triton 的包，它们应放在 runtime_common。
runtime_base = [
  "aiohttp",
  "anthropic>=0.20.0",
  "apache-tvm-ffi",
  "av",
  "blobfile==3.0.0",
  "build",
  "datasets",
  "easydict",
  "einops",
  "fastapi",
  "gguf",
  "helion==1.4",
  "interegular",
  "IPython",
  "llguidance>=1.7.6,<2.0.0",
  "mistral_common>=1.11.5",
  "modelscope",
  "msgspec",
  "ninja",
  "numpy",
  "openai==2.6.1",
  "openai-harmony==0.0.4",
  "orjson",
  "packaging",
  "partial_json_parser",
  "pillow",
  "prometheus-client>=0.20.0",
  "psutil",
  "py-spy",
  "pybase64",
  "pydantic",
  "python-multipart",
  "pyzmq>=25.1.2",
  "requests",
  "scipy",
  "sentencepiece",
  "setproctitle",
  "smg-grpc-servicer>=0.5.0",
  "soundfile==0.13.1",
  "tiktoken",
  "tqdm",
  "transformers==5.12.1",
  "uvicorn",
  "uvloop",
  "xxhash",
]
```

```
# runtime_common: 向后兼容组，安装结果与拆分前完全一致（评审确认）。
runtime_common = [
  "sglang[runtime_base]",
  "compressed-tensors", # 传递依赖 torch>=2.10，会升级覆盖 vendor torch
  "outlines==0.1.11", # 直接依赖 torch
  "timm==1.0.16",
  "torchao==0.9.0",
  "xgrammar==0.2.1", # Linux x86_64 上强制拉 triton，与 triton-ascend 冲突
]

# srt_empty: 纯 Python 安装组，供 sglang-plugin-FL 等 OOT 插件在
# torch_npu / torch_musa 等厂商构建上作为基座使用。
# 用法：cp pyproject_other.toml pyproject.toml && pip install -e ".[srt_empty]"
srt_empty = ["sglang[runtime_base]"]
```

python/sglang/srt/managers/scheduler.py

核心调度模块：移除顶层 CUDA/Triton 导入（CudaStream 移入 TYPE_CHECKING，triton_ops 加 try/except），并在 init_mamba_backend 增加 None 守卫，是无 triton 环境下调度器可导入的关键，评审专门讨论过此处。

```
from typing import TYPE_CHECKING, Any, Deque, Dict, List, Optional, Tuple, Union

if TYPE_CHECKING:
    # CudaStream 仅用于类型注解、运行期不求值，因此只在类型检查时导入，
    # 避免在无 CUDA 环境下 from torch.cuda import Stream 直接失败。
    from torch.cuda import Stream as CudaStream

try:
    from sglang.kernels.ops.mamba.triton_ops import (
        initialize_mamba_selective_state_update_backend,
    )
except ImportError:
    # srt_empty 等无 triton 安装下，mamba 后端初始化被置空并安全跳过。
    initialize_mamba_selective_state_update_backend = None


def init_mamba_backend(self) -> None:
    # None 守卫：没有 triton 时既不导入也不调用，保证调度器可正常启动。
    if initialize_mamba_selective_state_update_backend is not None:
        initialize_mamba_selective_state_update_backend(self.server_args)
```

test/registered/core/test_srt_empty_deps.py

新增的依赖契约测试，守护 runtime_base 不混入 torch 系依赖；经历了 NPU CI 失败后重新注册到 CPU 套件，是评审建议的直接落地。

```
# 守护 runtime_base 的 torch-free 契约：任何新依赖只要出现在黑名单里，
# 本测试就会在 CPU CI（base-a-test-cpu）上失败，提示应放入 runtime_common。
TORCH_PULLING_PACKAGES = frozenset(
    {
```

```
        "torch",
        "torchao",
        "timm",
        "xgrammar",
        "compressed-tensors",
        "outlines",
        "flashinfer",
        "sgl-kernel",
    }
)


def _parse_runtime_base() -> set:
    """解析 pyproject_other.toml 中 runtime_base 的裸包名（剥离版本与 extras）。"""
    # 优先 tomllib（Python 3.11+），否则回退 tomli。
    try:
        import tomllib
    except ModuleNotFoundError:
        import tomli as tomllib # type: ignore[no-redef]

    toml_path = Path(__file__).resolve().parents[3] / "python" / "pyproject_other.toml"
    if not toml_path.exists():
        pytest.skip(f"pyproject_other.toml not found at {toml_path}")

    with open(toml_path, "rb") as f:
        data = tomllib.load(f)

    runtime_base = data["project"]["optional-dependencies"]["runtime_base"]

    # "package[extra]>=1.0,<2.0; marker" -> "package"
    pkg_names = set()
    for dep in runtime_base:
        name = (
            dep.split("[")[0]
            .split(">")[0]
            .split("<")[0]
            .split("=")[0]
            .split("!")[0]
            .split(";")[0]
            .strip()
        )
        pkg_names.add(name.lower())

    return pkg_names


def test_runtime_base_no_torch_deps():
    """runtime_base 不得包含任何会传递拉入 torch 的包。"""
    pkg_names = _parse_runtime_base()
```

```
    violations = pkg_names & TORCH_PULLING_PACKAGES
    assert not violations, (
        f"runtime_base contains torch-pulling packages: {sorted(violations)}. "
        f"Move them to runtime_common to keep srt_empty torch-free."
    )


def test_runtime_base_not_empty():
    """sanity check: 防止 TOML 结构调整导致 runtime_base 被误裁。"""
    pkg_names = _parse_runtime_base()
    assert len(pkg_names) >= 20, (
        f"runtime_base only has {len(pkg_names)} packages, expected >= 20. "
        f"Did the toml structure change?"
    )
```

评论区精华

最核心的交锋是 alexnails 对 runtime_base 的依赖解析 dive：他用 uv pip compile 解析 linux/x86_64 py3.10，发现最初版本仍会拉到 torch==2.13.0、triton==3.7.1 与 15 个 nvidia-* wheel（共 163 包），元凶是 xgrammar（依赖 torch>=1.10 且 Linux x86_64 上强制 triton，与 triton-ascend 等厂商分支冲突）、compressed-tensors（当前版要求 torch>=2.10，会主动升级覆盖 vendor torch）、outlines（直接依赖 torch）。作者将三包迁入 runtime_common 后，安装图降为 128 包、零 torch/triton/nvidia。alexnails 还指出 trade-off：默认 --grammar-backend 是 xgrammar 且 xgrammar_backend.py 模块级导入，因此 srt_empty 安装没有开箱即用的结构化输出，作者接受该取舍。CudaStream 回退实现上，gemini-code-assist 建议用 DummyCudaStream 占位类避免 isinstance 抛 TypeError，alexnails 则建议 TYPE_CHECKING（CudaStream 仅作注解、运行期不求值），最终采用 TYPE_CHECKING。FLA_CHUNK_SIZE 方面，alexnails 质疑 try/except 后会出现漂移，作者把导入挪进 mamba_cache_chunk_size 属性并保留 64 兜底，承认漂移风险仍在，抽共享 consts 文件留作 follow-up。测试建议上，alexnails 建议 pre-commit hook 或注册测试断言 runtime_base 无 torch，作者落地为 test_srt_empty_deps.py，完整传递依赖解析（uv pip compile）留作后续 CI 步骤。

- runtime_base 依赖污染：163 包含 torch 全家桶 (design): 作者将三包移入 runtime_common，runtime_base 收敛为 128 包零 torch/triton/nvidia；同时接受 srt_empty 无默认结构化输出后端的 trade-off
- CudaStream 回退应使用 TYPE_CHECKING 而非 None (design): 采用 if TYPE_CHECKING 方案，作者同步移除无用的 nullcontext 别名
- FLA_CHUNK_SIZE 惰性导入后的漂移风险 (correctness): 采纳惰性导入方案，硬编码 64 作为最后兜底；抽共享常量留作 follow-up，本 PR 未处理
- scheduler 顶层仍会 import triton (design): 将该导入包 try/except 并置 None，init_mamba_backend 增加 None 守卫
- 依赖契约测试与 dev_empty aggregate 建议 (testing): 作者新增 test_srt_empty_deps.py（包名黑名单检查）；完整传递解析（uv pip compile）留作后续 CI 步骤，dev_empty 未实现

- NPU CI 失败与测试注册归属 (testing): 注册到 CPU 套件解决 CI 失败，符合仓库 CI 分工约定

风险与影响

- 风险：
  - FLA_CHUNK_SIZE 回退漂移：server_args.py 硬编码 64 作为兜底，若上游 sglang.kernels.ops.attention.fla.chunk_delta_h 修改 CHUNK_SIZE，mamba 缓存 chunk 大小将静默不一致，影响 Mamba 系模型的缓存点计算，且不会显式报错。
  - mamba 后端静默降级：scheduler.py 中 triton_ops 导入失败时 initialize_mamba_selective_state_update_backend 被置 None，init_mamba_backend 直接跳过；若未来 mamba 模型在无 triton 环境运行，可能推理错误而非显式失败。
  - 测试覆盖局限：test_srt_empty_deps.py 只做顶层包名黑名单检查，不做传递依赖解析；新增未知包若间接拉取 torch 不会被拦截，这正是评审中 alexnails 建议 uv pip compile 的原因。
  - srt_empty 功能缺口：无 xgrammar/outlines 后结构化输出无默认后端，且 xgrammar_backend.py 模块级导入意味着导入该模块即失败，需要用户自行安装后端。
  - 兼容性回归：runtime_common 从平铺列表改为自引用 extra（sglang[runtime_base]），老旧 pip/setuptools 对 extras 自引用的解析行为可能存在差异，理论上同包不同解析路径。
  - 影响面：两份核心模块（server_args.py、scheduler.py）的导入路径变更，加上 CI 注册归属调整，涉及安装矩阵与回归验证。
- 影响：
  - 用户侧：非 NVIDIA 平台（Ascend、MUSA 等）用户可执行 cp pyproject_other.toml pyproject.toml && pip install -e ".[srt_empty]" 获得无 torch 的纯 Python 基座，再叠加 OOT 插件实现多芯片推理；NVIDIA 用户安装命令不变、安装结果不变，无感知。
  - 系统侧：安装矩阵新增一个 extra 组，CI 的 CPU 套件（base-a-test-cpu）新增一条依赖契约守护测试，后续向 runtime_base 追加依赖时会被自动拦截。
  - 团队与生态侧：为 BAAI FlagOS 插件生态（sglang-plugin-FL、FlagGems、FlagCX）提供安装级入口，显著降低跨芯片支持的门槛；同时带来 runtime_base 白名单的持续维护成本与依赖治理责任。
  - 影响程度：属于基础设施类变更，对核心运行时行为无侵入，但对打包与安装路径的影响是长期的。
  - 风险标记：依赖契约变更，FLA_CHUNK_SIZE 回退值漂移风险，srt_empty 无结构化输出后端，核心模块导入路径调整，依赖检查仅浅层包名匹配

关联脉络

- PR #33403 [Scheduler] Honor explicit min-free-slots thresholds: 同改 python/sglang/srt/server_args.py，同一条核心配置文件的演进脉络；后续改动需注意与本 PR 惰性导入区域的交互。

- PR #33595 *Measure prefill busy time between launches*: 同改 python/sglang/srt/managers/scheduler.py，且同为调度器导入与初始化路径调整，展示该核心文件的持续演进。