# PR #22218 完整报告

sgl-project/sglang

[Experimental] Breakable Piecewise Cuda Graph

合并时间：2026-04-24 19:33

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/22218

## 执行摘要

- 一句话：实现不依赖 torch.compile 的可打断 CUDA 图（BCG）
- 推荐动作：该 PR 值得所有关注调度优化的开发者精读，尤其是 breakable_cuda_graph_runner.py 和 breakable_cuda_graph.py 中的图捕获与回放机制。设计上使用 contextvar 和函数装饰器实现图打断，比基于 FX 的路径更直观且易调试。但在生产环境中启用前，应在目标模型架构上进行充分的性能验证，并留意弱引用张量的边界问题。

## 功能与动机

受 #19102 和 @cctry 工作的启发，旨在提供一种更简单的 piecewise CUDA graph 方案，避免对 torch.compile 后端和 FX 图追踪的依赖，降低维护复杂度并提高可调试性。

## 实现拆解

1. 新增 Breakable CUDAGraph 核心库(breakable_cuda_graph/breakable_cuda_graph.py)：定义 BreakableCUDAGraph 类，用 _current_capture_var ContextVar 追踪当前捕获上下文；eager_on_graph 装饰器在函数调用处打断当前 CUDA 图并结束该段，开始新段。每个段都是真实的 torch.cuda.CUDAGraph，通过共享内存池的 use_count 管理生命周期，使得弱引用张量在回放时有效。
2. 新增独立上下文管理模块(breakable_cuda_graph/context.py)：提供 enable_breakable_cuda_graph 上下文管理器和 is_in_breakable_cuda_graph 查询，与 torch.compile 的 piecewise 上下文分离，避免耦合。
3. 创建 BreakableCudaGraphRunner(breakable_cuda_graph_runner.py)：独立于 PiecewiseCudaGraphRunner，但复用其 replay_prepare 方法。实现 _warmup、_capture_all、can_run 等关键方法。捕获时调用 BreakableCUDAGraphCapture 上下文管理器，并在每个 attention 层自动调用 eager_on_graph 包装的函数来打断图。
4. 修改 radix_attention.py 和 nemotron_h.py：在 RadixAttention.forward 中根据是否启用 BCG 分派到 bcg_unified_attention_with_output（通过 eager_on_graph(True) 包装）。类似地在 NemotronH Mamba2 层添加 breakable_nemotron_mamba2_with_output。
5. 修改 model_runner.py 和 server_args.py：在 init_piecewise_cuda_graphs 中根据 enable_breakable_cuda_graph 参数选择使用 BreakableCudaGraphRunner；新增 --enable-breakable-cuda-graph 命令行参数。

6. 测试和 CI 配置：将测试文件从 test/registered/cuda_graph/ 迁移到 test/registered/breakable_cuda_graph/，新增集成测试（Qwen3-8B + mgsm_en 精度测试）并注册为 large CI 套件。

关键文件：

- python/sglang/srt/model_executor/breakable_cuda_graph_runner.py（模块 调度器；类别 source；类型 data-contract；符号 BreakableCudaGraphRunner, init, _init_buffers, _run_forward）：新增的主要 Runner，封装了 BCG 的 warmup、capture、replay 全流程，与 PCG 平级。
- python/sglang/srt/model_executor/breakable_cuda_graph/breakable_cuda_graph.py（模块 基础库；类别 source；类型 data-contract；符号 GraphBreakInfo, _end_capture_segment, _begin_capture_segment, _instantiate_graph）：新一代 Piecewise CUDA Graph 基础设施，定义 BreakableCUDAGraph 类、eager_on_graph 装饰器以及捕获上下文。
- python/sglang/srt/model_executor/breakable_cuda_graph/context.py（模块 基础库；类别 source；类型 data-contract；符号 is_in_breakable_cuda_graph, enable_breakable_cuda_graph）：独立上下文管理，避免与 PCG 上下文耦合。
- test/registered/breakable_cuda_graph/test_breakable_cuda_graph.py（模块 测试；类别 test；类型 rename-or-move；符号 TestBreakableCudaGraph, setUpClass, tearDownClass, test_gsm8k_accuracy）：迁移并增强的测试文件，包含单元测试和集成测试。
- python/sglang/srt/models/nemotron_h.py（模块 模型适配；类别 source；类型 data-contract）：为 Mamba2 层添加 breakable 支持。
- python/sglang/srt/layers/radix_attention.py（模块 注意力层；类别 source；类型 dependency-wiring）：注意力层增加 BCG 分派分支，是图打断的关键点。
- python/sglang/srt/model_executor/model_runner.py（模块 模型执行器；类别 source；类型 data-contract）：条件初始化 BCG Runner。
- python/sglang/srt/server_args.py（模块 配置入口；类别 source；类型 core-logic）：新增命令行参数控制 BCG 启用。

关键符号：BreakableCudaGraphRunner.init, BreakableCudaGraphRunner._warmup, BreakableCudaGraphRunner._capture_all, BreakableCudaGraphRunner.can_run, BreakableCUDAGraph.replay, BreakableCUDAGraphCapture.enter, BreakableCUDAGraphCapture.exit, eager_on_graph, enable_breakable_cuda_graph, is_in_breakable_cuda_graph, bcg_unified_attention_with_output, breakable_nemotron_mamba2_with_output

## 关键源码片段

### python/sglang/srt/model_executor/breakable_cuda_graph_runner.py

新增的主要 Runner，封装了 BCG 的 warmup、capture、replay 全流程，与 PCG 平级。

```
# breakable_cuda_graph_runner.py — BCG Runner 核心
```

```
class BreakableCudaGraphRunner:
    """不依赖 torch.compile 的可打断 CUDA Graph Runner。"""

    # 复用 PCG 的 replay_prepare 方法，避免复制
    replay_prepare = PiecewiseCudaGraphRunner.replay_prepare

    def __init__(self, model_runner: ModelRunner):
        self.model_runner = model_runner
        self.device = model_runner.device
        self.capture_num_tokens = sorted(model_runner.server_args.piecewise_cuda_graph_
        tokens)
        self.graphs: dict[int, BreakableCUDAGraph] = {}
        self._init_buffers(model_runner) # 初始化需要的静态张量

    def _init_buffers(self, model_runner: ModelRunner):
        """创建用于捕获的静态输入缓冲区。"""
        # 每个不同 token 数对应一组 dummy 输入（位置编码、attention mask 等）
        # 这些缓冲区在捕获时用作固定输入
        # 代码略 ...

    def _warmup(self):
        """运行模型 warmup（不包括图捕获）。"""
        # 执行几次普通前向，触发热加载
        # 代码略 ...

    def _capture_all(self):
        """按预设 token 数量逐一捕获图。"""
        for num_tokens in self.capture_num_tokens:
            forward_batch = self._build_capture_forward_batch(num_tokens)
            graph = BreakableCUDAGraph()
            capture_stream = self.device_module.Stream(self.device)
            with enable_breakable_cuda_graph(): # 启用 BCG 上下文
                with BreakableCUDAGraphCapture(graph, stream=capture_stream):
                    with graph_capture(device=self.device): # 开始 CUDA 图捕获
                        self._run_forward(forward_batch) # 执行一次完整前向
            self.graphs[num_tokens] = graph # 保存

    def can_run(self, forward_batch: ForwardBatch) -> bool:
        """判断当前 forward_batch 是否可以用已捕获的图执行。"""
        if forward_batch.input_ids is None:
            return False
        num_tokens = forward_batch.input_ids.shape[0]
        idx = bisect.bisect_left(self.capture_num_tokens, num_tokens)
        if idx == len(self.capture_num_tokens):
            return False
        return True
```

python/sglang/srt/model_executor/breakable_cuda_graph/breakable_cuda_graph.py

新一代 Piecewise CUDA Graph 基础设施，定义 BreakableCUDAGraph 类、*eager_on_graph* 装饰器以及捕获上下文。

```
# breakable_cuda_graph.py — 核心图段管理

import logging
import threading
from contextvars import ContextVar
from typing import Any, Callable, Optional

import torch

try:
    from cuda.bindings import runtime as rt
except ImportError:
    rt = None

logger = logging.getLogger(__name__)

# 当前线程活跃的 BreakableCUDAGraphCapture 上下文
_current_capture_var: ContextVar[Optional["BreakableCUDAGraphCapture"]] = ContextVar(
    "current_capture", default=None
)
_current_stream_var: ContextVar[Optional[torch.cuda.Stream]] = ContextVar(
    "current_stream", default=None
)
_forked_streams_var: ContextVar[Optional[set[torch.cuda.Stream]]] = ContextVar(
    "forked_streams", default=None
)

class BreakableCUDAGraph:
    """管理多个 CUDA Graph 段的容器。每个段都是一个独立的 `torch.cuda.CUDAGraph`。"""
    def __init__(self):
        self.graphs: list[torch.cuda.CUDAGraph] = []
        self.streams: list[torch.cuda.Stream] = []
        self.pool: Optional[torch.cuda.caching_allocator.CUDAPlacedAllocator] = None
        # ... 其他字段

    def _begin_capture_segment(self, stream: torch.cuda.Stream):
        """开始一个新段的捕获。"""
        if self.pool is None:
            self.pool = torch.cuda.caching_allocator.CUDAPlacedAllocator()
        self.pool.beginAllocateToPool()
        self.streams.append(stream)

    def _end_capture_segment(self, stream: torch.cuda.Stream):
        """结束当前段，自动 join 任何分叉流。"""
        forked = _forked_streams_var.get()
        if forked:
```

```
            for s in forked:
                if _is_capturing(s.cuda_stream):
                    torch.cuda.synchronize(s)
            _forked_streams_var.set(set())
        self.pool.endAllocateToPool()

class BreakableCUDAGraphCapture:
    """上下文管理器：进入时开始第一个捕获段，退出时结束最后一个段。"""
    def __init__(self, graph: BreakableCUDAGraph, stream: torch.cuda.Stream):
        self.graph = graph
        self.stream = stream

    def __enter__(self):
        self._prev = _current_capture_var.get()
        _current_capture_var.set(self)
        _current_stream_var.set(self.stream)
        self.graph._begin_capture_segment(self.stream)
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        if exc_type is None:
            self.graph._end_capture_segment(self.stream)
        _current_capture_var.set(self._prev)
        _current_stream_var.set(None)

def eager_on_graph(break_point: bool = True):
    """装饰器：在捕获模式下，函数调用会结束当前段并开始新段。"""
    def decorator(fn: Callable) -> Callable:
        @functools.wraps(fn)
        def wrapper(*args, **kwargs):
            capture = _current_capture_var.get()
            if capture is None:
                return fn(*args, **kwargs)
            capture.graph._end_capture_segment(capture.stream)
            try:
                result = fn(*args, **kwargs)
            finally:
                capture.graph._begin_capture_segment(capture.stream)
            return result
        return wrapper
    return decorator if break_point else (lambda fn: fn)
```

# 评论区精华

Review 中最核心的讨论集中在以下几方面：

- 参数设计：merrymercy 指出 enable_breakable_cuda_graph 单一参数无法灵活组合解环 (prefill) 和解码 (decode) 的模式，建议引入 --cuda-graph-mode 字典或分离参数。

- 继承关系：merrymercy 和 cctry 均建议不要再从 PiecewiseCudaGraphRunner 继承，最终实现完全独立。
- radix_attention 复杂度：ispobock 和 cctry 认为 forward 中 BCG/PCG 的分支逻辑使代码复杂，需设计更通用的函数处理 MLA/MHA 差异；merrymercy 建议将处理函数抽离为 processing_mla。
- 弱引用张量安全性：frgossen 提出弱引用张量在共享内存池地址复用可能导致数据损坏，cctry 承认风险存在，表示实际使用中未传递弱引用跨段。
- 本地脚本：ispobock 要求删除本地测试脚本 run_bcg_comparison.sh，Oasis-Git 已移除。
- 参数配置设计改进 (design): 未解决，当前仅实现单一开关，后续 PR 可能改进。
- 继承关系与代码复用 (design): 已采纳：去掉继承，改为组合式复用。
- radix_attention 分支复杂化 (design): 部分解决：BCG 相关逻辑通过 eager_on_graph 和条件分支保持在内，但重构到独立函数的建议未完全采纳。
- 弱引用张量安全性问题 (correctness): 已知风险，当前实现通过不跨段传递弱引用避免，但长期需更安全方案。
- 本地测试脚本清理 (other): 已删除。

## 风险与影响

- 风险：
  - 弱引用张量数据安全性：如 frgossen 指出的，弱引用张量可能在池地址复用后指向无效数据，需要确保跨回放段不传播弱引用。（见文件 breakable_cuda_graph.py）
  - 性能退化风险：新增的 BCG 路径在 attention 层引入了上下文切换和段捕获开销，虽基准测试与 PCG 持平，但更多模型架构未验证。（radix_attention.py 中的新分支）
  - 与现有 PCG 兼容性：同时维护两套 piecewise 图机制可能引起配置和 bug 修复同步的负担。（model_runner.py 中条件分支）
  - 依赖 'cuda-python'：BCG 核心使用 cuda.bindings，若未安装则抛出 ImportError，这增加了一个可选但非无成本的依赖。
  - 启动日志过多：merrymercy 指出每个批次大小都打印日志可能使启动输出冗长。（breakable_cuda_graph_runner.py 中的 tqdm/logging）
- 影响：
  - 开发者：新增的实验性特性使社区可以尝试不使用 torch.compile 的 Piecewise CUDA Graph，降低门槛。但需要维护两人机制，增加了理解和贡献的复杂度。
  - 用户：通过 --enable-breakable-cuda-graph 尝试 BCG，可获得的性能与其环境有关。默认不启用，不影响现有工作流。
  - 系统：在模型加载时多一次捕获过程，消耗额外内存和时间，但回放时无明显开销。测试套件增加了约 130s 的 CI 时间。
  - 团队：需要持续关注两个 piecewise 图路径的同步演进，避免功能偏差。
  - 风险标记：弱引用张量数据安全性，性能未充分验证，与 PCG 双机制维护负担，依赖 cuda-python 包，启动日志冗长

## 关联脉络

- PR #19102 Original breakable CUDA graph proposal: 本 PR 直接受 #19102 启发，credit to @cctry，是该思想的具体实现。
- PR #25110 Fix BCG wrap for RadixLinearAttention: zminglei 在评论中指出 BCG 遗漏了 RadixLinearAttention 的包装，并提交修复。
- PR #22427 Score API PR: 提交 d49982f 显示 BCG 需要设置 capture_return_pooled_hidden_states 以兼容 Score API。