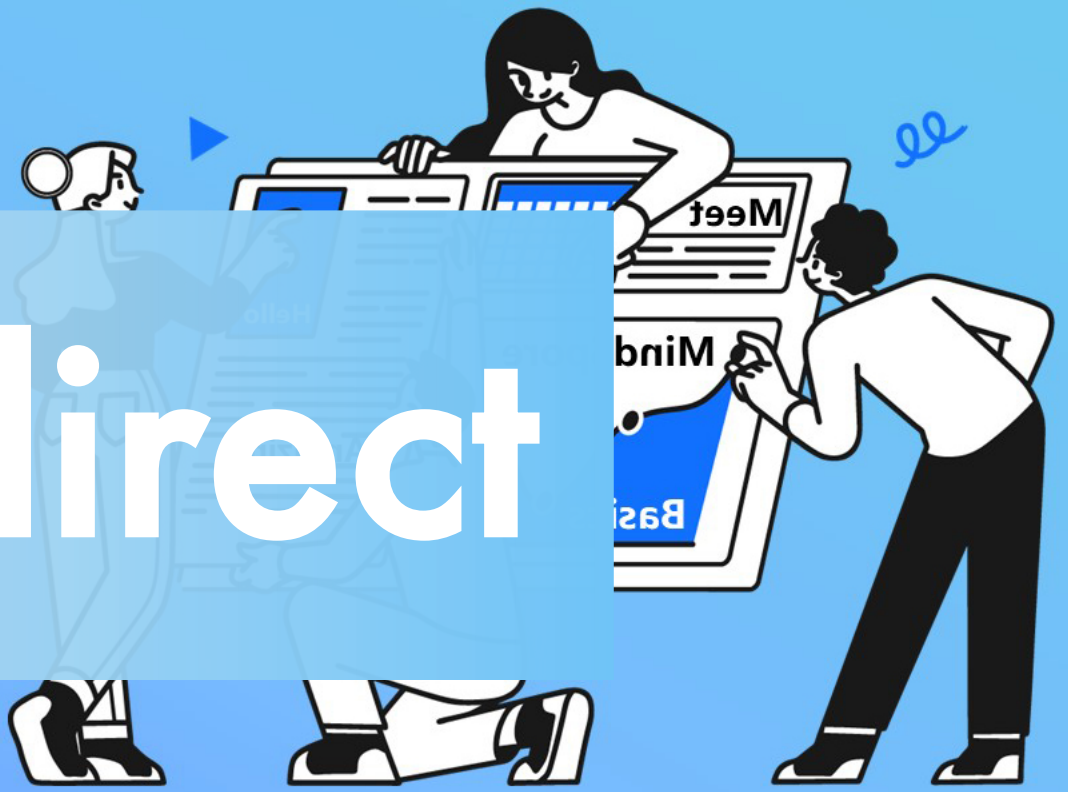
# 推理引擎-Kernel优化

# 间接优化Indirect

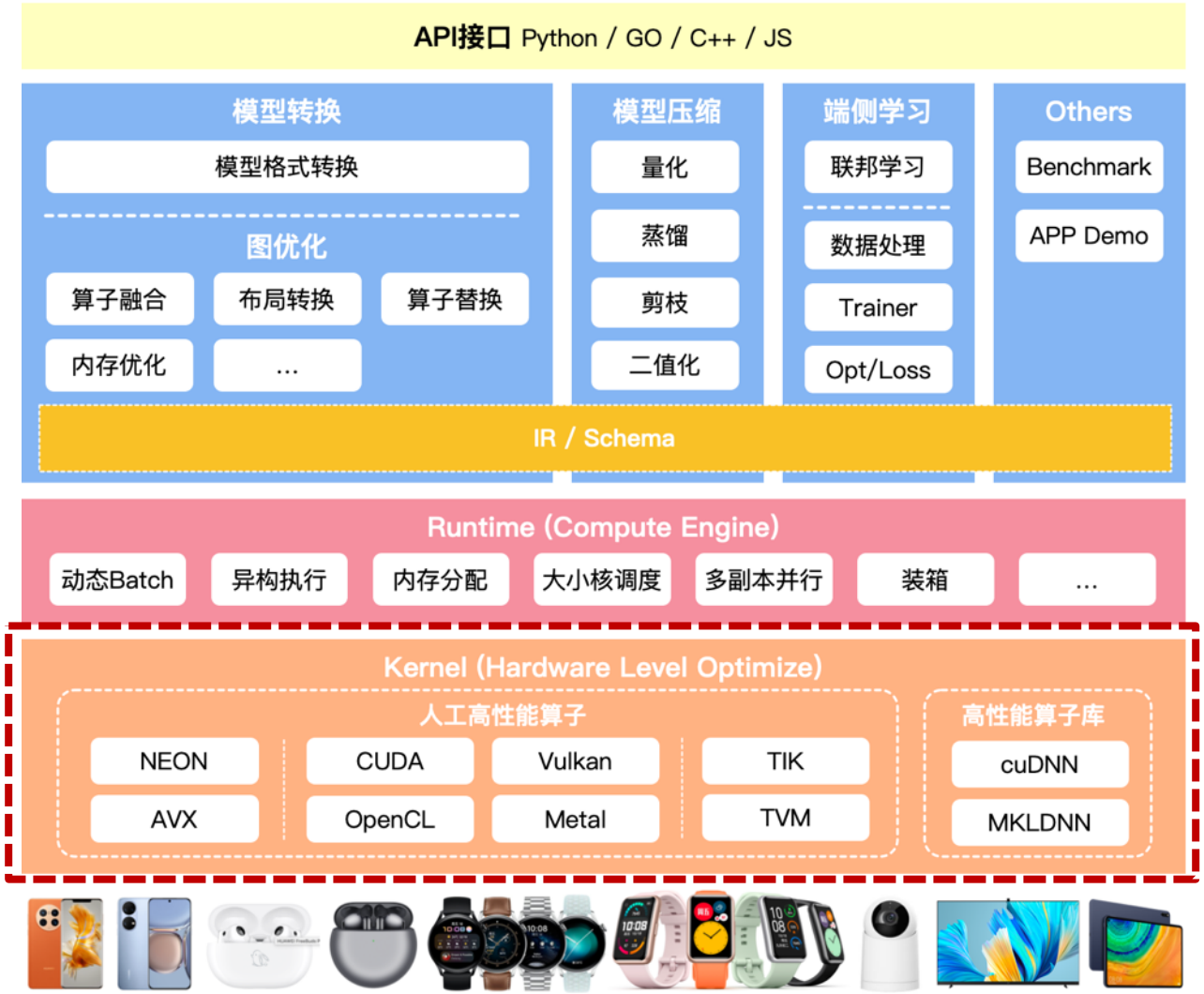


ZOMI

# Talk Overview

1. **推理系统介绍**：推理系统架构 – 推理引擎架构
2. **模型小型化**：CNN小型化结构 – Transform小型化结构
3. **离线优化压缩**：低比特量化 – 模型剪枝 – 知识蒸馏
4. **模型转换与优化**：模型转换细节 - 计算图优化
5. **Kernel 优化**
   - 算法优化 (Winograd / Strassen)
   - 内存布局 (NC1HWC0 / NCHW4)
   - 汇编优化 (指令与汇编)
   - 调度优化
6. **Runtime 优化**

# 推理引擎架构




高性能算子层

- 算子优化
- 算子执行
- 算子调度

# Talk Overview

## Conv Kernel 优化

- What is Convolution - 卷积的概念
- Im2Col Optimizer - Im2Col 优化算法
- Spatial Pack Optimizer – 空间组合优化
- Winograd Optimizer – Winograd 优化算法
- Indirect Algorithm – QNNPACK 间接卷积优化

# 间接卷积优化算法

# 基本介绍

- QNNPACK (Quantized Neural Networks PACKage) 是 Marat Dukhan (Facebook) 开发的专门用于量化神经网络计算的加速库，其卓越的性能表现一经开源就击败了几乎全部已公开的加速算法。

# 基本介绍

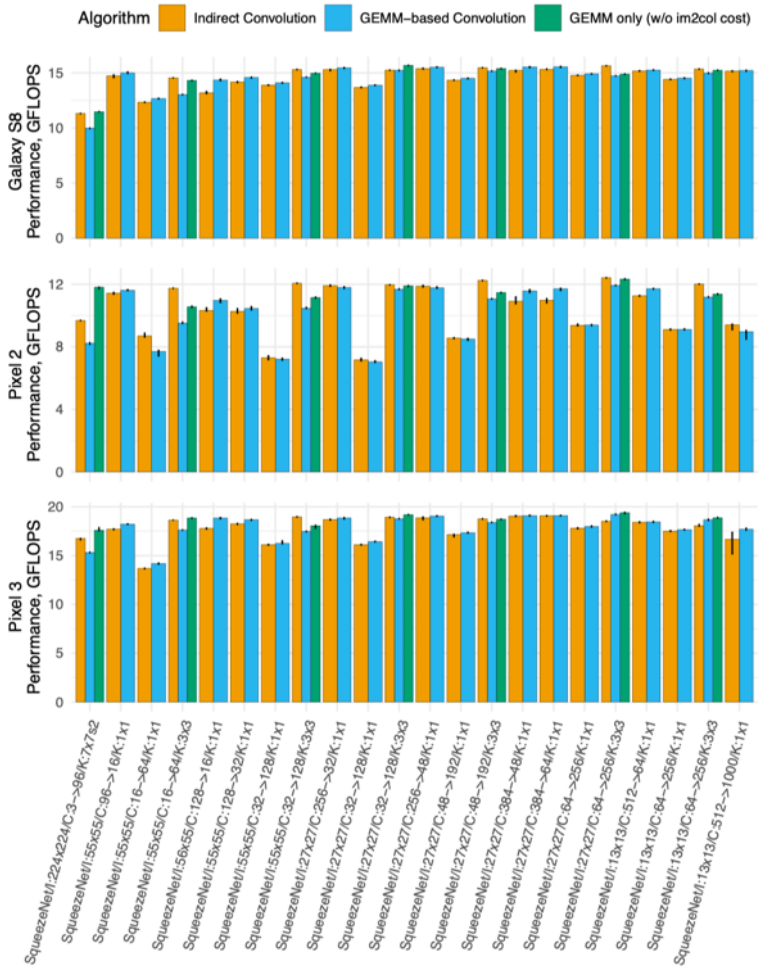



Figure 4. Performance of the Indirect Convolution algorithm and GEMM-based Algorithm on convolution operators of the SqueezeNet 1.0 model. Opaque bars represent median performance across 25 runs. Error bars represent 20% and 80% quantiles.

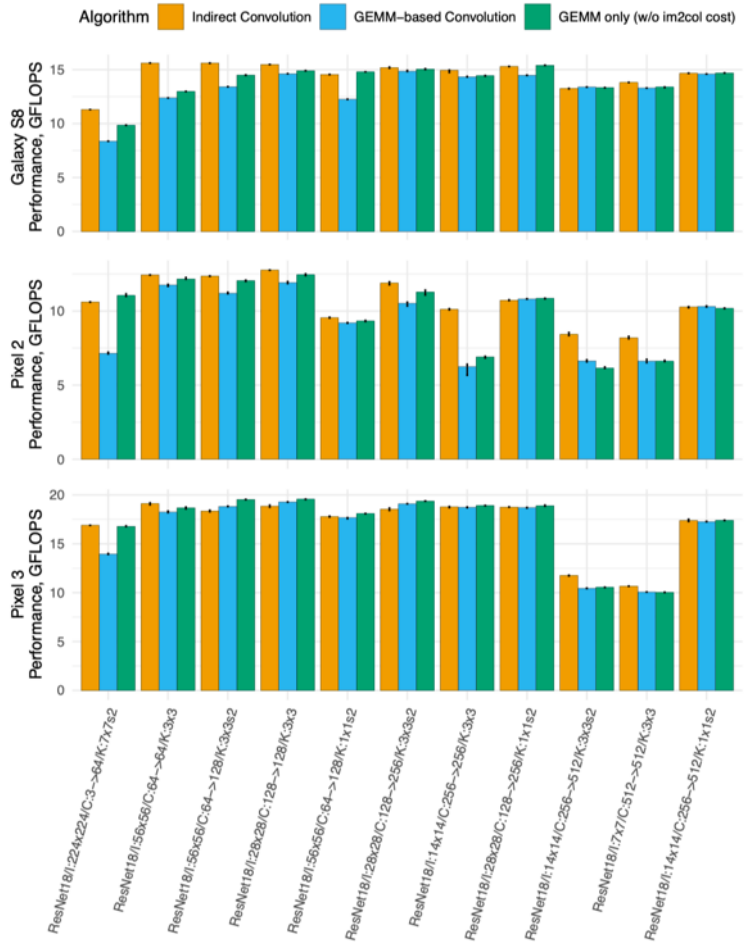



Figure 3. Performance of the Indirect Convolution algorithm and GEMM-based Algorithm on convolution operators of the ResNet-18 model. Opaque bars represent median performance across 25 runs. Error bars represent 20% and 80% quantiles.

| Device | non-1x1 | 1x1 stride-2 |
|---|---|---|
| Samsung Galaxy S8 | +10.97% | +8.02% |
| Google Pixel 2 XL | +23.26% | +0.84% |
| Google Pixel 3 | +4.31% | +0.51% |

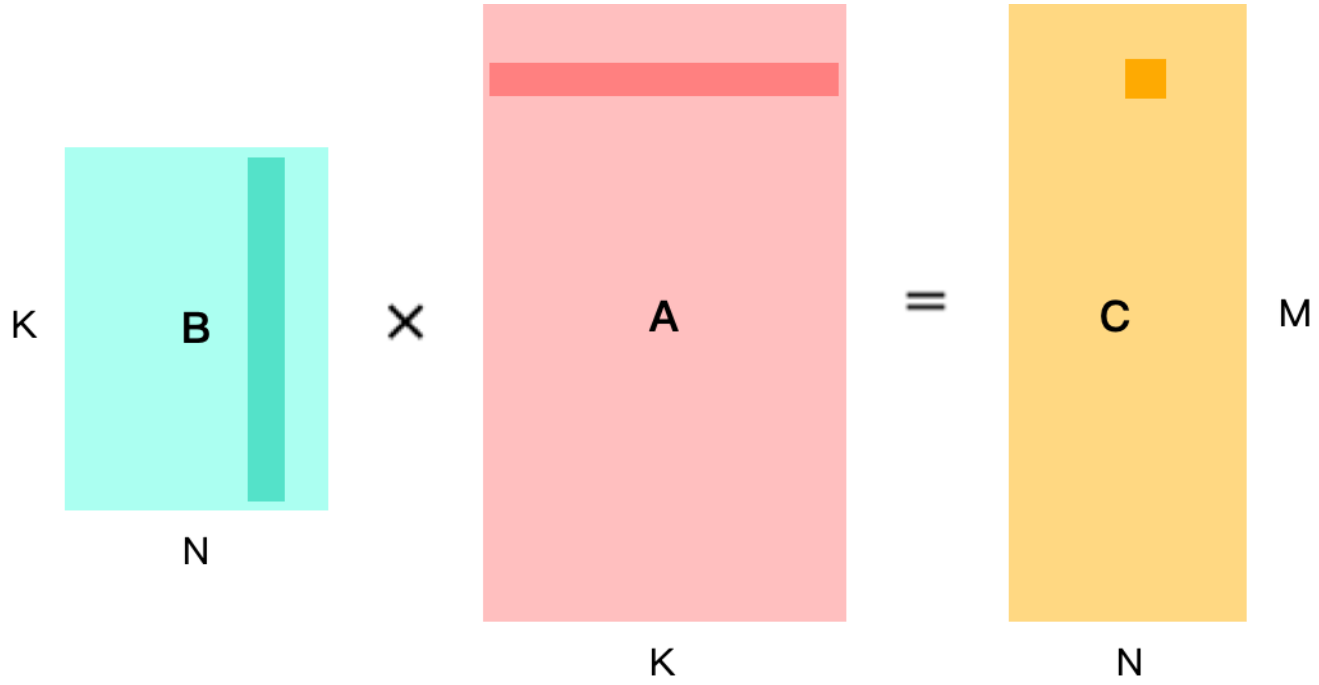
Table 3. Geomean performance of modified GEMM primitive relative to standard GEMM primtive on 1x1 and non-1x1 Convolutions in ResNet-18 model.

| Device | non-1x1 | 1x1 stride 1 |
|---|---|---|
| Samsung Galaxy S8 | +5.70% | -1.84% |
| Google Pixel 2 XL | +11.29% | -0.25% |
| Google Pixel 3 | +2.67% | -1.91% |

Table 4. Geomean performance of modified GEMM primitive relative to standard GEMM primtive on 1x1 and non-1x1 Convolutions in SqueezeNet 1.0 model.

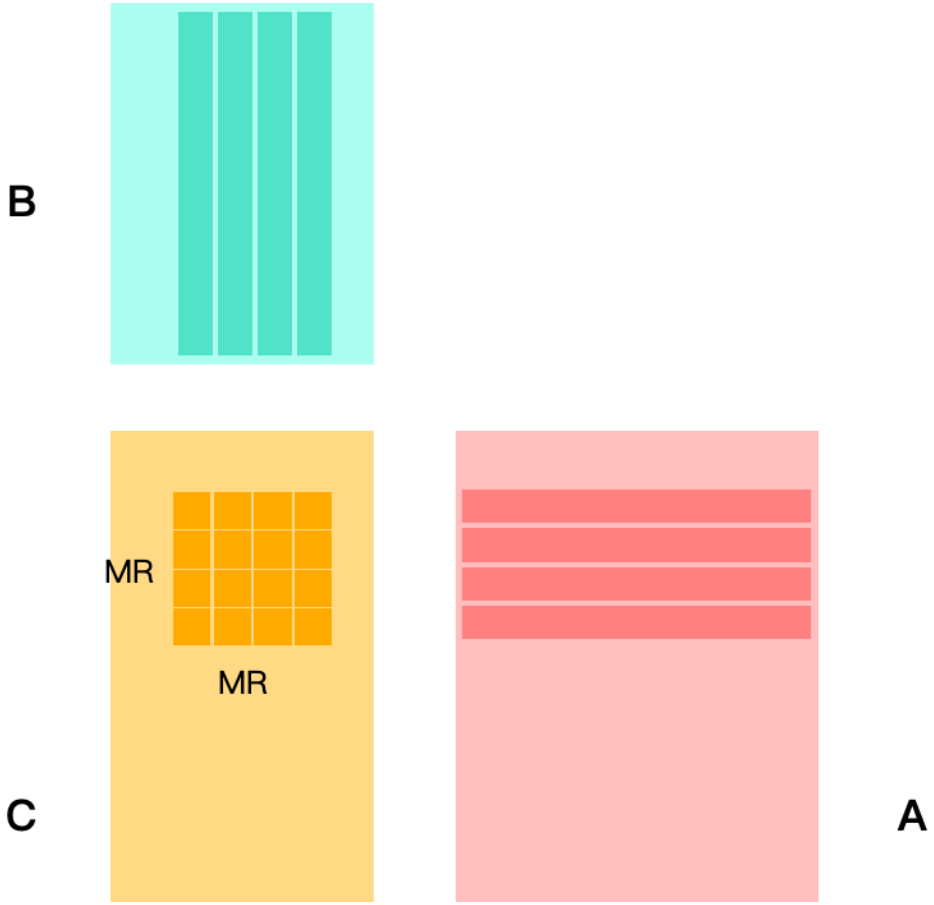
# 矩阵乘优化 Im2col 算法回顾

- 在计算 $MR \times NR$ 小块时，传统 GEMM 的方法是对 K 维度上拆分，在一次计算 kernel 处理中，仅计算 K 维的局部数据。那么在每次计算 kernel 处理过程，都会发生对输出的加载和存储。



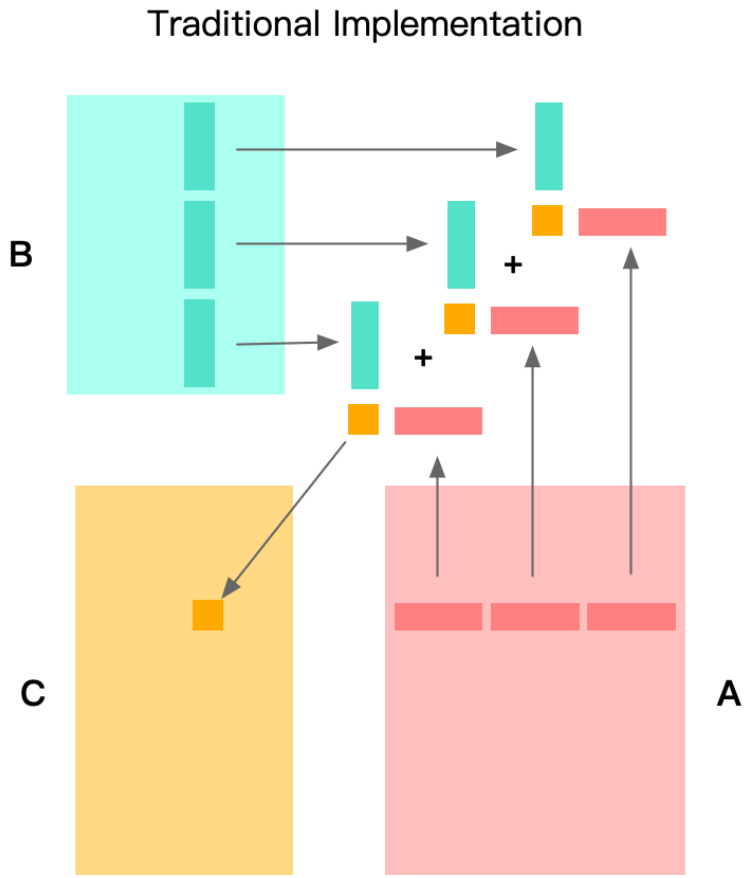
# 矩阵乘优化 Im2col 算法回顾

- 在计算 $MR \times NR$ 小块时，传统 GEMM 的方法是对 K 维度上拆分，在一次计算 kernel 处理中，仅计算 K 维的局部数据。那么在每次计算 kernel 处理过程，都会发生对输出的加载和存储。



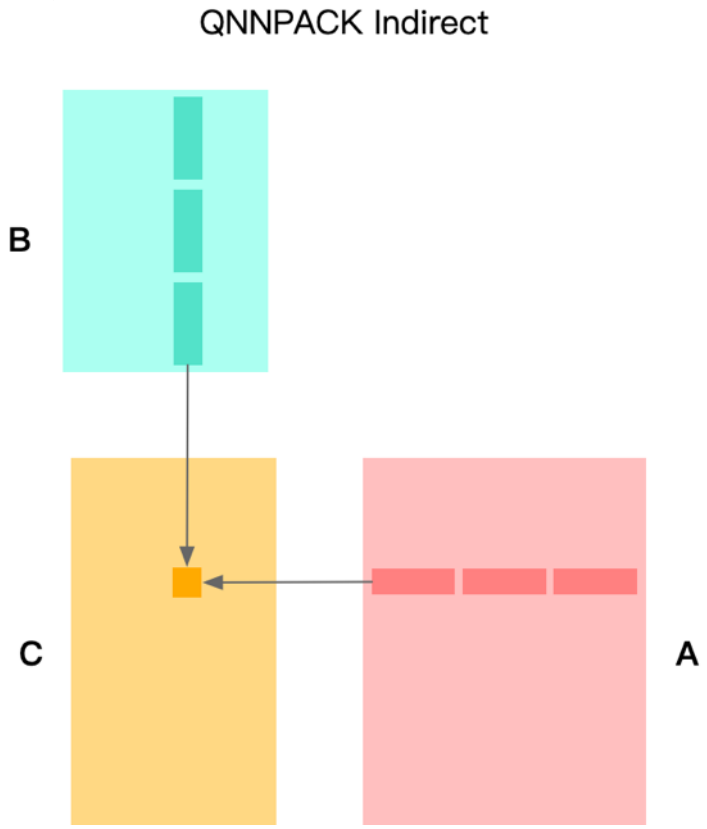
# 矩阵乘优化 Im2col 算法回顾

- 在计算 $MR \times NR$ 小块时，传统的方法是在 K 维度上拆分，在一次计算 Kernel 处理中，仅计算 K 维的局部。那么在每次计算 Kernel 的处理中，都会发生对输出的加载和存储。



# QNNPACK 算法思想

- QNNPACK 做法将整个 K 维全部在计算 Kernel 中处理完，消除了输出部分和的访存。这里所说的「将整个 K 维全部」并不是指 K 维不进行拆分，而是指拆分后不和其他维度交换，实际计算中 K 维会以 $2^n$ 为基础进行拆分。



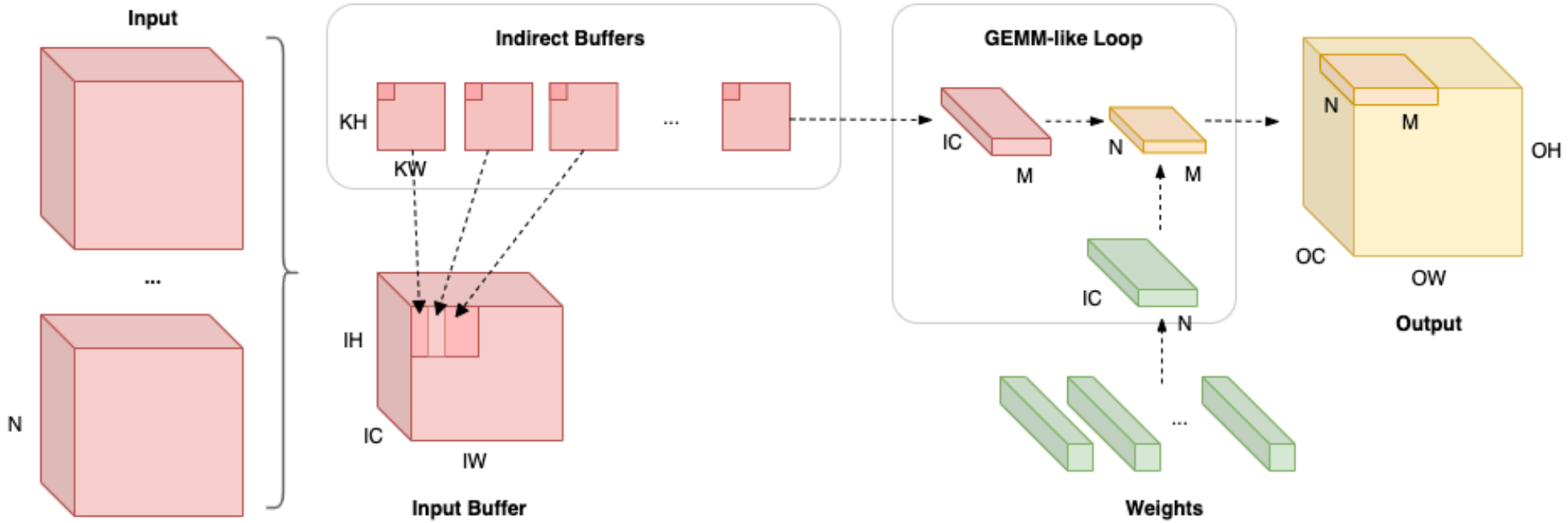
# QNNPACK 算法思想

- Im2col 优化算法存在两个问题：1）第占用大量的额外内存；2）需要对输入进行额外的数据拷贝。这两点如何才能解决呢？

- 间接卷积算法给出的答案是间接缓冲区（Indirect Buffer），对内存重新组织（Repacking）可以改进高速缓存命中率，从而提高性能。

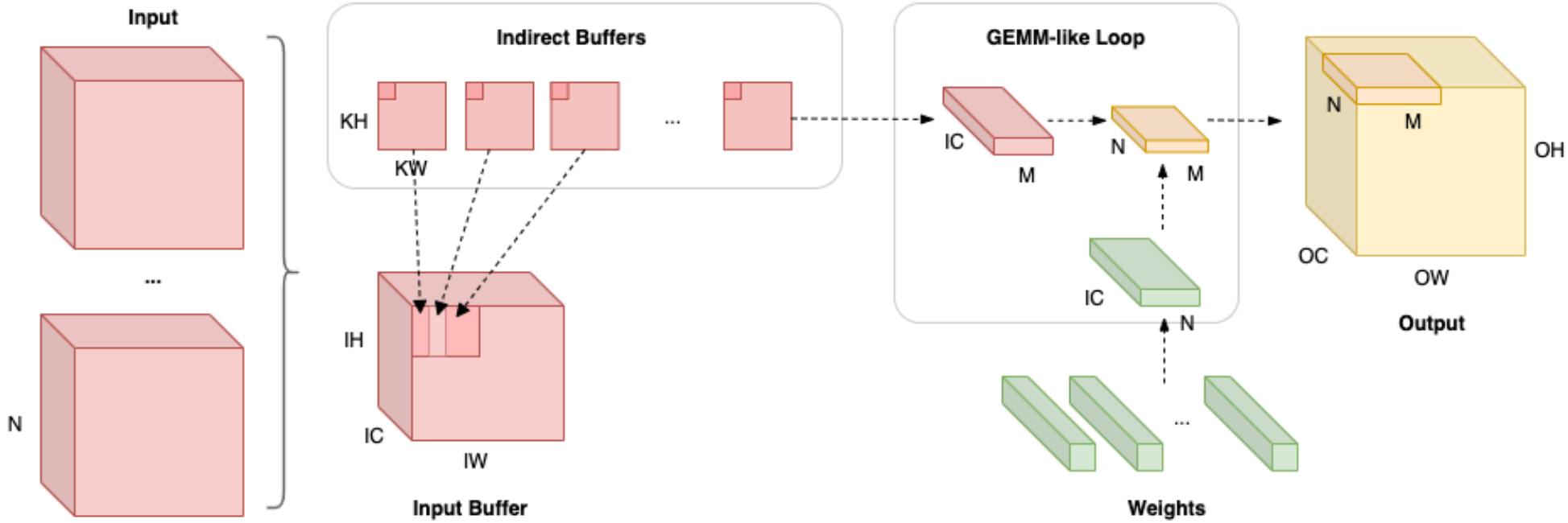
# Indirect Convolution Algorithm 工作流程

- Indirect 算法在输入缓冲区基础上构建间接缓冲区（Indirect Buffer），而间接缓冲区是间接卷积算法的核心。在网络运行时，每次计算 M×N 的输出，其中 M 为将 OH×OW 视作一维后的向量化规模（一般 M×N 为 4×4 、8×8 或 4×8 ）。



# Indirect Convolution Algorithm 工作流程

- Indirect 算法在输入缓冲区基础上构建间接缓冲区（Indirect Buffer），而间接缓冲区是间接卷积算法的核心。在计算 M×N 规模大小输出时，经由间接缓冲区取出对应输入缓冲区数据，并取出权重，计算出结果，整体计算过程等价于计算 M×K 和 K×N 矩阵乘。



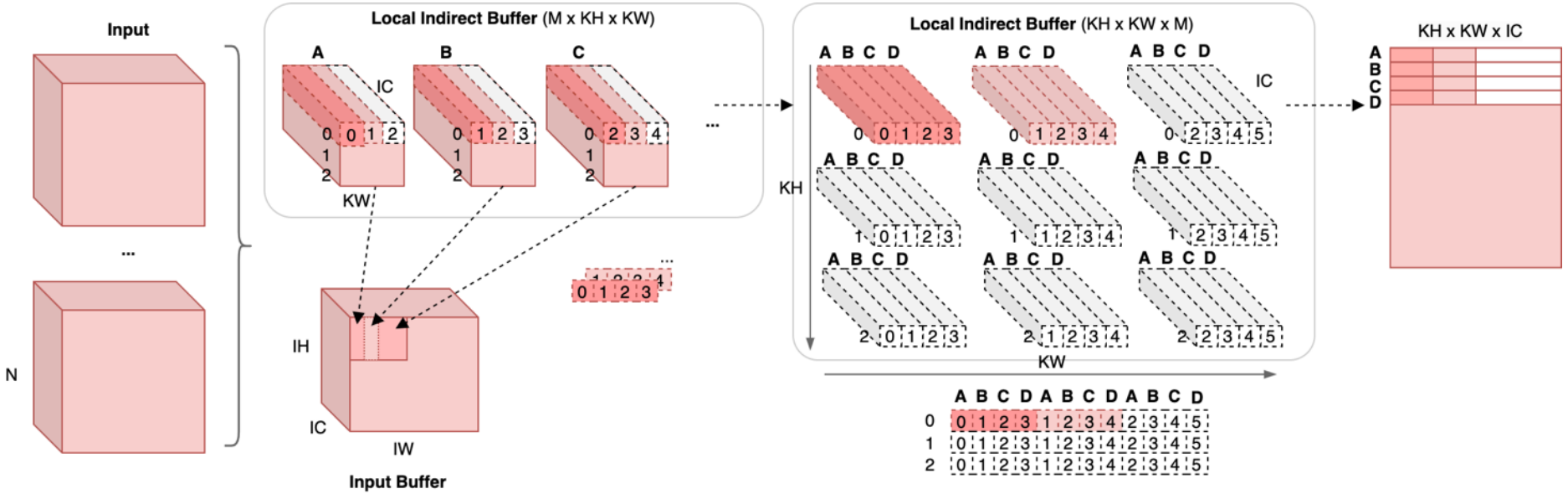
# Indirect Convolution Algorithm 工作流程

**在实现中，软件的执行过程分为两部分：**

1. 准备阶段：加载模型，配置输入缓冲区；重排权重，使其内存布局适用于后续计算；
2. 运行阶段：对于每个输入执行 $(OH * OW/M) * (OC/N)$ 次循环，每次使用 GEMM 计算 $M \times N$ 大小输出。

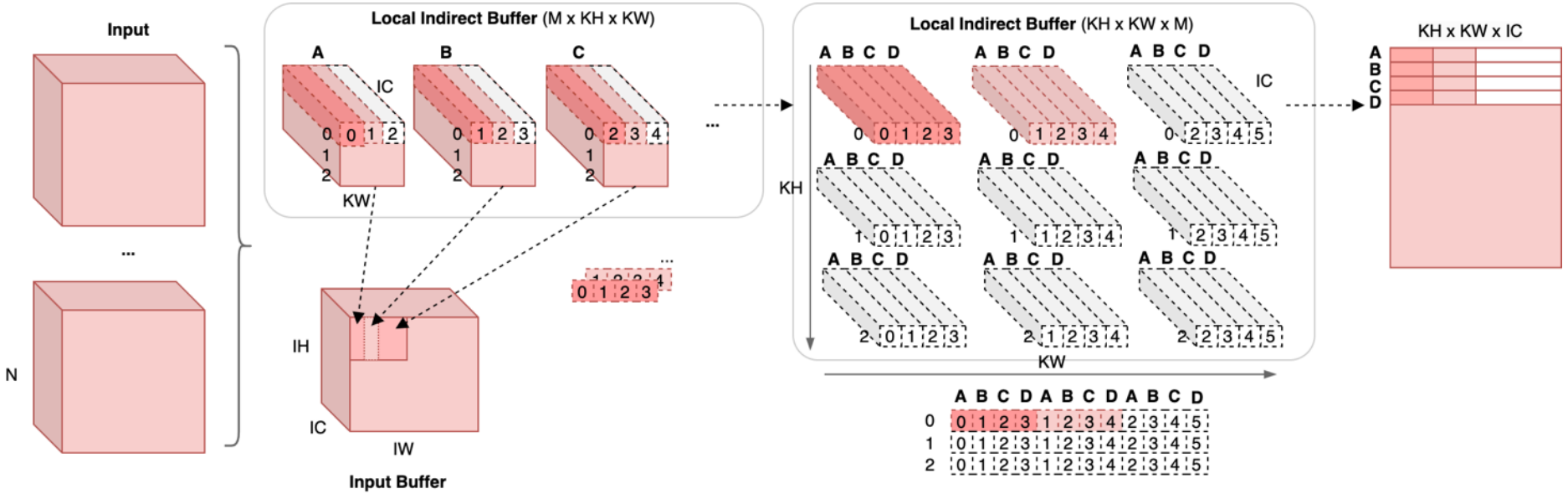
# 间接缓冲区布局

- 间接缓冲区可以理解为一组卷积核大小的缓冲区，共有 $OH \times OW$ 个，每个缓冲区大小为 $KH \times KW$（每个缓冲区对应某个输出要使用的输入地址）。每计算一个空间位置输出，使用一个间接缓冲区；空间位置相同而通道不同的输出使用相同间接缓冲区，缓冲区中的每个指针用于索引输入中 IC 个元素。

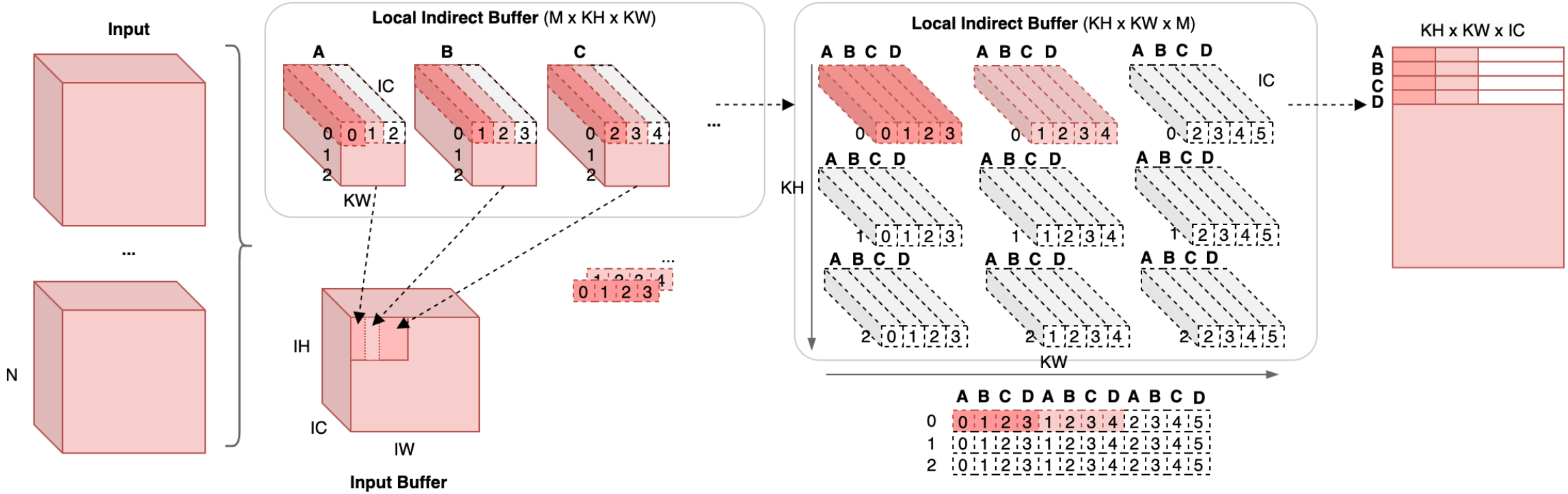


# 间接缓冲区布局

- 间接缓冲区可以理解为一组卷积核大小的缓冲区，共有 $OH \times OW$ 个，每个缓冲区大小为 $KH \times KW$（每个缓冲区对应某个输出要使用的输入地址）。在计算时，随着输出的索引内存地址移动，选用不同的间接缓冲区，即可得到相应的输入地址。无需再根据输出目标的坐标计算要使用的输入的地址，这等同于预先计算地址。



# 间接缓冲区布局



# 使用间接缓冲区计算

- 卷积之所以可以使用 Im2col 优化算法，本质原因在于其拆解后忽略内存复用后的计算过程等价于矩阵乘。而间接缓冲区使得可以通过指针模拟出对输入的访存。

**优点**

- 间接卷积优化算法解决了卷积计算的三个问题，1）是空间向量化问题，2）地址计算复杂问题，3）内存拷贝问题。

**缺点**

- 通过间接卷积算法，建立的缓冲区和数据重新组织（Repacking）对内存造成大量的消耗。

# 卷积优化

# 算法总结

# 卷积优化总结

- 本文讨论的优化方法都是通用卷积优化方法，随着神经网络处理器（如昇腾 NPU、寒武纪 MLU、Google TPU）的发展，以及其他通用计算处理器 GPGPU 的拓展（如点积相关的指令：Nvidia GPU DP4A、Intel AVX-512 VNNI、ARM SDOT/UDOT ），深度学习的优化还会继续不断深化和挖掘。

# 引用


1. https://zhenhuaw.me/blog/2019/reveal-qnnpack-implementation.html
2. https://github.com/pytorch/QNNPACK
3. https://arxiv.org/abs/1907.02129