

PyPTO 开源介绍

作者：王子楠

时间：2025/12/25

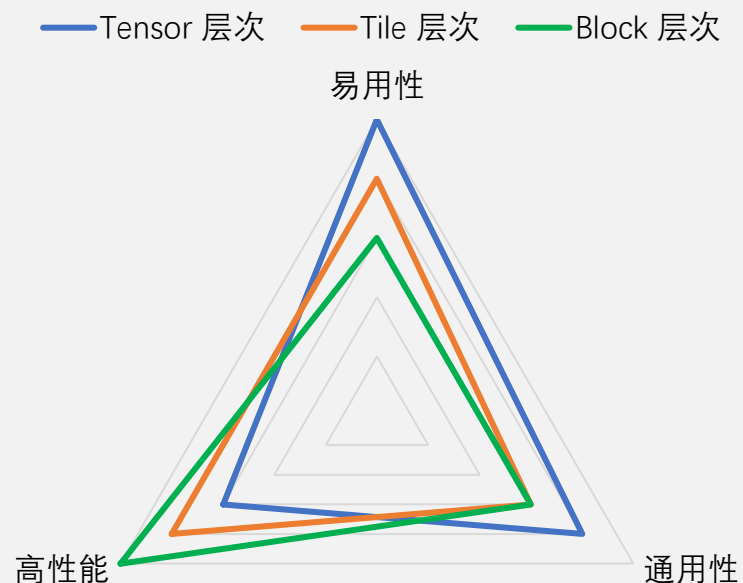
<https://gitcode.com/cann>

CANN

PyPTO(Python Parallel Tensor/Tile Operation) 简介

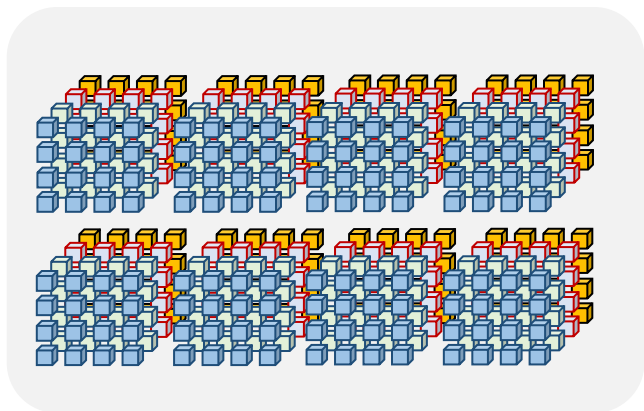
楔子：AI 编程领域的不可能三角

PyPTO 通过分层抽象解决不可能三角



PyPTO 支持不同层次抽象混合执行

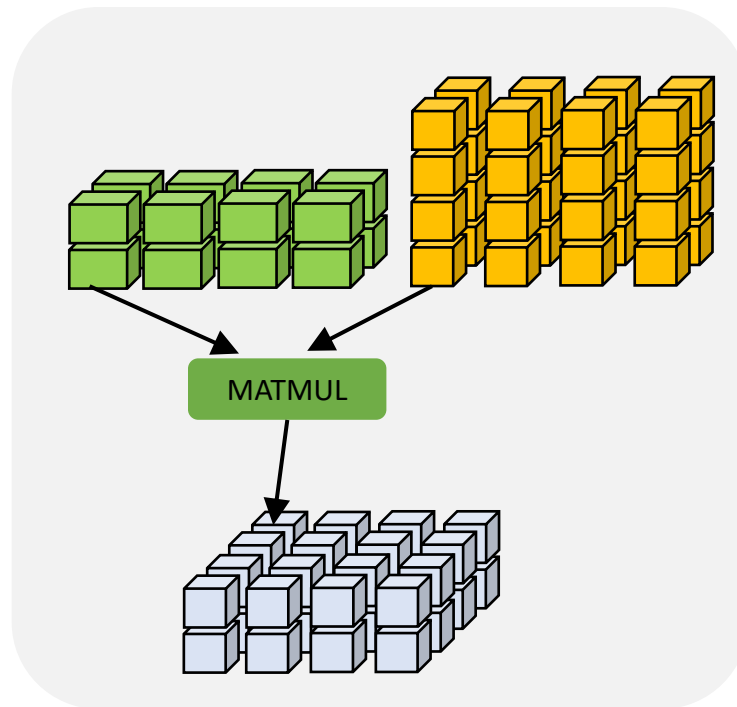
选择基于 Tile(smaller Tensor) 编程模型的原因



Scalar Operation (32bit)

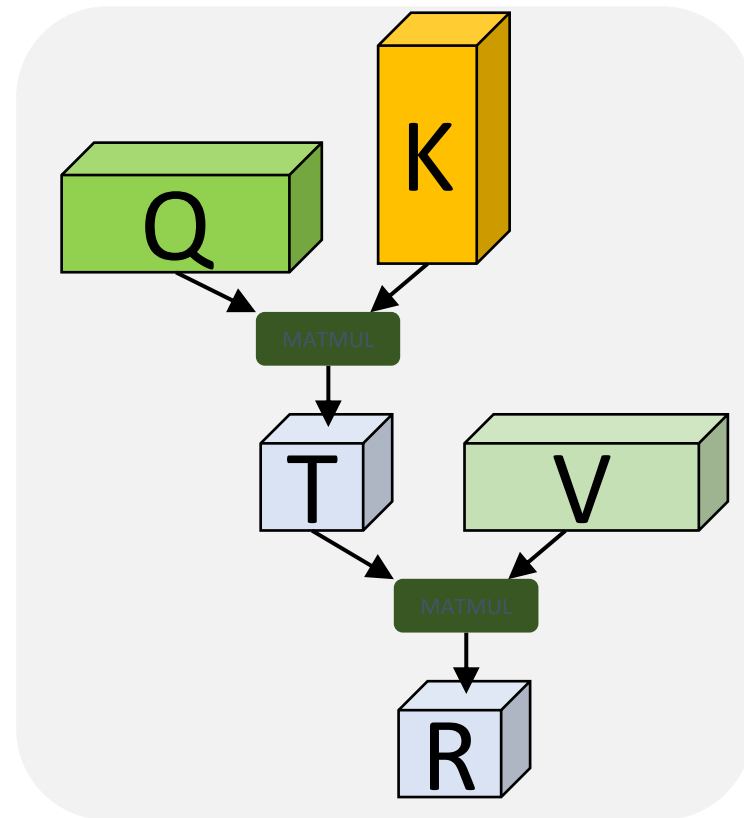
- ✓ 灵活性高
- ✗ 控制复杂度高，硬件实现复杂
- ✗ 计算冗余，效率不高

<https://gitcode.com/cann>



Tile Operation (16KB)

- ✓ 适配硬件存储层级
- ✓ 刚好适配核内Buffer存储
- ✓ 抽象简单 控制适中



Tensor Operation (256MB)

- ✓ 表达简单，python编程
- ✗ 数据量过大，塞不进芯片

PyPTO：基于 Tile 的编程与白盒优化

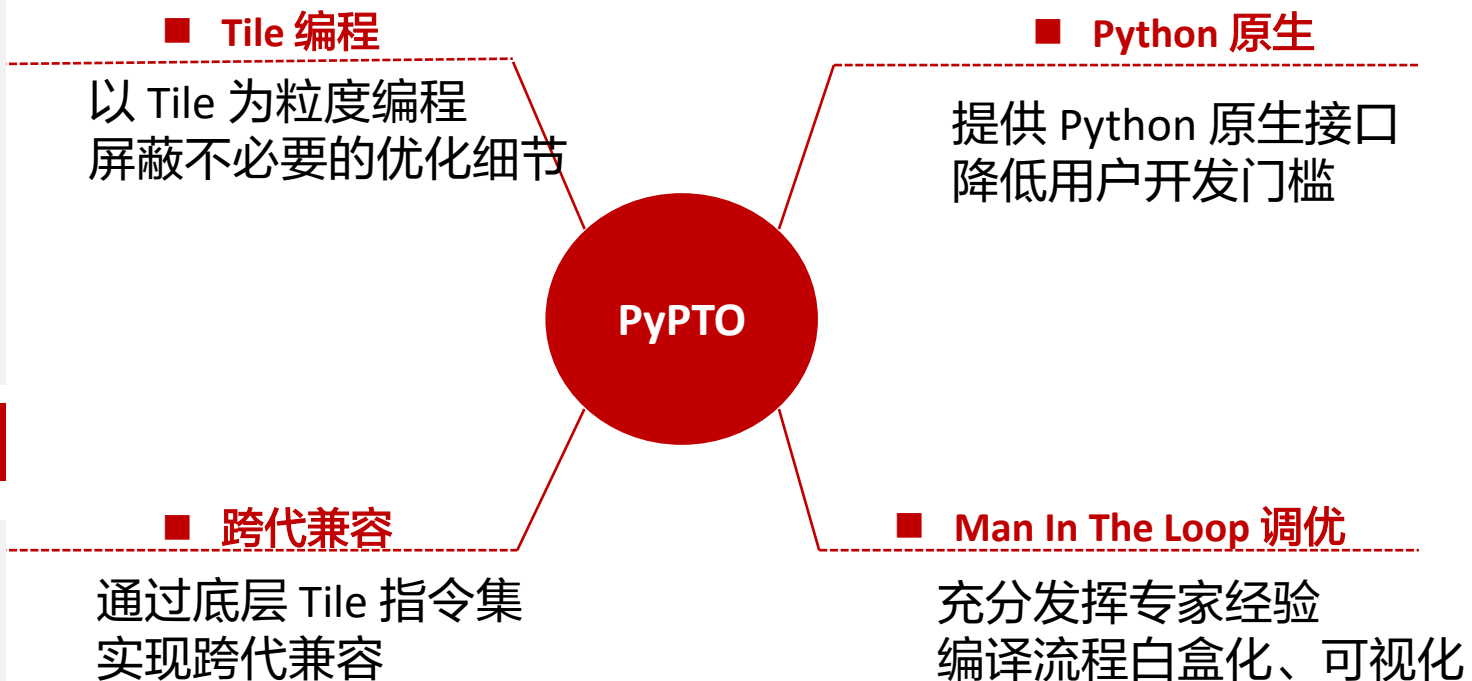
PyPTO 是什么？

- **本质：**面向 AI 加速器的高性能编程框架
- **目的：**简化算子开发流程，同时保持高性能计算能力
- **范式：**创新的 PTO 编程
- **理念：**基于 Tile 的编程模型

PyPTO 的定位

- 不仅限于单个算子的开发，更强调**复杂融合算子，甚至于整个模型网络**的开发能力
- 已有编程框架的**补充而非替换**，特别适合需要深度性能优化和细粒度控制的场景

<https://gitcode.com/cann>



PyPTO 的核心价值与目标用户

PyPTO 的核心价值

- ❑ **降低开发门槛：**算法开发者直接使用 PyPTO 实现高性能算子，无需深入了解底层硬件实现细节。
- ❑ **提升开发效率：**Tensor 级别抽象贴近数学表达式，使得开发者能够快速获得性能良好的初始实现。
- ❑ **保留优化空间：**通过多层级 IR 和 Pass 系统，PyPTO 支持从快速可用到极致性能的平滑过渡。
- ❑ **分层抽象设计：**PyPTO 对不同开发者暴露不同抽象层次，平衡编程简单性和开发者的控制力。
- ❑ **完整工具链：**PyPTO 提供可视化、调试、性能分析一体化支持，帮助开发者快速定位问题、理解性能瓶颈，并实现深度定制。

PyPTO 的目标用户群体

- ❑ **算法开发者：**使用 Tensor 层次编程，快速实现和验证算法。用户专注于算法逻辑，无需关心底层硬件细节，即可获得性能良好的实现。
- ❑ **性能优化专家：**可使用 Tile 或 Block 层次，根据用户对抽象层次和控制力需求而定，进行深度性能调优。用户可以在保持易用性的同时，获得对硬件资源的细粒度控制。
- ❑ **系统开发者：**可在 Tensor/Tile/Block 和 PTO 虚拟指令集层次上进行三方框架对接或集成，和工具链开发。用户利用 PyPTO 的多层次抽象，实现灵活的框架集成和扩展。
- ❑ **研究人员：**探索新的编程模型和优化技术。PyPTO 的模块化设计和开放的 IR 系统为研究提供了良好的实验平台。

PyPTO 产生的背景和动机—— AI 加速器编程的挑战



AI 加速器编程的挑战

传统算子开发的复杂性

算法开发与算子开发的鸿沟

硬件多样性与编程抽象

性能优化与开发效率



当前核心难点

理解算子的数学计算属性
&考虑硬件友好的执行方式

算法开发专注于算法逻辑和数学表达
&算子开发要将算法映射到硬件执行

高层抽象难以充分利用硬件特性
&底层抽象显著增加开发复杂度

性能优化需要深入的专业知识和大量的调优工作
&快速迭代和验证算法需要高效的开发工具

PyPTO 解决的核心问题——聚焦易用性和高性能

消除算法与算子开发的鸿沟

- 通过 Tensor 级别抽象，算法开发者可以直接实现高性能算子，无需等待专门的算子开发团队。
- 这消除了传统分工模式带来的沟通成本和迭代延迟。

平衡编程简单性和控制力

- 通过分层抽象设计，对不同开发者暴露不同抽象层次。
- 算法开发者可以使用高层次的 Tensor API，而性能优化专家和系统开发者可以深入到 Tile、Block 甚至虚拟指令集层次。

支持快速迭代和深度优化

- 提供“快速可用 → 灵活调优 → 深度优化”的平滑过渡。大多数开发者可以快速获得可用的实现。
- 而性能敏感的开发者的可以通过工具链进行深度优化，追求极致性能。

提供完整的工具链支持

- 可视化、调试、性能分析一体化支持，帮助开发者快速定位问题、理解性能瓶颈，并实现深度定制
- 可视化工具和分析工具，高质量示例和文档，与主流 IDE 和构建系统的集成，这些都是生态成功的关键因素。

PyPTO 核心设计原则

■ 分离关注点

清晰分层（计算/编译/执行），各层独立优化，降低耦合。

■ 渐进式抽象

Tensor → Tile → Block → Execution → Instruction 逐级 lowering，兼顾易用性与控制力

■ 显式优于隐式

计算、内存、依赖关系全部显式表达，提升可调试性与优化透明度。

■ 硬件感知

显式管理内存层次与执行单元，充分释放硬件性能，而非隐藏细节。

■ 可扩展性优先

模块化、可插拔设计，支持新硬件、新优化 Pass、新工具的快速集成。

■ 易用性优先

降低认知与使用门槛，友好的编程模型与工具链是生态繁荣的基石

六大
设计原则

PyPTO 编程范式设计

PTO 编程范式

- 使用 Tensor 作为数据的基本表达方式，通过一系列对 Tensor 的基本运算来描述并组装完整的计算流程。

声明式编程

- 开发者只需描述"做什么"而非"怎么做"。这种编程方式让开发者能够专注于算法逻辑，而将优化和执行交给框架处理

多层次抽象设计

- 框架既能够为大多数开发者提供易用的高层抽象，又能够为专家用户提供细粒度的控制能力

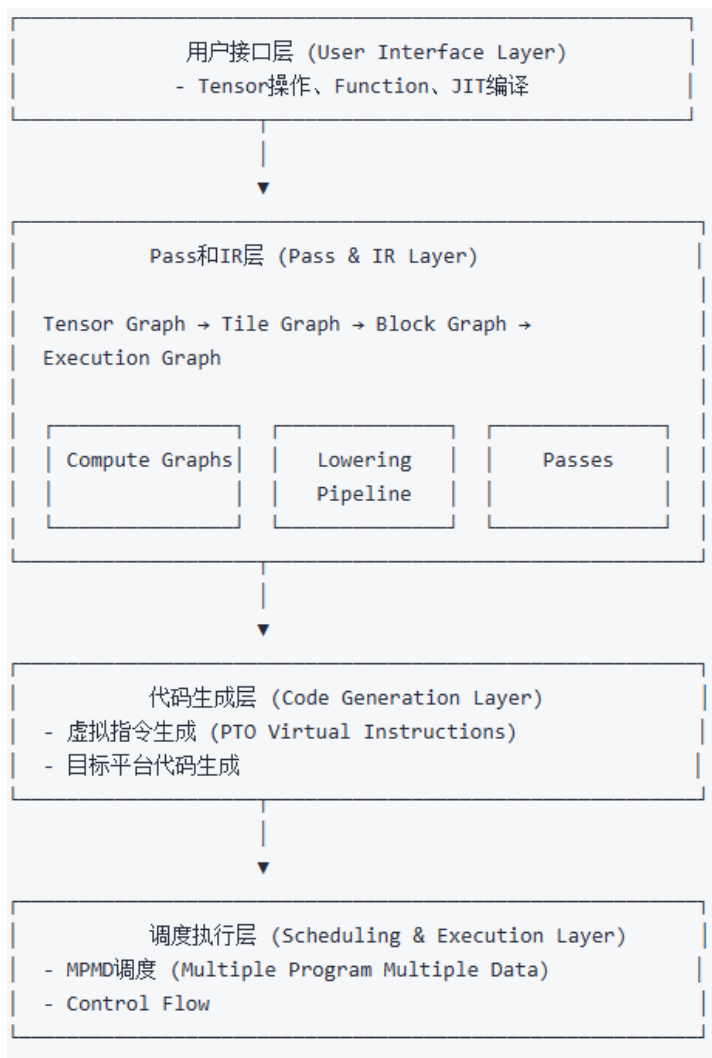
计算图驱动

- 计算图驱动的执行模型，通过构建计算图，框架可以自动进行优化、调度和执行。

Python 化设计

- 开发者可以通过 Python API 进行编程，而且可以在 Python 域对框架能力作自定义和扩展。

PyPTO 核心架构



PyPTO 框架与开发者交互的接口层
开发者直观表达计算逻辑

多层级 IR 设计
支持从高到低多个抽象层次表示

将优化后的 IR 转换为目标平台的可执行代码

负责将可执行代码在设备上调度执行

PyPTO 用户接口层

通过 frontend.jit 装饰器构建 PyPTO 函数

通过编译配置参数控制编译优化

控制流

for if else

Machine

承载

AICPU

HOST CPU

计算流

+ - x ÷

PASS

承载

AICORE

执行

根据算子数学表达式及计算逻辑，实现算子函数代码（**硬件无关**）

```
@pypto.frontend.jit()
def softmax(
    input_tensor: pypto.Tensor((b, n1, n2, dim), pypto.DT_FP32),
) -> (
    pypto.Tensor((b, n1, n2, dim), pypto.DT_FP32)
):
    """
    Softmax implementation with dynamic batch size support.

    This function processes input tensors in batches, applying softmax
    to each batch independently. The batch dimension is marked as dynamic,
    allowing variable batch sizes at runtime.

    Parameters
    -----
    input_tensor : pypto.Tensor
    shape [batch, n1, n2, dim]
    """
    output_tensor = pypto.tensor((b, n1, n2, dim), pypto.DT_FP32)
    tile_b = 1 # Process one batch at a time
    b_loop = b // tile_b

    # Tiling shape setting for efficient execution
    pypto.set_vec_tile_shapes(1, 4, 1, 64)

    for idx in pypto.loop(0, b_loop, 1, name="LOOP_L0_bIdx", idx_name="idx"):
        b_offset = idx * tile_b
        b_offset_end = (idx + 1) * tile_b

        # Extract batch slice
        input_view = input_tensor[b_offset:b_offset_end, :n1, :n2, :dim]

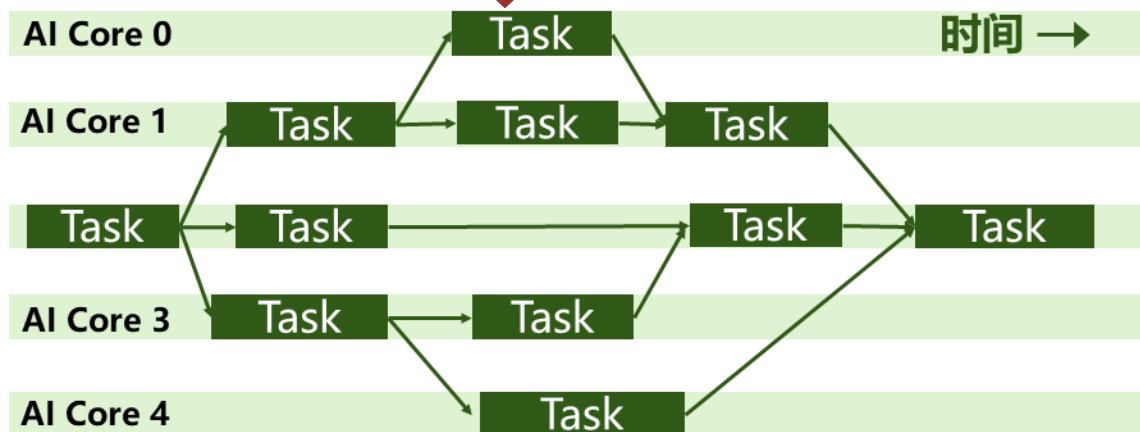
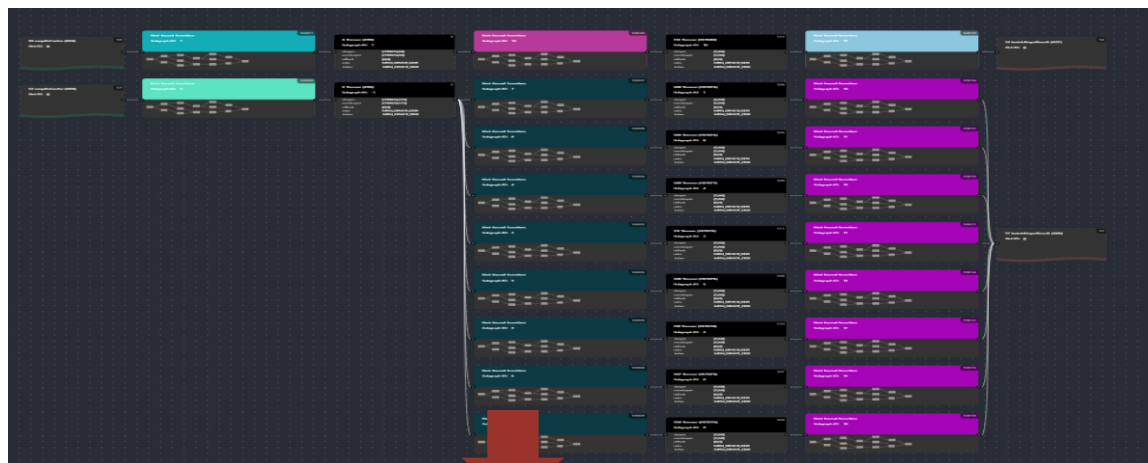
        # core softmax formula
        row_max = pypto.amax(input_view, dim=-1, keepdim=True)
        sub = input_view - row_max
        exp = pypto.exp(sub)
        esum = pypto.sum(exp, dim=-1, keepdim=True)
        softmax_out = exp / esum

        # Assemble result back to output tensor
        pypto.assemble(softmax_out, [b_offset, 0, 0, 0], output_tensor)

    return output_tensor
```

PyPTO 编译与调优白盒化

- 通过数据形成任务之间的依赖，可以通过拓扑序解依赖



多核泳道图

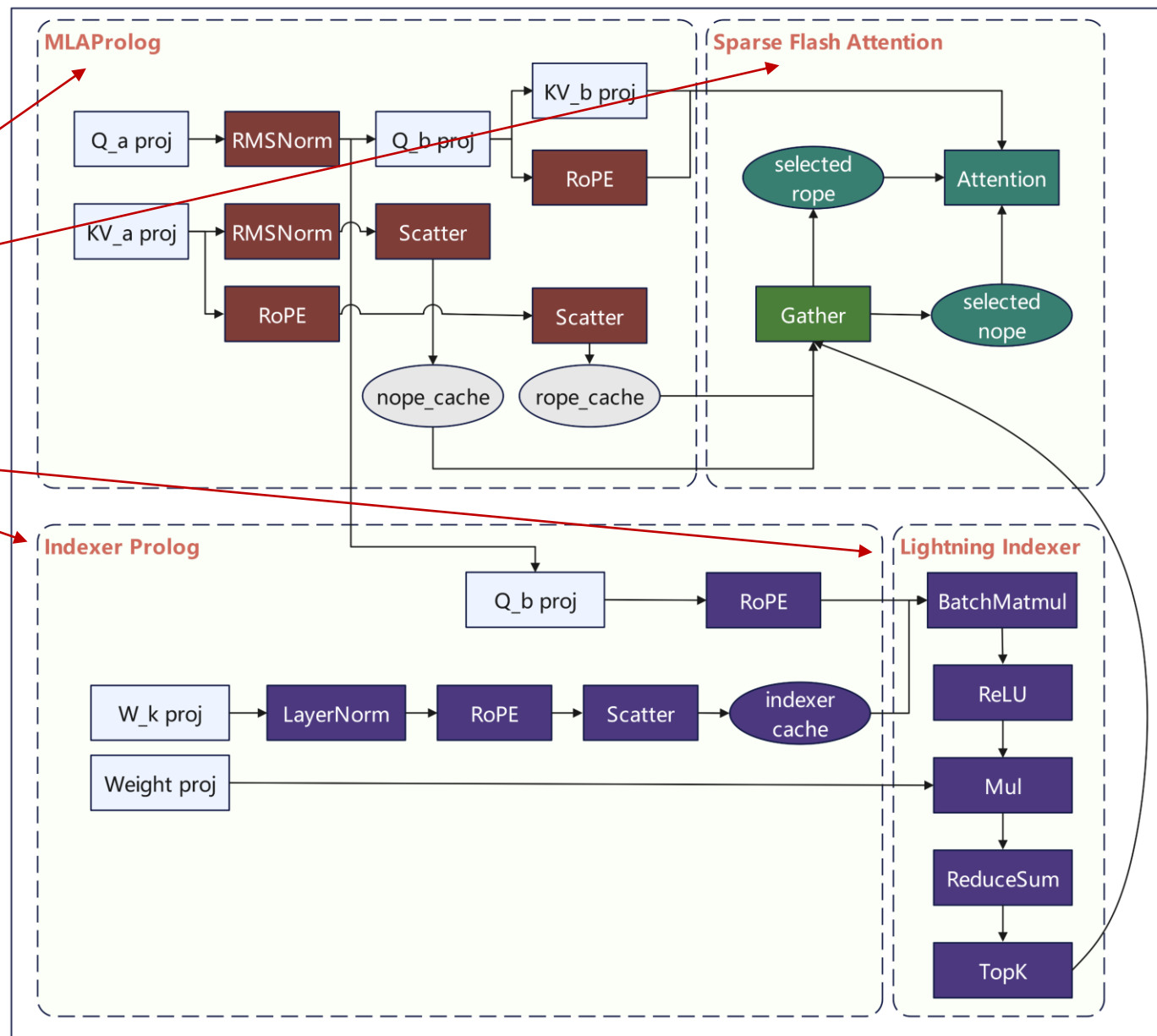
<https://gitcode.com/cann>

- 采集上板数据，形成泳道图；
- 不同颜色是不同子图，MPMD 调度



DeepSeek V3.2-Exp 开发：自主控制融合范围

将 Attention 层拆解为 4 个子图并行开发；
通过框架融合成一个算子

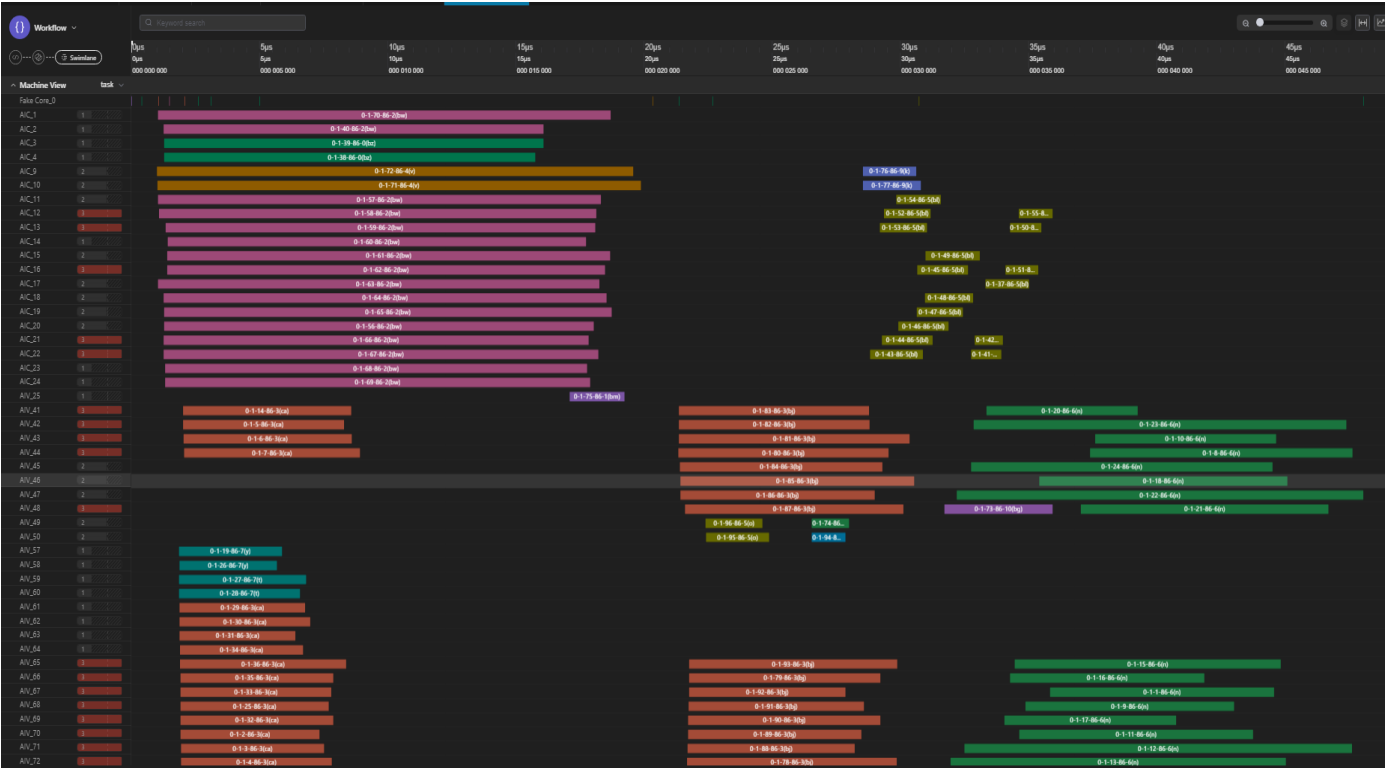


Indexer Prolog 调优实例

类别	Indexer Prolog 算子组成
PyPTO	1 / 个
	IndexerProlog
小算子	9 / 个
	QuantBatchMatmul, SplitVD, MatMul, InterleaveRope, ConcatV2D, Cast, LayerNorm, DynamicQuant, ScatterNdUpdate

计算流呈现如下特点:

- ✓ Q, Cache 和 Weight 计算互相独立且串行;
- ✓ Q 耗时最长, 可以掩盖其他两部分耗时



PyPTO 当前实现

功能集

用户 API

核心功能

编译优化

工具链

主要功能

(i) Python API; (ii) Tensor 操作; (iii) Function 定义;
(iv) JIT 编译; (v) 配置接口; (vi) 动态 Shape 相关支持

(i) Tensor 层次编程; (ii) 动态形状支持;
(iii) 符号化编程; (iv) PyTorch 集成;

(i) Tensor Graph; (ii) Tile Graph; (iii) Block Graph
各层级别优化 Passes 初步实现

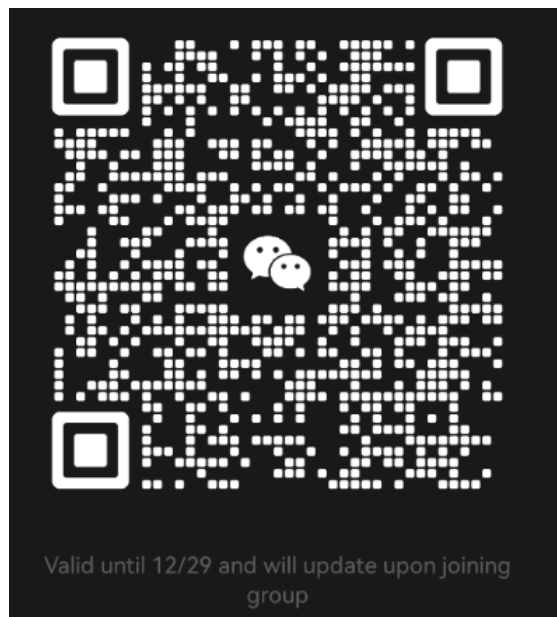
(i) 计算图可视化; (ii) 泳道图;
(iii) 编译中间产物 Dump; (iv) IDE 集成支持

PyPTO Roadmap



欢迎加入我们

PyPTO 技术交流群



PyPTO 开源仓



<https://gitcode.com/cann>

Thank you.

社区愿景：打造开放易用、技术领先的AI算力新生态

社区使命：使能开发者基于CANN社区自主研究创新，构筑根深叶茂、跨产业协同共享共赢的CANN生态

Vision: Building an Open, Easy-to-Use, and Technology-leading AI Computing Ecosystem

Mission: Enable developers to independently research and innovate based on the CANN community and build a win-win CANN ecosystem with deep roots and cross-industry collaboration and sharing.



上CANN社区获取干货



关注CANN公众号获取资讯