

MINISA：面向下一代可重构推理加速器的最小指令集架构

Jianming Tong‡、Devansh Jain†、Yujie Li‡、Charith Mendis†、Tushar Krishna‡
†伊利诺伊大学，厄巴纳-香槟，美国
devansh9@illinois.edu，charithm@illinois.edu
‡佐治亚理工学院，亚特兰大，佐治亚州，美国 jianming.tong@gatech.edu，
yli3821@gatech.edu，tushar@ece.gatech.edu

arXiv:2603.20623v1 [cs.AR] 21 Mar 2026

摘要——**现代可重构人工智能加速器依赖丰富的映射和数据布局灵活性，以在矩阵乘法、卷积以及人工智能之外的新兴应用中保持较高利用率。然而，通过细粒度微控制来实现这种灵活性，需要从片外存储器获取配置位，导致控制开销高得难以承受。本文提出 MINISA，这是一种最小指令集，可按虚拟神经元（VN）的粒度对可重构加速器进行编程。该粒度既是在保留硬件灵活性的同时所能达到的最粗控制粒度，也是避免不必要控制成本的最细粒度。首先，我们介绍 FEATHER+，这是对 FEATHER 的适度改进，它消除了运行时数据流/布局协同切换所需的冗余片上复制，并支持输入和权重数据在执行前不可用、因而无法进行离线布局操作的动态场景。随后，MINISA 将 FEATHER+ 的控制抽象为三条布局设置指令，分别用于输入、权重和输出VN，以及一条用于设置数据流的映射指令。这样既减少了控制开销和指令占用，又保留了 FEATHER+所支持的合法映射与布局空间。结果表明，在涵盖人工智能（GPT-oss）、全同态加密和零知识证明的 50 个 GEMM 工作负载中，对于多种规模，MINISA 可将片外指令流量的几何平均值降低 $35\times$ 倍至（4×10^5）×。这消除了指令获取停顿；此类停顿占微指令周期的96.9%，使最高可实现$31.6\times$ 端到端加速比，适用于 16×256 FEATHER+。代码地址：https://github.com/maeri-project/FEATHER/tree/main/minisa。**

I. 引言

新的人工智能革命推动了人工智能数据中心的安装量激增。随着人工智能工作负载日趋多样化，人工智能加速器也变得越来越可重构，以高效支持各种形态的工作负载。其性能来自两个方面：（i）高吞吐量、低精度的矩阵/向量执行；（ii）可编程数据移动，支持多样化的数据流和布局。这种可编程性如今不仅对卷积神经网络和 Transformer 至关重要，对可编译为矩阵和向量运算符的工作负载也同样重要，包括同态加密（HE [22]）和零知识证明（ZKP [21]）。然而，这种灵活性也带来了一个挑战：**控制开销**。

FEATHER [23] 代表了最先进的加速器，可针对每个独立工作负载实现（数据流、布局）协同切换。然而，其编程接口对缓冲区布局、片上路由以及数据到处理单元的映射提供了**细粒度**或**微编码**控制。尽管这种范式具有很强的表达能力，却会使指令轨迹膨胀，增加片外指令获取流量，并占用有限且宝贵的片上存储器来存储指令。由此产生的控制占用会直接限制芯片上可容纳的计算瓦片大小，降低算术强度并损害端到端吞吐量。随着加速器在阵列维度和工作负载多样性方面不断扩展，这种控制开销将成为首要瓶颈（§III-D）。

本文认为，最大限度降低控制开销的恰当抽象边界是**虚拟神经元（VN）**。一个虚拟神经元对应最小的硬件点积原子。与当前虚拟神经元相比，**粒度更粗**的抽象会忽略处理单元间的映射灵活性，而**粒度更细**的抽象则会引入不必要的控制成本，却无法带来更高的灵活性。因此，在虚拟神经元这一层级进行编程，正好能够在不损失硬件内置灵活性的前提下，将控制开销降至最低。

基于这一洞见，我们提出了一种紧凑的 VN 抽象指令集架构 MINISA，仅用八条指令就能捕捉 FEATHER [23]，这一最先进的可重构加速器在映射和布局方面的灵活性。布局设置指令用于配置片上缓冲区中 VN 的嵌套循环顺序。映射指令指定数据流，并在当前布局配置下触发计算瓦片的执行。对于单个层，程序轨迹由三条布局指令以及一系列 ExecuteMapping 和 ExecuteStreaming 调用组成。对于连续的层，前一层的输出会成为下一层的输入，因此可以使用下一层的 SetIVNLayout 来指定其输出布局。这样便可以跳过当前层的 SetOVNLayout。

虽然 MINISA 压缩了 FEATHER 的控制占用，但基线 FEATHER 微架构在处理动态工作负载时仍存在两项实际问题：它可能导致片上缓冲区中的数据重复，而且其缓冲区/互连设计依赖于这样一种假设：某个操作数（通常是权重）需要预先确定，并在程序执行前重新排列为首选布局。当两个操作数都可能在运行时到达或发生变化时，例如大语言模型推理，这一假设便限制了其适用性。为消除这些障碍而不重新设计加速器，我们对 FEATHER [23] 进行扩展，将其升级为 FEATHER+，仅通过两项架构改进——（1）互连和（2）缓冲区组织——提升分发灵活性，从而（1）消除片上缓冲区中的数据重复，并（2）支持权重并非预先确定的场景

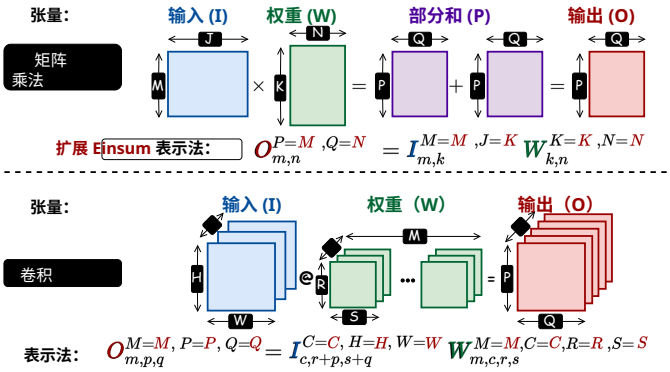


图 1：工作负载示意图。卷积通过 im2col 转换为矩阵乘法。我们使用颜色编码：蓝色/绿色/紫色/红色分别表示输入（I）/权重（W）/部分和（psum，P）/输出（O）。

编译时布局预重排。我们的贡献如下：

- 我们发现，控制开销是限制可重构人工智能加速器中分块大小和指令效率的关键因素；指令获取停顿占比从 0%（PE 阵列小于 8×8时的 FEATHER+）上升至 96.9%（16×256 FEATHER+）。
- 我们提出了 FEATHER+，这是一项适度的改进，能够：（1）支持动态输入数据；（2）消除片上缓冲区中的数据重复；（3）支持在 NEST 内加载输入，并实现权重流式传输。
- 我们为 FEATHER 提出了 MINISA，这是一种精简的 VN 级指令集架构，仅包含八条指令，同时保留硬件支持的映射和布局灵活性。
- 我们证明，MINISA 通过避免指令获取停顿，可实现最高达 31.6× 的加速比。我们展示了，对于 16×256 FEATHER+，MINISA 将指令数据比从 ∼100×（微指令编程）降至可忽略的水平（MINISA）。MINISA 保持 $< 0.1\%$ 的指令周期占比。

II. 背景

A. 工作负载规范

现代人工智能推理工作负载——从大语言模型型注意力块到文生图流水线——主要由矩阵乘法和卷积构成，其操作数值和形状 可能因层和请求而异。为以统一方式描述这种多样性，我们采用 扩展 *einsum* 记法 [5]。以矩阵乘法为例（图 1）。

$$O^{P=M,Q=N}_{m,n} = I^{M=M,J=K}_{m,k} \cdot W^{K=K,N=N}_{k,n}$$

每个张量均标注有：

- **上标**：定义逻辑形状的秩名称 有序列表。秩名称首字母大写，例如 $I \in \mathbb{R}^{M\times J}$、$W \in \mathbb{R}^{K\times N}$、$P \in \mathbb{R}^{P\times Q}$、$O \in \mathbb{R}^{P\times Q}$。
- **下标**：由秩变量和常数组成的准仿射索引表达式有序列表（例如 $[m,k]$）。秩变量采用对应秩名称的小写形式。

部分和表示中间累加状态。当归约秩的秩变量（例如 k）在映射中分布于不连续的循环层级时，就会产生这些状态；此时需要聚合部分结果才能得到最终输出。

B. 映射与布局

给定的扩展 einsum（图 1）可以通过多种合法调度来执行。我们将计算和内存访问的细粒度调度称为**映射**，它指定了秩 [11], [20] 的划分、排序和并行性。

每个张量都可以独立地采用不同方式组织在片上存储器中。我们用**布局**表示片上缓冲区 [20], [23] 中数据的细粒度组织方式。

C. 可重构加速器

对于现代算子广阔的形状空间，不存在一种在全局范围内始终最优的数据流 [17], [20]。因此，需要**可重构加速器**在运行时调整映射和布局，以在各种工作负载下保持较高的利用率 [3], [12]–[15], [18], [23], [24]。FEATHER [23] 是一项近期的实用设计，可低成本地实现数据流与布局的协同切换，使可重构执行能够投入部署。

然而，FEATHER 有两个局限。

- **预先已知的权重经过离线重排，形成理想布局**。这是因为 FEATHER 中只有一种归约内重排 NoC，专为支持输出布局重排而设计。对于权重，从缓冲区到计算阵列的刚性点对点连接要求权重必须预先已知，并经过离线重排以形成所需布局。这种方式适用于一般卷积，但对于近期的大语言模型而言效率低下，因为输入和权重都可能在运行时到达或发生变化。此外，离线重组后的权重可能在片上缓冲区中包含冗余的重复数据。
- **高控制开销**。对单个开关进行编程，在规模扩大时会产生占主导地位的配置开销。

为同时消除这两个问题，我们在 §III 中介绍**FEATHER+** 及其编程视角，并在 §IV 中介绍 FEATHER 的虚拟神经元抽象和基于虚拟神经元的指令集架构 MINISA+ 。

III. FEATHER+ 与编程视角

本节介绍 FEATHER+ 及其编程视角。

A. *FEATHER*+概述

在其计算单元中，FEATHER+（图 2）包含：

- **NEST**：按列独立的 $AH\times AW$ 处理单元阵列。每个处理单元包含 $2\times AH$ 个本地寄存器（采用双缓冲以隐藏加载延迟），并执行 AH元素点积。
- **BIRRD**：一种多级归约重排序网络，可在数据归约期间更改目标存储体。

在其存储器中，FEATHER+ 包含：

- **流式缓冲区** 存储**“流式张量”**，该张量在每个 NEST 列内自上而下流水传输，并由该列中的所有处理单元复用。
- **驻留缓冲区** 将**“驻留张量”** 存储在处理单元的本地寄存器中（绿色）。当流式张量经过处理单元时，这些寄存器中的值会与其参与一次点积运算。
- **输出缓冲区** (OB) 用于存储需要后续更新的部分和，即时间归约。OB 是唯一采用多 bank 的缓冲区，每个 bank 都配有独立的地址生成机制，以支持灵活的输出布局重排。

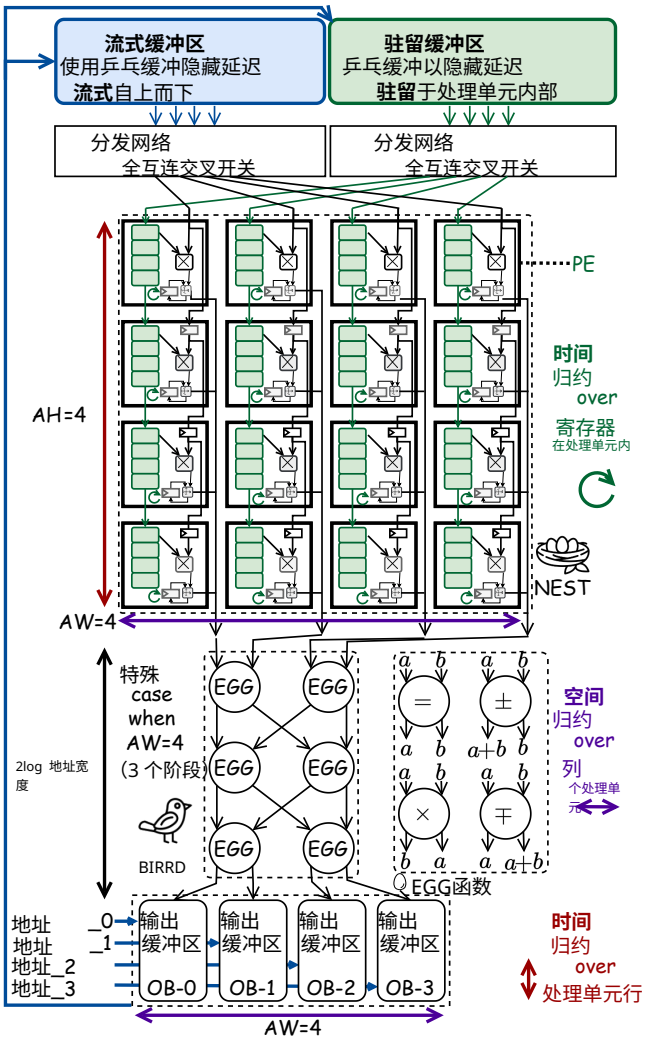


图 2：FEATHER+ 的程序员视图。它是一种能够协同切换映射与布局的可重构加速器。

B. 将 *FEATHER* 架构改进为 *FEATHER+*

FEATHER+ （图 2）有三项架构改进：

- **从缓冲区到 NEST 的全互连分发**。我们用两个独立的全互连交叉开关，取代流式缓冲区和驻留缓冲区到 NEST 的逐列点对点连接。这样，任何驻留数据都可以灵活地多播到任意处理单元列，无需在片上缓冲区中显式存储同一数据的重复副本。这种交叉开关式分发在当代人工智能专用集成电路中很常见 [8]–[10], ，且资源开销合理。

- **简化流式缓冲区分银行**。采用全互连分发后，FEATHER+不再需要多银行流式缓冲区接口来访问各银行中的不同数据行。我们改回单银行设计，并简化地址生成。

- **从输出缓冲区到驻留缓冲区的连接**。这使得当前层的输出（作为下一层的输入）既可以驻留在处理单元的本地寄存器中，也可以流经处理单元，与下一层的权重进行点积。

总的来说，这些改动：(1) 不再要求必须在执行前确定“驻留张量”，才能将其以首选布局放入驻留缓冲区，(2) 释放片上缓冲区中重复的数据；(3) 支持将输入和权重都用作“驻留张量”。FEATHER+是本研究的默认硬件后端。

C. FEATHER 的编程视角+

FEATHER+ 是一种受约束的数据流加速器 [19]。程序员需要在两个约束下，通过两种数据流选择，将工作负载映射到“三级归约”中。

1) 灵活映射：

a) 三级时间与空间归约：

- **处理单元内的时间归约**通过一个 AH元素点积完成：本地寄存器中的数据与随时间流入的数据进行点积运算，从而生成单个部分和。

- **通过 BIRRD**对**一行处理单元进行空间归约**。
- **在 OB**中对**一列处理单元进行时间归约**。

这种灵活的归约方式允许将任意选定的处理单元组合在一起，以计算一个点积。

b) 两种灵活的混合数据流：

- **输入输出静态（IO-S）**：将输入存储在每个处理单元中，流式传输权重，并在处理单元内局部累加部分和。
- **权重输出驻留（WO-S）**：将权重存储在每个处理单元中，流式传入输入，并在处理单元内本地累加部分和。

对于矩阵乘法，若满足 $M > N$，则选择 IO-S；否则选择 WO-S。

2）映射约束：

a）约束 *1*：处理单元内点积：处理单元本地寄存器中的所有 AH 数据都必须参与同一个点积，以确保每个处理单元的驻留张量随时间不断归约。

b）约束 *2*：列内数据复用：流式传入的张量会在同一列内的所有处理单元之间复用，以便与各自的驻留张量计算点积。不过，NEST 中的所有列彼此完全独立运行。

D. FEATHER 的控制开销+

尽管 FEATHER+ 支持板级映射/布局选择，但通过开关级和周期级微控制来实现这种灵活性会产生大量开销。程序必须为每个周期指定 BIRRD 控制以及缓冲区地址生成。这会直接带来两个后果。首先，控制状态会占用片上存储，而这些存储原本可用于存放激活值和权重，从而限制可行的最大分块大小并降低算术强度。其次，获取长指令轨迹会增加片外指令流量，对内存受限层尤其不利。随着加速器阵列规模扩大，这种控制压力会成为端到端吞吐量的首要限制因素。如表 I 所示，采用微控制为 FEATHER+配置每个独立开关时，在更大规模下，获取指令所带来的存储开销将占据主导地位。这种控制开销促使我们提出 MINISA——一种以低成本实现硬件灵活性的抽象。

表 I：矩阵乘法指令提取的显式停顿 $\sum_{m}\sum_{k}\sum_{J=40}\sum_{K=40}\sum_{N=88} I^{M=65536}\cdot W^{n}$、、、

FEATHER	4×4	8×8	4×64	16×16	8×128	16×256
停顿	0	0	75.3%	65.2%	90.4%	96.9%

IV. MINISA

本节介绍 MINISA，这是一种最小指令集，可在虚拟神经元（VN）粒度上配置 FEATHER+ 。我们将详细介绍 VN 抽象、 VN 下的映射约束、相应的指令集架构以及执行模型。*A.* 核心思想：以 *VN*（*AH* 元素点积）作为抽象

MINISA 源于一个简单的架构观察：

FEATHER+ 始终在其 *PE* 中以 AH 元素点积的粒度进行计算，并支持对来自任意 *PE* 的点积结果进行灵活归约。

因此，我们将**虚拟神经元**[1] 定义为与这一硬件原子相匹配的最小软件操作数片段，即 AH 元素点积。这带来了一个关键洞见：

以 *VN* 粒度进行编程，是在保留 *PE* 间映射灵活性的同时所能采用的最粗粒度控制，也是能够避免不必要开关级开销的最细粒度控制。

具体而言，MINISA 将 FEATHER+ 的合法映射和布局空间重新表述为 VN 粒度。这种因式分解保留了 FEATHER+ 内置的可重配置性，同时大幅压缩了指令占用，从而支持更大的片上区块并提升端到端性能。

B. MINISA 抽象——虚拟神经元（*VN*）

1）定义：A虚拟神经元是最小的硬件点积原子，同时也是软件（编译）中在不损失硬件灵活性的前提下最大的原子单元。对于具有 $AH \times AW$ 个 NEST 的 FEATHER+，每个 PE 执行一个 AH 元素点积，因此 VN 大小应小于或等于 AH。为便于说明，本文以下将 VN 大小用作 AH 。

2）操作数专属虚拟神经元：不同操作数各自拥有虚拟神经元，包括输入（I_{VN}）、权重（W_{VN}）、部分和（P_{VN}）和输出（O_{VN}）。每个虚拟神经元通过沿归约维度将连续的阵列高度个元素分组而得，即输入的 J 、权重的 K 以及部分和与输出的 Q （它们会成为后续层的输入 J ）。例如，权重VN的索引为 $W_{VN}(r,c)$，其中$r \in [0, \lceil K//AH \rceil - 1]$ 和 $c \in [0, N-1]$ 分别表示行列索引。这样， I_{VN} 和 W_{VN} 分别准确表示输入和权重中将由单个处理单元中的局部点积**消耗**的一组元素。每个处理单元的点积会在 P_{VN} 或 O_{VN} 中产生一个元素，具体取决于是否需要进一步归约。

3) FEATHER+ VN 下的映射约束：VN 抽象隐藏了约束 1，因此约束 2（列内 I_{VN} 复用）是唯一限制。 I_{VN} 在每个 NEST 列内自上而下进行流水线处理，要求该列中的所有处理单元都计算其 W_{VN} 与此共享的 I_{VN} 数据流之间的点积，以实现正确的局部归约。

C. MINISA 指令集架构概述

1) 指令集架构： MINISA 定义了四种指令集架构（表II），用于以 VN 粒度重新配置片上缓冲区中的映射和数据布局，

[1]我们借用了 MAERI 中的这一术语 [13] 。MAERI 是一种灵活的数据流加速器，它将 VN 定义为大小可变的点积引擎。但在这里，我们用它来抽象表示由处理单元执行的定长点积。

表 II：MINISA指令概述。

指令集架构名称	指令集架构含义
SetIVNLayout	配置片上缓冲区中 I_{VN} in 的数据布局。
SetWVNLayout	配置 W_{VN} 在片上缓冲区中的数据布局。
SetOVNLayout	为输出VN配置输出 缓冲区布局，并清除（初始化）片上输出缓冲区以进行累加。
ExecuteMapping	指定 VN 级映射参数，并触发执行当前 VN 布局下、 PE 阵列上的单个计算瓦片。
ExecuteStreaming	指定每列中 VN 的流式传输方式并在 IO-S 与 WO-S 之间交换数据流。
Load	将数据从片外存储器 Load 到流式／驻留缓冲区。
Write	将数据从流式／驻留缓冲区 Write 到片外存储器。
激活	对流式／驻留缓冲区中的数据执行激活函数。

并支持用于数据加载/写入和激活的指令集架构。

- `SetIVNLayout`/ `SetWVNLayout`：配置输入虚拟神经元（I_{VN}）和权重（W_{VN}）的片上流式缓冲区和驻留缓冲区布局。

- `SetOVNLayout`：配置输出缓冲区中输出虚拟神经元（O_{VN}）的布局，该布局将成为下一次连续操作的输入布局。

- `ExecuteMapping`：将驻留张量（W_{VN}或 I_{VN}）映射到一个计算瓦片中的处理单元。

2) ISA 位宽分析：ISA 参数位宽旨在支持片上缓冲区容量与架构维度之间的最大比率。具体而言，这适用于驻留或流式缓冲区深度（$D = D_{sta} = D_{str}$）与 NEST 宽度或高度（AW、 AH）之间的比率。工作负载参数可在这些位宽限制内配置为任意值；如果实际元素索引超过

工作负载形状，它会自动进行零填充。

ExecuteMapping	*OpCode*	G_r	G_c	r_0	c_0	s_r	s_c
位宽	3	$\lceil \log_2(AW) \rceil$		$\lceil \log_2(D/AH \cdot AW) \rceil$		$\lceil \log_2(D/AH) \rceil$	
取值范围	111	$[1, AW]$		$0 \le r_0 c_0 \le \lfloor D/AH \cdot AW \rfloor$		$[0, D/AH]$	
例如，在以下范围内的非零值 W_{VN}：				$r_0 < K, c_0 < N, s_r \le K/2, s_c \le N/2$			

ExecuteStreaming	*OpCode*	*Dataflow(df)*	m_0	s_m	VN_{SIZE}	T
位宽	3	1	$\lceil \log_2(D/AH) \rceil$-1		$\log_2(AH)$	$\lceil \log_2(D/AH) \rceil$
取值范围	011	0/ 1(用于输入驻留/权重驻留)	$1 \le x \le \lceil D/AH \rceil$-1		$1 \le x \le AH$	$1 \le x \le \lceil D/AH \rceil$

图 3： $AH \times AW$ NEST 的 Execute* 字段定义。

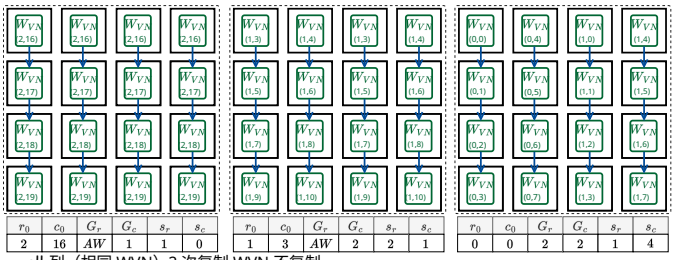
D. 执行映射：*NEST* 的灵活映射

1) 定义：对于一个 $AH \times AW$ PE 阵列，PE (a_h, a_w)表示位于第 $a_h \in [0, AH)$ 行、第 $a_w \in [0, AW)$ 列的处理单元。每个处理单元可以容纳单个 W_{VN}。因此，我们的目标是指定每个处理单元中存储的是哪个 $W_{VN}(r, c)$。以 WO-S 数据流为例，MINISA 使用六个参数实现所有合法映射 $\theta_{EM} = (r_0, c_0, G_r, G_c, s_r, s_c)$。

$$r = r_0 + \left\lfloor \frac{a_w}{G_r} \right\rfloor \qquad c = c_0 + s_r\, a_h + s_c \left(a_w \bmod G_c\right) \tag{1}$$

其中，(r_0, G_r) 控制每列中 W_{VN}s 的放置位置，同时满足架构约束：同一处理单元列中的所有处理单元共享相同的 W_{VN} “行索引（r）”。

- r_0 表示 NEST 中 W_{VN} 的起始行索引。
- G_r 指定在递增至下一个行索引之前，共享相同 W_{VN} 行索引（r）的连续处理单元列数量，因此其上限为列数（AW）。(c_0, s_r, G_c, s_c) 控制跨处理单元列的 W_{VN}。
- c_0 选择起始 W_{VN} 列索引（c）。
- G_c 定义 c 在各列间 W_{VN} 的复制周期。
- s_c 确定一个周期内不同处理单元列模式之间 W_{VN}的列索引（c）间距。
- s_r 是跨越 PE 行的时间步长，用于确定W_{VN} 在处理单元列中的列索引如何递增。



r_0	c_0	G_r	G_c	s_r	s_c
2	16	AW	1	1	0

r_0	c_0	G_r	G_c	s_r	s_c
1	3	AW	2	2	1

r_0	c_0	G_r	G_c	s_r	s_c
0	0	2	2	1	4

all 列（相同 WVN）2 次复制 WVN 不复制

图 4： 4×4NEST 的执行映射示例。

表 III：跨 VN 布局的统一 3 位排列编码。每列都将相同的顺序_id 值应用于 VN 约束（$K_{L0}=AH$、 $J_{L0}=AH$、 $Q_{L0}=AH$）后剩余的操作数特定三秩集合。

顺序	W_{VN} 顺序	I_{VN} 顺序	O_{VN} 顺序
000	$k_{L1}\to n_{L0}\to n_{L1}$	$j_{L1}\to m_{L0}\to m_{L1}$	$p_{L1}\to p_{L0}\to q_{L1}$
001	$k_{L1}\to n_{L1}\to n_{L0}$	$j_{L1}\to m_{L1}\to m_{L0}$	$p_{L1}\to q_{L1}\to p_{L0}$
010	$n_{L0}\to k_{L1}\to n_{L1}$	$m_{L0}\to j_{L1}\to m_{L1}$	$p_{L0}\to p_{L1}\to q_{L1}$
011	$n_{L0}\to n_{L1}\to k_{L1}$	$m_{L0}\to m_{L1}\to j_{L1}$	$p_{L0}\to q_{L1}\to p_{L1}$
100	$n_{L1}\to k_{L1}\to n_{L0}$	$m_{L1}\to j_{L1}\to m_{L0}$	$q_{L1}\to p_{L1}\to p_{L0}$
101	$n_{L1}\to n_{L0}\to k_{L1}$	$m_{L1}\to m_{L0}\to j_{L1}$	$q_{L1}\to p_{L0}\to p_{L1}$
110/111	保留	保留	保留

集合WVN布局位宽取值范围

$OpCode$	$Order$	N_{L0}	N_{L1}	K_{L1}
3	3	$\lceil\log_2(AW)\rceil$	$\lceil\log_2(D/AH)\rceil$	$\lceil\log_2(D/AH)\rceil$
000	$[0,5]$	$N_{L1}N_{L0}K_{L1}\le[D/AH]$		

集合IVN布局位宽取值范围

$OpCode$	$Order$	M_{L0}	M_{L1}	J_{L1}
3	3	$\lceil\log_2(AW)\rceil$	$\lceil\log_2(D/AH)\rceil$	$\lceil\log_2(D/AH)\rceil$
001	$[0,5]$	$M_{L1}M_{L0}J_{L1}\le[D/AH]$		

集合OVN布局位宽取值范围

$OpCode$	$Order$	P_{L0}	P_{L1}	Q_{L1}
3	3	$\lceil\log_2(AW)\rceil$	$\lceil\log_2(D/AH)\rceil$	$\lceil\log_2(D/AH)\rceil$
010	$[0,5]$	$P_{L1}P_{L0}Q_{L1}\le[D/AH]$		

加载/存储位宽取值范围

$OpCode$	$HBM\ Address$	$Target$
3	$\log_2(HBM\ Cap)$	1
101/ 100		0 /:1驻留/流式缓冲区

图 5：MINISA 对布局、加载/存储和数据流的规范。 D 表示驻留/流式缓冲区的深度。

每个缓冲区最多可容纳 $D/AH{\cdot}AW$ 个 VN，使得$r_0{\cdot}c_0$达到峰值 $D/AH{\cdot}AW$。步长在一半处达到峰值。任何 $W_{VN}(r,\ c)$ 超出张量边界的部分均隐式填充为零。

2) 案例研究： 由于各列彼此独立，图 4 展示了三种执行映射选择：(1) 在所有列中复制相同的 WVN，使所有 IVN块能够在每一列中并行处理。(2) 将列划分为两个复制组，使 IVN流能够拆分为两个独立子流并行处理。(3) 为每列分配不同的 WVN集合，使每列处理独立的 IVN流。

E. ExecuteStreaming

1) 定义： 在 WO-S 下，流式 I_{VN}s 从每个处理单元列的顶部进入，并在向下传播时于连续周期中复用。因此，我们需要指定随时间向每个处理单元列注入哪个 $I_{VN}(m,\ j)$。

为尽量减少指令集架构开销，ExecuteStreaming 复用配对的 ExecuteMapping 所定义的参数 θ_{EM} ，并额外引入五个参数 $\theta_{ES}=(m_0,\ s_m,\ T,\ VN_{size},\ df)$。

- m_0 是起始流式行索引。
- s_m 是流式 I_{VN}的时间步长。
- T 是注入每个处理单元列的 I_{VN} 数量。
- df 改变数据流选择（IO-S 或 WO-S）。
- VN_{size} 定义虚拟神经元的大小。所有字段均编码“值-1”，省略零值以减少位宽 h.

给定配对的执行映射，在时间步骤 $t\in[0,T$ 注入处理单元列 a_w 的 I_{VN}（$m,\ j$）为

$$j=r_0+\left\lfloor\frac{a_w}{G_r}\right\rfloor,\quad m=m_0+s_m\,t+\left\lfloor\frac{a_w \bmod G_r}{G_c}\right\rfloor.$$

2) 案例研究： 例如，考虑一个具有 $AH\times4$ 个处理单元的 PE 阵列，其配对的执行映射参数选择为（r_0、 G_r、 G_c） =（0、2、 1），并将 ExecuteStreaming 设置为（m_0、 s_m、 T、 df） =（0、 3、 3、 1）。处理单元列 0 和 1 属于归约组 $j=0$，而处理单元列 2 和 3 属于归约组 $j=1$。在每个包含两列的组内，两列进行拆分，并总体并行处理 I_{VN}。因此，经过三个注入周期后，进入四个处理单元列顶部的 I_{VN}为:

2	$I_{VN}(6,0)\downarrow$	$I_{VN}(7,0)\downarrow$	$I_{VN}(6,1)\downarrow$	$I_{VN}(7,1)\downarrow$
1	$I_{VN}(3,0)\downarrow$	$I_{VN}(4,0)\downarrow$	$I_{VN}(3,1)\downarrow$	$I_{VN}(4,1)\downarrow$
0	$I_{VN}(0,0)\downarrow$	$I_{VN}(1,0)\downarrow$	$I_{VN}(0,1)\downarrow$	$I_{VN}(1,1)\downarrow$
cycle	PE col 0	PE col 1	PE col 2	PE col 3

在该阵列中，每一列对应硬件中的一个处理单元列，而每一行表示一个周期内注入该处理单元列顶部的 I_{VN}。注入后，每个 I_{VN} 会在连续周期内沿对应的处理单元列向下传播，因此同一个流式张量（在 WO-S 下为 I_{VN}）会被该列中的所有处理单元在时间上复用。

*F. Set*VNLayout*：片上缓冲区的灵活布局

1) 核心思想：地址宽度缓冲区中的四级划分张量：三个布局指令（`SetIVNLayout`、`SetWVNLayout`和`SetOVNLayout`）定义了逻辑二维秩张量如何放置到物理 $D\times AW$ 片上缓冲区中。每条指令针对不同的操作数类型（I_{VN}/W_{VN}/P_{VN}和 O_{VN}）。在 WO-S 下， I_{VN}映射到流式缓冲区，而 W_{VN} 映射到驻留缓冲区。在 IO-S 下，这些角色互换。

从总体上看，布局规范分为三个步骤：(1) 选择秩划分因子，(2) 选择划分后秩的排序，以及 (3) 将生成的 VN 序列折叠到 $D\times AW$ 缓冲区中。

2) 分区因子：以权重矩阵（$K,\ N$）为例，我们首先以元素粒度表示布局。每个秩分为两个层级：

$$K=K_{L1}K_{L0},\qquad N=N_{L1}N_{L0},$$
$$k=k_{L1}K_{L0}+k_{L0},\qquad n=n_{L1}N_{L0}+n_{L0}.$$

由此得到四个循环秩 $\{k_{L0},k_{L1},n_{L0},n_{L1}\}$，它们既可表示连续布局，也可表示跨步布局。

VN 始终沿归约秩将连续元素分组（即 $K/J/Q$图 6 中的 W_{VN}/I_{VN}/O_{VN}），因此 MINISA 将最内层归约级因子固定为 VN 的大小

，例如将 K_{L0} 设为 W_{VN}的 VN 大小。VN 大小可以是任意不超过 AH的值。本文以 AH 为例进行说明。

此外，单个 VN 的所有元素会在多个周期内串行访问（而非在单个周期内并行访问），并且该 VN 的所有元素都放置在缓冲区中同一列索引下连续的缓冲区行中。

3) 排序： VN 抽象掉了归约秩的第 0 级划分因子，剩余的自由度仅为其余三个秩的排列顺序，例如，$\{K_{L1},\ N_{L0},\ N_{L1}\}$ 对于 W_{VN}。对于 VN 粒度布局，这个缩减后的循环空间仍是完备的。在这种秩分区下，我们通过 $W_{VN}(r,\ c)$ 来标识权重 VN，其中

$$r = k_{L1}, \qquad c = n_{L1} \cdot N_{L0} + n_{L0}.$$

a) 地址生成： 我们首先根据选定的秩顺序展平所有 VN，得到一个一维索引 L，然后通过在 $D \times AW$缓冲区上进行行优先布局，将 L映射为缓冲区坐标。对于具有秩 $\{K_{L1}, N_{L0}, N_{L1}\}$ 的 W_{VN}，

$$\mathbf{R} = [K_{L1},\ N_{L0},\ N_{L1}], \qquad \mathbf{RV} = [k_{L1},\ n_{L0},\ n_{L1}],$$

其中，$\mathbf{R}$表示有序秩，$\mathbf{RV}$ 表示其对应的秩变量。令 $P = (p_0,\ p_1,\ p_2)$ 为 0、1、2 的一个置换，用于定义从最外层到最内层的循环顺序。由此得到的展平 VN 索引为

$$L = RV_{p_0} \cdot R_{p_1} R_{p_2} + RV_{p_1} \cdot R_{p_2} + RV_{p_2}$$

且物理缓冲区地址为

$$addr_{row} = \left\lfloor \frac{L}{AW} \right\rfloor, \qquad addr_{col} = L \bmod AW.$$

4）位宽分析：

a) 排序： 三个分区秩的任意排序提供 $3! = 6$种合法排列选择，需要 $\lceil \log_2 6 \rceil = 3$ 位。因此，排序 用 3 位编码。表 III 列出了每个编码对应的操作数特定映射。

b) 划分因子： 分区因子决定张量的大小，这些张量必须适配整体缓冲区容量。对于 W_{VN}， $D \times AW$缓冲区可存储的 VN 数量上限为 $K_{L1} \cdot N_{L1} \cdot N_{L0} \le \lfloor \frac{D}{AH} \rfloor \cdot AW$。这限制了整体位宽。此外，我们还限制第 0 级非归约因子（例如 $N_{L0} \le AW$），因为在 AW 范围内，较大的值与现有值在性能上等效。

5) 案例研究： 图 6 使用循环顺序 $n_{L0} \to k_{L1} \to n_{L1}$（其中包含 $N_{L0}=4$、$K_{L1}=2$、$N_{L1}=2$）。对于 $n_{L0}=0$，第一行缓冲区在 $AW = 4$ 列中依次放置 $W_{VN}(0,0)$ $W_{VN}(0,4)$ $W_{VN}(1,0)$ $W_{VN}(1,4)$。相同模式会在 $n_{L0} = 1,\ 2,\ 3$ 中重复。此案例研究明确表明，SetWVNLayout 使用紧凑的 VN 级描述符（分区因子 +排序）编码了完整的合法布局空间。

G. 执行模型

1) MINISA 指令集架构的作用：MINISA 将 配置、数据移动和计算触发清晰地分离开来。

- **仅配置指令集架构：** SetIVNLayout 和 SetWVNLayout 仅用于编程内部控制/状态寄存器。它们不会移动数据，也不会启动计算。

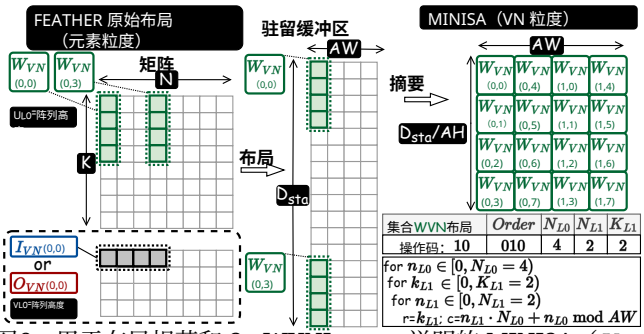


图6：用于布局规范和 SetWVNLayout 说明的 MINISA（$K = 8$, $N = 8$, $AH = AW = 4 \Rightarrow K_{L0} = AH$）。要点：MINISA 以 VN 粒度组织布局。

- **内存移动指令集架构：** Load 和 Write 会触发片外存储器与流式/驻留缓冲区之间的数据传输。SetOVNLayout 还会在分块边界管理输出缓冲区生命周期：在累加前初始化输出分块，并根据数据流（IO-S：流式，WO-S：驻留）将已完成的分块从输出缓冲区提交到下一个操作数缓冲区。

- **计算触发指令集架构：** FEATHER+ 仅在接收 E.Streaming 和 E.Mapping 时触发片上活动。每次发射都会将驻留张量从片上缓冲区加载至 NEST，并在当前布局下对一个分块进行计算。

- **子分块执行：** 布局配置可在多对（执行映射、ExecuteStreaming）发射之间复用，使一系列子分块能够对相同输出作出贡献。

2）执行模型：单个层的规范轨迹为：

$$\texttt{Set*VNLayout} \ \to \ \{\texttt{E.Mapping/E.Streaming}\}^{\times T}$$

其中， T 表示该层消耗当前已加载的 I_{VN}/W_{VN}所需的计算瓦片数量。布局指令首先定义操作数放置并初始化输出分块，随后ExecuteMapping序列会处理具有唯一映射的所有分块。

对于连续的层，第 i 层的输出成为第 i+1层的输入。因此，SetIVNLayout 仅在第一层发射一次，而第 i层的 SetOVNLayout 会复用为第 i+1层的 SetIVNLayout，这一步可以选择性跳过。每个新层仍会发射其自身的 SetWVNLayout、SetOVNLayout 以及一批执行映射指令。

3) 演示案例研究： 图 7 展示了矩阵乘法的这一执行语义。假设SetIVNLayout、SetWVNLayout 和 Load已经建立了布局并加载了所需的 VN。随后，片上张量由两个连续的 ExecuteMapping/Streaming 指令（❸）在同一个 $AH \times AW$ NEST 上进行处理。两个分块共享相同的 SetOVNLayout，并将 P_{VN}累加到同一个最终的O_{VN}中。由于 O_{VN}的布局在两次映射中保持不变，来自两个分块的 P_{VN}都能保证累加到一致的位置，从而完成完整的 O_{VN}，无需额外的输出重新布局或中间数据移动。

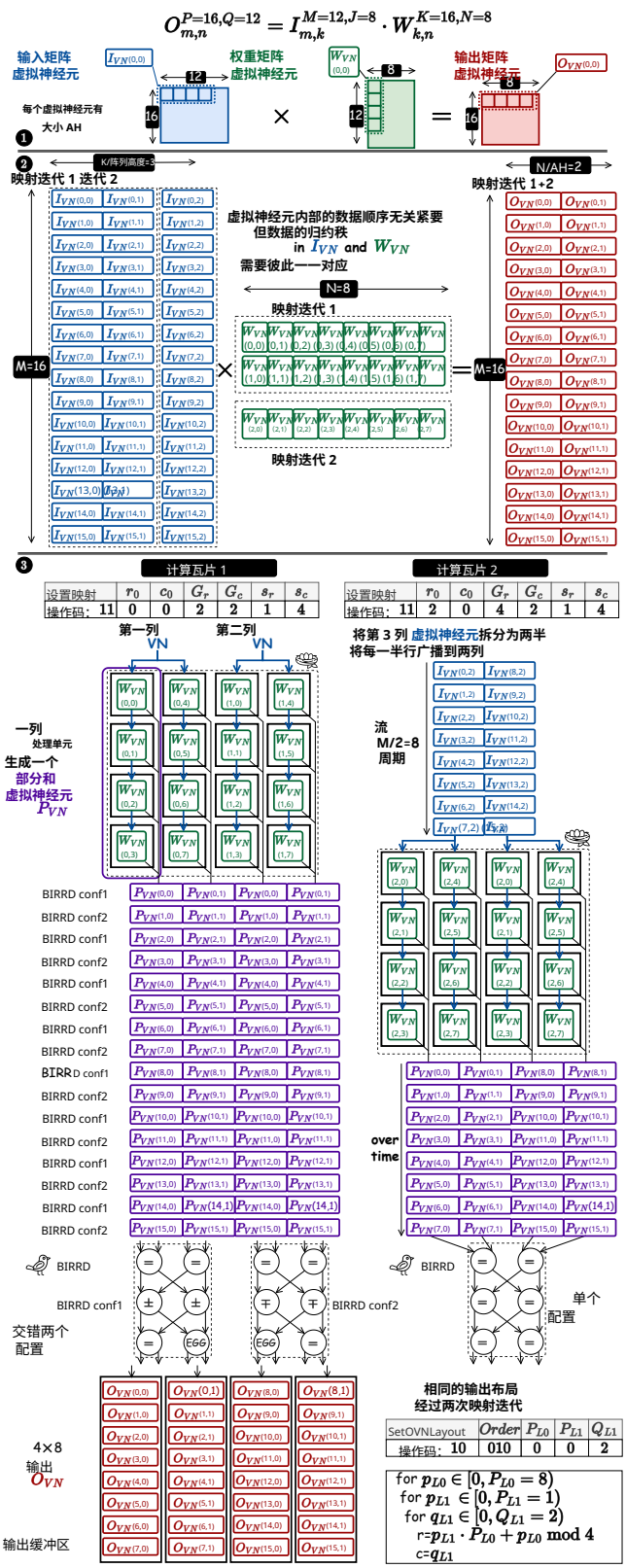


图 7：在 MINISA 下将矩阵乘法映射到 FEATHER+(阵列高度×地址宽度=4×4NEST)，作为 2 个计算分块。

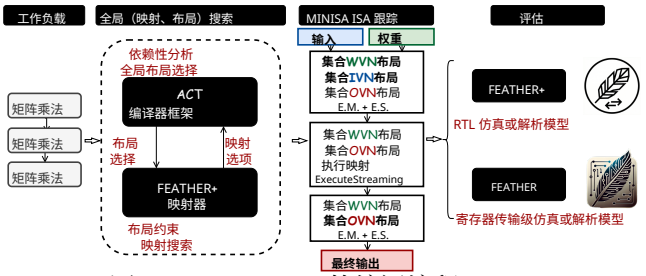


图 8：FEATHER+ 的编译流程。

V. 编译流水线

A. 概述

对于包含一系列矩阵乘法的工作负载，我们提供了一个解析式映射与布局协同搜索框架 FEATHER+映射器。选定的（数据流、布局）会被确定性地转换为 MINISA 指令集架构的轨迹，随后进一步降级为 FEATHER+“微指令”。

对于表示为层链（或有向无环图）的工作负载，其中包含矩阵乘法和激活（例如 Softmax），我们将 FEATHER+ 映射器作为布局约束映射搜索框架集成到 ACT 生态系统 [6], [7], 中，如图 8 所示。具体而言，ACT 首先执行图级分析，并识别布局灵活区域——即在保持正确性的同时可以更改张量布局的子图，但须满足区域边界处的任何布局约束。对于此类区域中的每一层，ACT 都会调用 FEATHER+ 映射器执行布局约束映射搜索。选定的逐层映射选项会发送回 ACT，以确定全局（映射、布局）组合中延迟最低的方案。最终方案将被降级为 MINISA 指令集架构跟踪，并在 FEATHER / FEATHER+ 寄存器传输级仿真或解析仿真模型中执行。

B. *FEATHER*+映射器的整体流程

FEATHER+映射器采用映射优先、布局其次的搜索策略。它首先枚举候选映射，然后推导出能够实现各映射且不会产生存储体冲突的缓冲区布局。这种排序基于一个关键观察：尽管映射与布局的完整联合空间很大，但映射空间本身可以仅用三个参数简洁地表示：(1) 计算分块大小，(2) VN 组构成，以及 (3) 列复制。固定一个映射候选后，剩余的布局搜索仅限于布局排序和第 0 级分区因子，其范围要小得多，且可以高效地检查其可行性。在所有可行的 {映射、布局} 组合中，映射器会返回估算延迟最低的组合。图 7 和图 9 以一个分为两个计算瓦片的矩阵乘法为例，展示了这一流程。从映射器的角度来看，IO-S 等同于转置后的 WO-S 配置。

1）步骤 *1*——将工作负载转换为虚拟神经元：首先，将输入工作负载（如卷积或矩阵乘法）改写为一组虚拟神经元（VN），例如图 7 中的 ❷。沿归约维度将每个张量划分为固定长度的 VN，每个 VN 的长度至多为 AH

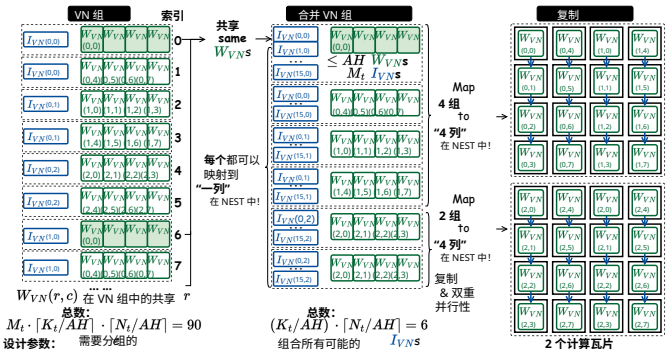


图 9：映射器示意图，展示如何将图 7 中的“VN 组”和“组合”映射到下方❷处的两个计算瓦片，以及如何进行“复制”。

从而适配每个处理单元的本地寄存器。VN 是映射器操作的标准软件抽象。

在这种表示下，每个张量都变为一个二维虚拟神经元数组，由一个非归约秩和一个归约瓦片秩进行索引，即 I_{VN}、W_{VN} 和 O_{VN} 的二维逻辑数组。执行过程随后简化为：将共享同一归约秩索引的一个 I_{VN} 和一个 W_{VN} 映射到同一个处理单元上，以进行点积。例如，$I_{VN}(*, 0)$ 只能与 $W_{VN}(0, *)$ 配对，其中 $*$ 表示任意非归约索引。换言之，工作负载将成为一个逻辑通用矩阵乘法（$O^{P=M}_{m}{}^{Q=N}_{n} \doteq I^{M=M}_{m}{}^{J=K}_{k} \cdot W^{K=K}_{n}{}^{N=N}_{k}$，$M$，$K$，$N$），其秩形状为（$M$，$K$，$N$）。

2) 步骤 *2*——划分工作负载：分块有两个目的：将张量限制在片上缓冲区容量范围内，并提供控制手段，以便调整瓦片大小，使片外传输与计算重叠。对于通用矩阵乘法，一个分块因此表示形状为 $M_t \times K_t \times N_t$ 的子问题，由一次或多次 NEST 调用执行。

3) 步骤 *3*——形成 *VN* 组：每个 NEST 列都是一个独立的映射单元，由 AH 个处理单元组成，每个处理单元存储一个 W_{VN}，在 WO-S 下，或 I_{VN}，在 IO-S 下。因此，映射器将 VN 分组成 *VN* 组，每个 VN 组包含一个流式 I_{VN}，以及最多 AH W_{VN} 个可使用相同 I_{VN} 的分块（映射到同一列时）。因此，VN 组是单列中最小的可调度单元。对于一个通用矩阵乘法分块，VN 组的数量为 $\lceil \frac{K_t}{AH} \rceil \cdot \lceil \frac{N_t}{AH} \rceil$。

在一个 VN 组内，所有分块共享相同的归约秩索引，但映射器可以选择不同的非归约索引，将其放入同一组中。此选择定义了第二个映射参数。

4) 步骤 *4*——跨流式输入合并 *VN* 组：随后合并 VN 组，以捕获驻留 W_{VN} 在多个流式 I_{VN} 之间的复用。一个合并 *VN* 组包含所有 I_{VN}，它们在一个 NEST 列上执行时，复用同一组最多 AH W_{VN} 个。换言之，在切换到不同的驻留-W_{VN} 配置之前，它指定分配给一列的完整工作负载。

对于一个通用矩阵乘法分块，组合 VN 组的数量为 $\lceil \frac{K_t}{AH} \rceil \cdot \lceil \frac{N_t}{AH} \rceil$。同一组合 VN 组中的所有 I_{VN} 和 W_{VN} 共享相同的归约瓦片索引。此步骤明确了原本会隐藏在众多独立 VN 组之间的复用机会。

5) 步骤 *5* - 选择列复制：由于 NEST 具有 AW 列，映射器必须决定同时执行多少个组合 VN 组，以及是否应将其中一些组复制到多个列上。在一种极端情况下，映射器选择 AW 个不同的组合 VN 组，并为每列分配一个组。在另一种极端情况下，它可能选择少于 AW 个组，并将其 W_{VN} 复制到多个列上，从而允许对同一组中的不同子集 I_{VN} 进行并行处理。该复制因子是第三个映射参数。

例如，如果一个分块生成六个组合 VN 组和 $AW = 4$，映射器可以在第一次调用中调度四个不同的组，然后在第二次调用中以复制因子二调度剩余的两个组。当可用组的数量无法与物理列数整除匹配时，这种选择可以提高列利用率，也说明了为什么即使在单个逻辑张量块内，FEATHER+ 也能从可重构映射中受益。

6) 第 *6* 步 - 搜索可行布局：上述三个映射参数会生成一个候选映射池。对于每个候选映射，映射器会搜索能够实现该候选映射且不会产生存储体冲突或端口冲突的张量布局。布局搜索仅限于布局顺序和第 0 级分区因子，并且会根据以下可行性条件检查每个候选映射。

a) 缓冲区容量合法性：所述 I_{VN}、W_{VN} 和 O_{VN} 必须适配流式、驻留和输出缓冲区。

b) 流式／驻留缓冲区合法性：由 `ExecuteMapping`、`SetIVNLayout` 和 `SetWVNLayout` 组成的组合，在读取 I_{VN} 或 W_{VN} 时不得产生存储体冲突。

c) 输出缓冲区合法性：所选的 SetOVNLayout 不得在下引入输出缓冲区端口冲突。

任何违反上述条件的候选方案都会被舍弃。*7)* 步骤 *7* - 生成 *MINISA ISA* 跟踪：在所有可行的 {映射、布局} 组合中，映射器会选择预估端到端延迟最低的组合。随后，所选方案会被确定性地转换为一条 MINISA ISA 跟踪，其中包含 ExecuteMapping、SetWVNLayout、SetIVNLayout 和 SetOVNLayout 指令。最终，FEATHER+ 解析性能模型会对生成的跟踪进行评估，以估算延迟和利用率。

对于多层工作负载，映射器还会确保层间布局兼容性：第 i 层的输出布局必须与第 $i+1$ 层所需的输入布局相匹配。随后，映射器会搜索所有保留下来的跨层组合，并选择整体延迟最低的方案。

VI. 评估

A. 评估设置

方法。我们采用两级仿真框架。周期精确的 FEATHER+ 寄存器传输级模型（同时支持 MINISA 和微指令）用于验证功能正确性和时序。经寄存器传输级验证的 FEATHER+映射器结合解析模型，将研究扩展至更大规模的阵列。

配置。我们使用 FEATHER+，并通过扫描 $\in \{(4,\ 4/16/64)、(8,\ 8/32/128)、(16,\ 16/64/256)\}$，采用 $(AH,\ AW)$。

表 IV：矩阵乘法的工作负载规格。

工作负载	形状为 $O^{P=M}$ 的矩阵乘法
全同态加密：BConv	$(65536\times K)\cdot(K\times N),\ K\in[28,60],\ N\in[72,160]$ (41 种形状)
全同态加密：数论变换	$J=K=N\in\{1024,2048,4096\}, M\in\{64,128,256\}, M\le K/16$
零知识证明：数论变换	$J=K=N\in\{8192,16384,32768\}, M\in\{K/32,K/16\}$
GPT-oss	$M=2048,\ (J=K,\ N)\in\{(64,\ 2048),\ (2880,\ 4096/5120/201088),\ (4096,\ 2880)\}$

表 V：FEATHER+ 的实验设置。

$(AH,\ AW)$	On-芯片容量 (MB)	MINISA 指令集架构位宽		
	(StrB/ StaB, OB, 指令)	Set*VNLayout E.Mapping E.Streaming		
(4, 4/16/64)	(1.6, 0.8, 0.5)	42/40/38 位	81/83/85 位	57/51/45 位
(8, 8/32/128)	(6.4, 3.2, 1.0)	43/41/39 位	86/88/90 位	58/52/46 位
(16, 16/64/256)	(25.6, 12.8, 2.0)	44/42/40 位	91/93/95 位	59/53/47 位

片上数据静态随机存取存储器的规模随 AH 变化，并划分为流式（40%）、驻留（40%）和输出（20%）缓冲区。专用指令缓冲区（0.5MB、1MB、2 MB）由固定的片外指令接口提供服务，带宽为 9 B/cycle。片外数据带宽建模为：输入/权重为 AW B/cycle，输出为 $4AW$ B/cycle。

工作负载。我们评估了来自三个领域的 50 个通用矩阵乘法内核：大语言模型推理（GPT-OSS 20B）、同态加密（全同态加密）自举 [2]，以及来自全同态加密 [22]和零知识证明（ZKP） [21]的数论变换内核。形状列于表 IV。

基线。 对于每种工作负载和阵列规模，我们在相同映射下比较以下两种配置。

- **微指令（基线）：** 采用显式、细粒度控制，每个开关和处理单元都按周期进行配置。
- **MINISA（本文方案）：** 基于 VN 的 MINISA 指令。

指标。 (1)延迟：在真实轨迹上使用 Nsight/JAX 性能分析器测量图形处理器/张量处理器的延迟；FEATHER+的结果来自周期精确的寄存器传输级实现。我们报告以下情况下的最佳延迟：图形处理器上采用分块、跨步或连续布局的 CUDA 内核；以及张量处理器上将 (M, N) 分片到八个张量核心的最佳方案。(2)利用率：矩阵乘法端到端执行过程的平均计算利用率。

B. 控制开销评估

我们首先量化并比较图 12 中 MINISA 与基线的总指令字节数和读取延迟。

1) 指令存储量缩减：在全部 50 个工作负载中，微指令所占的存储开销最高可达数据的 $100\times$ 倍。由于微指令流必须显式配置开关和地址互连网络。MINISA 将此类开销降低了几何平均值 $2\times10^4\times$ at $(AH,\ AW)$ =(16，256)。随着工作负载规模增大，所需指令也更多，因此开销降低幅度大致随工作负载规模增长。

2) 指令获取延迟降低：我们评估了指令压缩如何转化为 FEATHER+ 不同规模下的端到端加速比（图 10）。

- **小规模（≤ 64 处理单元）：** 在（$AH,\ AW$）=（4，4），（4，16），（8，8）时，微指令消耗足够小，可被计算掩盖，因此加速比为 1。
- **大规模（>64 处理单元）：** 随着阵列规模增大，BIRRD 指令和缓冲区地址分别按 $O(AW\log_2(AW))$ 和 $O(D\times AW)$ 增长。基线中，指令获取导致的停顿周期占比从 (4，64) 时的**75.3%** 上升到 (16，256) 时的**96.9%**。

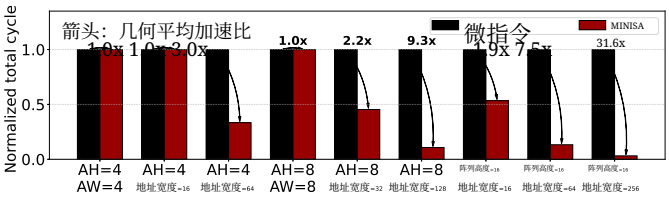


图 10：端到端加速比和停顿分析。随着阵列规模增至 16×256，MINISA 可实现高达 $31.6\times$的加速比。**结论：** 性能提升源于消除了指令获取停顿。微指令基线在大规模下受指令限制，而 MINISA 在所有配置下都将指令获取停顿降至零。

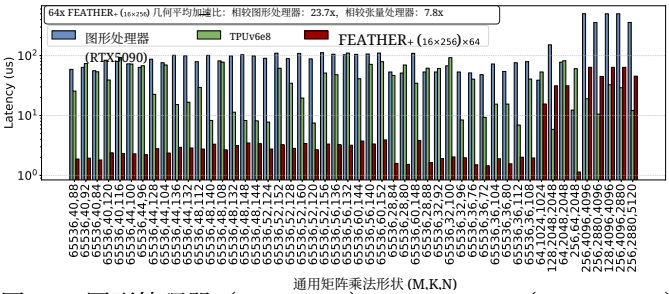


图 11：图形处理器（RTX 5090）、TPUv6e-8（$256\times256\times8$）和 FEATHER+（（16×256）$\times64$）之间的延迟比较。

由于 MINISA 在所有测试规模下都消除了获取停顿，几何平均加速比会随着规模增大而提升： $1.9\times$在16×16时为 $7.5\times$，在 16×64 时为 $31.6\times$，并在 16×256时最高可达。这些结果证明，VN 抽象能够通过简洁的指令集架构，在硬件中隐藏复杂的重新配置过程。

C. 性能评估

1）与业界基线（张量处理器和图形处理器）进行比较：我们将 FEATHER+ 与 RTX5090 和 TPUv6e 进行比较。所有设备的功耗均调整至大致相同的 575W：八个 TPUv6e 张量核心，以及 64 个 FEATHER+16×256 实例，以 8×8 网格互连，如图 11 所示。

FEATHER+ 实现了 $23.7\times$ （相比 RTX5090）以及 $7.8\times$ （相比 TPUv6e）的几何平均加速比。其关键原因在于灵活性较低的图形处理器/张量处理器存在粒度不匹配问题（例如，对于 INT8，TPUv6e 以最小粒度 $8\times256\times256$处理通用矩阵乘法，而 RTX5090 则以 $16\times32\times8$处理）。当通用矩阵乘法的形状无法整除这些固定粒度时，计算资源就会利用不足。FEATHER+支持以每个处理单元列 $T\times AH\times AH$ 为粒度进行矩阵乘法，从而在运行时适配通用矩阵乘法的多样形状，其中 $T\in[1,M]$ 是每行需要流式传输的 I_{VN}总数。

2）延迟分解： 我们进一步利用图 13 中代表性工作负载的周期级分解，对架构效率进行剖析。我们将执行时间划分为四个相互重叠的组成部分：计算、输入/权重流式传输（加载输入/*W*）、从输出缓冲区到流式/驻留缓冲区的输出移动（输出→流），以及片外输出传输（存储输出）。

对不规则形状的鲁棒性。计算利用率曲线（图 11 中的红线）在各种张量形状下始终保持较高水平，

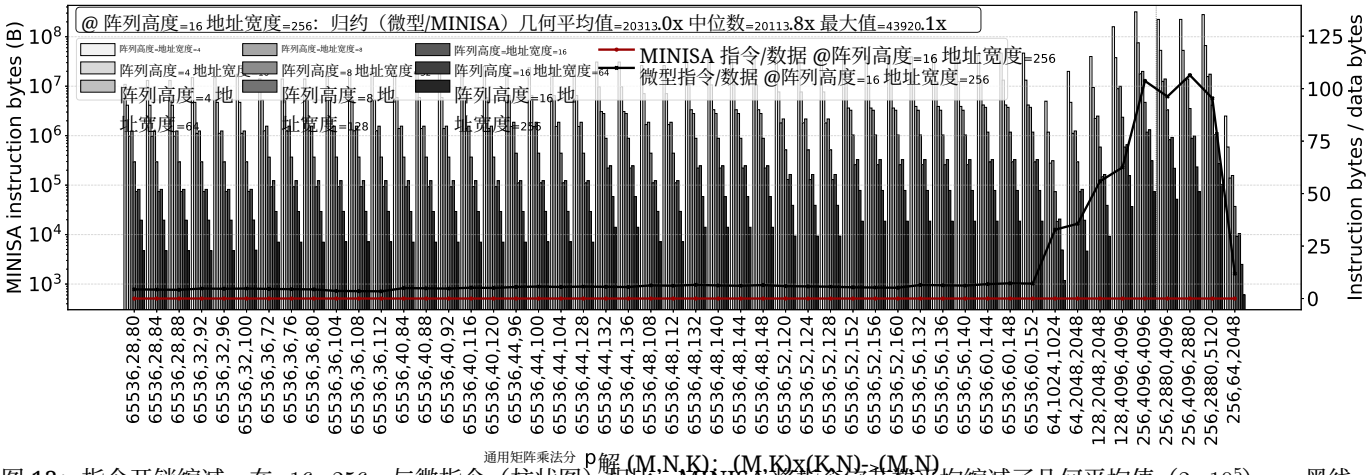


图 12：指令开销缩减。在 16×256，与微指令（柱状图）相比，MINISA 将指令字节数平均缩减了几何平均值（2×10^5）×。黑线和红线分别表示基线和 MINISA 的指令与数据比率。**要点：** MINISA 最多可将指令量缩减 $4.4\times10^5\times$，使指令加载不再处于关键路径上。

这表明 FEATHER+ 对形状不规则性具有很强的鲁棒性。

• 不规则工作负载：对于形状无法整除计算粒度的 FHE 和零知识证明计算内核（例如 $K=10$ 或 $N=21$），FEATHER+ 仍能维持 $>60\%$ 的平均计算利用率。这与计算利用率仅为 3% 的刚性脉动阵列形成鲜明对比；在后者中，维度不匹配通常会引入填充开销，导致大多数处理单元处于空闲状态。

• 规则工作负载：对于维度与 TPUv6e 执行粒度完全对齐的通用矩阵乘法（例如，K，$N\in\{1024,2048\}$），所有架构都能接近峰值利用率。在这种情况下，FEATHER+ 的速度约慢 30%，原因是重配置开销，而非利用不足。即便如此，其开销仍低于 RTX5090，这表明 MINISA 通过紧凑的控制路径和较低的指令开销，有效摊销了重配置成本。

要点： 可重配置性使 FEATHER+即使面对高度不规则的工作负载，也能保持较高的计算利用率，而刚性架构则会遭受填充导致的低效。MINISA 通过最大限度地降低重配置开销，进一步控制了这种灵活性的成本。

D. 可扩展性消融研究

我们分析 FEATHER+的性能和资源如何随阵列宽度（AW）和阵列高度（AH）扩展。

1) 扩展 AW：更多独立并行性： AW 表示 NEST 列的数量。由于各列独立运行，增加 AW 可提升列级并行性，而不改变每列的执行粒度。

在 $AH=16$下，将 AW 从 64 扩展到 256 可实现近乎线性的扩展：FEATHER+ 在各项基准测试中实现了约 $4\times$ 的平均加速比，同时计算利用率几乎保持不变（图 13）。因此，更宽的阵列可直接转化为吞吐量提升，而不会带来额外的利用不足。

扩展 AW 的硬件成本也很有利。NEST 和片上缓冲区呈线性增长，即 $O(AW)$。互连开销增长更快，但仍低于二次方：BIRRD 的扩展规模为 $O(AW\log AW)$，而分发网络则受限于 $O(AW^2)$。总体而言，扩展 AW 是增加吞吐量的高效方式，其成本主要由互连开销决定。

2) 扩展 AH：受时间复用限制影响的更高并行性： 增加 AH 会提升整体 MAC 数和并行性，但也会提高工作负载粒度；要充分利用一个处理单元列，需要进行 $1\times AH\times AH$ 矩阵乘法。

具体而言，每个处理单元执行一个 AH元素的局部点积。 AH 决定每个处理单元所支持的最大 VN 大小，从而确定 FEATHER+的计算粒度。一列中的所有处理单元复用同一个流式张量，因此扩展 AH 会增加时间复用限制。

要使一个 $AH\times AW$ 阵列实现完全利用，需要 $VN_{\text{size}}=AH$。当 $VN_{\text{size}}<AH$时，FEATHER+ 仅激活 $VN_{\text{size}}\times AW$个处理单元，并且每个处理单元仅使用 VN_{size} 个本地寄存器，跳过其余行，以降低驻留张量加载延迟和流水线延迟。

这种权衡也体现在性能上。对于 $AW=64$，将 AH 从 4 扩展到 16 可带来 $2.6\times$–$4\times$的加速比，具体取决于工作负载大小。增益来自更大的点积和更高的列内并行性，但更大的 AH 也会提高实现完全利用率所需的最小工作负载粒度。因此，扩展 AH 可提升峰值吞吐量，但会使利用率对 VN 大小更加敏感。

在资源方面，本地存储呈二次增长，即 $O(AH^2)$；乘法器数量呈线性增长，即 $O(AH)$。

要点。增加 AW 可扩展独立并行性。增加 AH 会提高并行性，同时带来时间复用需求，并增加计算粒度。

E. 资源开销评估

最后，我们量化了支持动态工作负载所需的 FEATHER+ 架构改进（包括全互连分发）的硬件成本。表 VI 汇总了面积结果。

与 FEATHER 相比， [23], FEATHER+ 仅增加了最多 7% 的资源开销，因为引入的全互连分发网络的面积开销可由分布式寄存器和计算资源分摊。

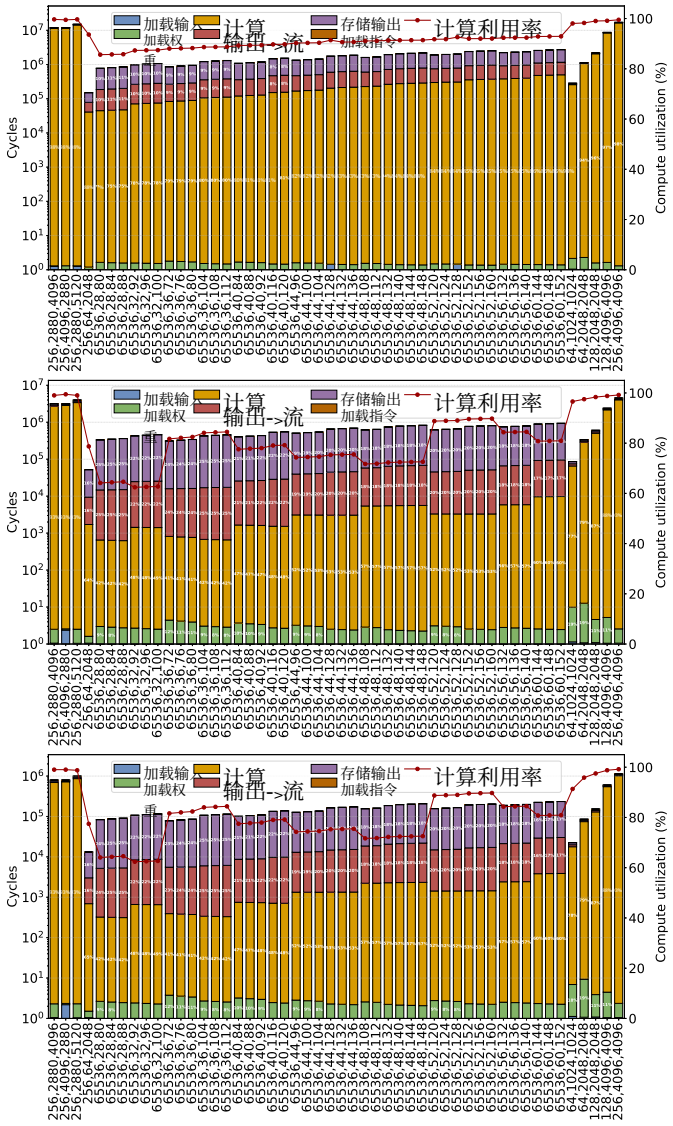


图 13：代表性工作负载的延迟分解和计算利用率。**要点**：采用 MINISA 的 FEATHER+ (4×64，上图)、(16×64，中图)和 (16×256，下图)，能够在所有不规则工作负载形状下成功利用可用的处理单元，例如 K=10 和 $K = 2^n$，$n \in \mathbb{Z}^+$。

VII. 相关工作

人工智能加速器指令集。商用人工智能加速器通常采用驱动固定功能单元的单体式指令集，以最大限度地提升标准深度学习算子的效率。谷歌的 TPU [8]、Meta 的 MTIA [4]，以及亚马逊的 Trainium [1] 等架构，旨在执行粗粒度、规则的矩阵乘法。这些僵化的粒度缺乏所需的细粒度差异化可编程性，因而无法支持同态加密（HE）和零知识证明（ZKP）等新兴工作负载中存在的不规则数据流和复杂代数结构。

可重构数据流加速器。为应对张量形状和算子的多样性，可重构架构（如 MAERI [13],SIGMA [18]，和 Flexagon [16]）引入灵活互连网络以支持多样化的数据流 [17]。PolyGraph [3], DSA- Gen [24],和 Over-Gen [15]等后续工作则探索通过覆盖层生成来映射各种计算内核。然而，这些设计通常将计算与存储器布局分离，导致显著的数据编组开销。FEATHER [23] 通过实现数据流与数据布局的低成本协同切换来解决这一问题。然而，至关重要的是，先前的可重构基线（包括 *FEATHER*）都依赖细粒度微配置。随着阵列规模扩大，这会造成过高的控制开销和指令获取停顿，成为端到端性能的瓶颈。MINISA 专门针对这一可扩展性危机，通过将抽象层级提升至虚拟神经元，消除冗余控制流量，同时保留架构灵活性。

表 VI：台积电 28nm 工艺中 FEATHER (F) 与 FEATHER+(F+) 的布局布线后面积(μm^2)和功耗（毫瓦）比较。

设置	面积 (μm^2)			功耗 (毫瓦)		
	F	F+	增加 ↑	F	F+	增加 ↑
4×4	70598	71573	1.38%	44.59	45.34	1.67%
8×8	174370	176573	1.26%	108.97	110.49	1.39%
16×16	476174	482044	1.23%	293.47	297.72	1.45%
4×64	1259903	1352697	7.37%	854.77	915.14	7.06%
8×128	3198595	3441146	7.58%	2240.27	2350.88	4.94%

我们将所有片上缓冲区的深度设为 64，并在布局布线时将其实现为寄存器。在实际部署中，较大的缓冲区则应使用 SRAM 宏。

用于密码学的 AI ASIC（HE 和 ZKP）。密码学原语的高计算成本推动了硬件支持走向分化。先前的研究表明，通过将高精度模运算转换为形状各异的低精度稠密矩阵乘法，利用人工智能专用集成电路来实现 HE [22]和 ZKP 原语 [21] 具有显著优势。MINISA 和 FEATHER+ 提升了小型矩阵乘法的计算利用率，尤其适用于尺寸与硬件计算维度不对齐的情况。

VIII. 结论

MINISA 是首个支持在运行时协同切换数据流和数据布局、且指令开销可忽略不计的指令集架构，使单个加速器能够高效支持形状差异显著且不规则的工作负载。其关键在于虚拟神经元（VN）抽象，它将重配置的粒度从单个元素提升至硬件原子点积单元的粒度。该抽象精确地体现了硬件的可重配置性，不会产生额外冗余开销，并以编译器友好的形式统一映射和布局控制。结合 FEATHER+ 架构，MINISA 使重配置在实践中切实可行，并能够高效执行矩阵乘法，处理来自全同态加密、零知识证明和人工智能等不同领域的各种形状。

IX. 致谢

本研究部分由 ACE 资助。ACE 是 JUMP 2.0 的七个中心之一；JUMP 2.0 是由美国国防高级研究计划局赞助的半导体研究公司（SRC）项目。我们感谢 Niansong Zhang、Zhiru Zhang 及各位审稿人提出的宝贵意见。

参考文献

[1] AWS。（2025年）AWS NKI指令集架构。 [在线]。可访问：https://awsdocs-neuron.readthedocs-hosted.com/en/latest/nki/api/nki.isa.html [2] A. A. Badawi、J. Bates、F. Bergamaschi、D. B. Cousins、S. Erabelli、N. Genise、S. Halevi、H. Hunt、A. Kim、Y. Lee、Z. Liu、D. Micciancio、I. Quah、Y. Polyakov、S. R.V.、K. Rohloff、J. Saylor、D. Suponitsky、M. Triplett、V. Vaikuntanathan 和 V. Zucca，《OpenFHE：开源全同态加密库》，密码学电子印本文档库，论文 2022/915，2022年，https://eprint.iacr.org/2022/915。

[3] [ ] 2022 915 在线。可访问：https://eprint.iacr.org/ / V. Dadu、S. Liu 和 T. Nowatzki，《PolyGraph：揭示灵活性对图处理加速器的价值》，发表于第*48*届计算机体系结构国际年会论文集，ISCA '21 系列。IEEE出版社，2021年，第 595–608 页。 [在线]。可访问：https://doi.org/ 10 1109 52012 2021 00053 。/ISCA。/ A. Firoozshahian、J. Coburn、R. Levenstein、R. Nattoji、A. Kamath、O. Wu、G. Grewal、H. Aepala、B. Jakka、B. Dreyer、A. Hutchin、U. Diril、K. Nair、E. K. Aredestani、M. Schatz、Y. Hao、R. Komuravelli、K. Ho、S. Abu Asal、J. Shajrawi、K. Quinn、N. Sreedhara、P. Kansal、W. Wei、D. Jayaraman、L. Cheng、P. Chopda、E. Wang、A. Bikumandla、A. Karthik Sengottuvel、K. Thottempudi、A. Narasimha、B. Dodds、C. Gao、J. Zhang、M. Al-Sanabani、A. Zehtabioskuie、J. Fix、H. Yu、R. Li、K. Gondkar、J. Montgomery、M. Tsai、S. Dwarakapuram、S. Desai、N. Avidan、P. Ramani、K. Narayanan、A. Mathews、S. Gopal、M. Naumov、V. Rao、K. Noru、H. Reddy、P. Venkatapuram 和 A. Bjorlin，《MTIA：面向 Meta 推荐系统的第一代硅芯片》，发表于第*50*届计算机体系结构国际年会论文集，ISCA '23 系列。美国纽约：计算机协会，2023年。

[4] [ ] 10 1145 3579371 3589348 在线。可访问：https://doi.org/ 。/ 。M. Gilbert、Y. N. Wu、A. Parashar、V. Sze 和 J. S. Emer，《LoopTree：支持融合层数据流加速器的探索》，发表于*2023*年*IEEE*系统与软件性能分析国际研讨会（*ISPASS*），2023年，第 316–318 页。[6] D. Jain、M. Frigo、J. Arora、A. Pardeshi、Z. Wang、K. Patel 和 C. Mendis，*TAIDL*：张量加速器指令集架构定义语言及可扩展测试预言机自动生成。美国纽约：计算机协会，2025年，第 1316–1333 页。 [在线]。可访问：https://doi.org/ 10 1145 3725843 3756075 。/ 。D. Jain、A. Pardeshi、M. Frigo、K. Patel、K. Khulbe、J. Arora 和 C. Mendis，《ACT：根据张量加速器指令集架构描述自动生成编译器后端》，2025年10月。[在线]。可访问：https://arxiv.org/abs/ 2510 09932 。

[5] [ ] 10 1145 3579371 3589348 在线。可访问：https://doi.org/ 。/ 。M. Gilbert、Y. N. Wu、A. Parashar、V. Sze 和 J. S. Emer，《LoopTree：支持融合层数据流加速器的探索》，发表于*2023*年*IEEE*系统与软件性能分析国际研讨会（*ISPASS*），2023年，第 316–318 页。[6] D. Jain、M. Frigo、J. Arora、A. Pardeshi、Z. Wang、K. Patel 和 C. Mendis，*TAIDL*：张量加速器指令集架构定义语言及可扩展测试预言机自动生成。美国纽约：计算机协会，2025年，第 1316–1333 页。 [在线]。可访问：https://doi.org/ 10 1145 3725843 3756075 。/ 。D. Jain、A. Pardeshi、M. Frigo、K. Patel、K. Khulbe、J. Arora 和 C. Mendis，《ACT：根据张量加速器指令集架构描述自动生成编译器后端》，2025年10月。[在线]。可访问：https://arxiv.org/abs/ 2510 09932 。

[7] N. P. Jouppi、D. Hyun Yoon、M. Ashcraft、M. Gottscho、T. B. Jablin、G. Kurian、J. Laudon、S. Li、P. Ma、X. Ma、T. Norrie、N. Patil、S. Prasad、C. Young、Z. Zhou 和 D. Patterson，《三代产品演进带来的十点经验：谷歌 TPUv4i 工业产品》，发表于*2021*年*ACM/IEEE*第*48*届计算机体系结构国际年会（*ISCA*），2021年，第 1–14 页。[9] N. P. Jouppi、G. Kurian、S. Li、P. Ma、R. Nagarajan、L. Nai、N. Patil、S. Subramanian、A. Swing、B. Towles、C. Young、X. Zhou、Z. Zhou 和 D. Patterson，《TPU v4：一台支持嵌入硬件的光学可重构机器学习超级计算机》，2023年。[10] N. P. Jouppi、D. H. Yoon、G. Kurian、S. Li、N. Patil、J. Laudon、C. Young 和 D. A. Patterson，《用于训练深度神经网络的领域专用超级计算机》，《*ACM*通讯》，第 63 卷，第 7 期，第 67 78, 2020 [ ] 10 1145 3360307 – 页。在线。可访问：https://doi.org/ 。/ T. Krishna、H. Kwon、A. Parashar、M. Pellauer 和 A. Samajdar，《深度学习加速器中的数据编排》，2020年。[12] H. Kwon、P. Chatarasi、V. Sarkar、T. Krishna、M. Pellauer 和 A. Parashar，《Maestro：一种以数据为中心的方法，用于理解深度神经网络映射中的复用、性能和硬件成本》，《*IEEE Micro*》，第 40 卷，第 3 期，第 20–29 页，2020年。[13] H. Kwon、A. Samajdar 和 T. Krishna，《MAERI：通过可重构互连网络实现深度神经网络加速器上的灵活数据流映射》，发表于第*23*届编程语言与操作系统架构支持国际会议（*ASPLOS*）论文集，2018年。[14] L. Liu、J. Zhu、Z. Li、Y. Lu、Y. Deng、J. Han、S. Yin 和 S. Wei，《粗粒度可重构架构与设计综述：分类、挑战与应用》，《*ACM*计算综述》，第 52 卷，第 6 期，2019年10月。 [在线]。可访问：https://doi.org/10.1145/3357375

[8] [15] S. Liu、J. Weng、D. Kupsh、A. Sohrabizadeh、Z. Wang、L. Guo、J. Liu、M. Zhulin、R. Mani、L. Zhang、J. Cong 和 T. Nowatzki，《Overgen：通过领域专用覆盖层生成提升 FPGA 易用性》，载于第*55*届*IEEE/ACM*微体系结构国际研讨会论文集，系列号 MICRO '22。IEEE出版社，2023年，第35–56页。 [ ] 10 1109 56248 2022 00018 在线。可访问：https://doi.org/ 。/MICRO 。 。F. Munoz-Martınez、R. Garg、J. L. Abellan、M. Pellauer、M. E. Acacio 和 T. Krishna，《Flexagon：一种用于高效深度神经网络处理的多数据流稀疏-稀疏矩阵乘法加速器》，2023年。 [在线]。可访问：https://arxiv.org/abs/ 2301 10852 。

[9] [16] A. Parashar、P. Raina、Y. S. Shao、Y.-H. Chen、V. A. Ying、A. Mukkara、R. Venkatesan、B. Khailany、S. W. Keckler 和 J. Emer，《Timeloop：深度神经网络加速器评估的系统化方法》，载于国际系统与软件性能分析研讨会论文集（*ISPASS*），2019年。[18] E. Qin、A. Samajdar、H. Kwon、V. Nadella、S. Srinivasan、D. Das、B. Kaul 和 T. Krishna，《Sigma：一种采用灵活互连网络、用于深度神经网络训练的稀疏且非规则通用矩阵乘法加速器》，载于*2020*年*IEEE*高性能计算机体系结构国际研讨会（*HPCA*），2020年，第58–70页。[19] J. Seo、J. Tong、T. Krishna 和 H. Kwon，《面向实时多任务、多模型机器学习工作负载的受约束数据流加速器探索》，载于*2025*年*IEEE*系统与软件性能分析国际研讨会（*ISPASS*），2025年，第1–11页。[20] J. Strzeszynski、J. Tong、K. Lee、N. Xiong、A. Parashar、J. S. Emer、T. Krishna 和 M. Yan，《Squareloop：探索机器学习的最优身份验证模块策略》，载于第*14*届硬件与体系结构安全和隐私支持国际研讨会论文集，系列号 HASP '25。美国纽约：计算机协会，2025年，第37–45页。 [在线]。可访问：https://doi.org/ 10 1145 3768725 3768732 。/ 。J. Tong、J. Dang、S. Langowski、T. Huang、A. Ali、J. Kun、S. Devadas 和 T. Krishna，《Morph：利用人工智能专用集成电路实现零知识证明》，载于第*63*届*ACM/IEEE*设计自动化大会论文集，系列号 DAC '26。IEEE出版社，2026年。[22] J. Tong、T. Huang、J. Dang、L. de Castro、A. Itagi、A. Golder、A. Ali、J. Jiang、J. Kun、Arvind、G. E. Suh 和 T. Krishna，《利用ASIC人工智能芯片实现同态加密》，载于*2026*年*IEEE*高性能计算机体系结构国际研讨会（*HPCA*），系列号 HPCA'26，澳大利亚，2026年。[23] J. Tong、A. Itagi、P. Chatarasi 和 T. Krishna，《FEATHER：一种支持数据重排序、用于低成本片上数据流切换的可重构加速器》，载于第*51*届计算机体系结构国际研讨会论文集，系列号 ISCA '24。阿根廷：计算机协会，2024年。[24] J. Weng、S. Liu、V. Dadu、Z. Wang、P. Shah 和 T. Nowatzki，《Dsagen：合成可编程空间加速器》，载于*2020*年*ACM/IEEE*第*47*届计算机体系结构国际研讨会（*ISCA*），2020年，第268–281页。

附录

A. 摘要

该 artifact 包含面向 FEATHER+ 可重构加速器的 MINISA 工具链。其中包括：(1) FEATHER+ 映射器，用于搜索在 $AH \times AW$- FEATHAER+ 上处理 GEMM 工作负载的最优映射和布局选择；(2) 周期精确分析性能模型，以及包含 5 个引擎的异步执行模拟器；(3) 图形用户界面，通过逐周期动画展示 FEATHER+ 的工作方式；(4) 指令压缩分析，对比 MINISA 与显式微配置；(5) 与图形处理器（NVIDIA RTX 5090）和张量处理器（Google TPU v6e-8）基线的延迟比较。该 artifact 重现了所有评估图表：指令缩减率、相对于微指令的加速比、包含计算利用率的延迟分解，以及 FEATHER+ 与图形处理器／张量处理器的延迟比较。

表 VII：FEATHER+符号化通用矩阵乘法的映射器搜索参数 $O^{P=M,m\quad Q=N,n} = I^{M=M,m\quad J=K,k} \cdot W^{K=K,N=N}_{k,n}$。

类别	Knob	符号 / 选择集	含义
数据流选择 WO – S, IO – S	WO – S	$(M_s,\ K_s,\ N_s) = (M,\ K,\ N)$	在原始通用矩阵乘法上执行搜索。
	IO – S	$(M_s,\ K_s,\ N_s) = (N,\ K,\ M)$	在搜索前交换外层维度。
分块选择（确定分区因子）	输出行秩	$M_t \in \{AH, 2AH, 4AH, \ldots, M_s\}$	每个分块处理的输出行数。
	归约秩	$K_t \in \{AH, 2AH, 4AH, \ldots, K_s\}$	每个分块处理的归约深度。
	输出列秩	$N_t \in \{1, 2, 4, \ldots, N_s\}$	每个分块处理的输出列数。
布局选择	权重顺序	$o_{Wvn} \in \{0, 1, 2, 3, 4, 5\}$	用于 W_{VN} 布局的表 III 置换。
	输入顺序	$o_{Ivn} \in \{0, 1, 2, 3, 4, 5\}$	用于表 III 置换的 I_{VN} 布局。
	输出顺序	$o_{Ovn} \in \{0, 1, 2, 3, 4, 5\}$	用于表 III 置换的 O_{VN} 布局。
	复制因子	$d \in \{1, 2, \ldots, \lfloor AW/n_{\text{col}} \rfloor\}$	将每种列类型复制为多个列。
组合 VN 组 CG(K_g, N_t)	$W_{VN}(k_g, n_t)$ 列间 n_t 步长	∈ {块，跨步}	块：$s_r = 1$，$s_c = AH$。跨步：$s_r = n_{\text{col}}$，$s_c = 1$。改变并发 WVN 存储体访问模式。仅在 $n_{\text{col}} > 1$ 时不同。
	$I_{VN}(m_t, k_g)$ 列间 m_t 步长	∈ {交错，连续}	交错：$s_m = n_{\text{re}}$ p（跨步行）。连续：$s_m = 1$（连续块）。更改并发 IVN 银行访问模式。仅在 $n_{\text{rep}} > 1$ 时有所不同。

B. 工件检查清单（元信息）

- **算法：** 面向可重构加速器的 MINISA 指令集架构矩阵乘法编译；虚拟神经元（VN）抽象，并针对表 VII 中的选项进行穷举分块搜索。
- **程序：** Python 脚本：evaluate.py（映射布局联合搜索）、analyze.py（分析 + GPU/TPU 对比）、figure_drawer/*.py（绘图生成）
- **编译流水线：** 工作负载 → VN 组 → 组合 VN 组 →复制 → MINISA 跟踪记录 → 延迟。
- **数据集：** 来自三个领域的 50 个 GEMM 工作负载：全同态加密（FHE）、零知识证明（ZKP）和 ChatGPT 级开源大语言模型推理；预先收集的 GPU/TPU 基准延迟测量 • **运行时环境：** Linux（Ubuntu 22.04）；Python 3.10+；包含 numpy、pandas、matplotlib、pyyaml 的 Conda 环境
- **硬件：** 任何 x86-64 或 ARM64 机器，配备 ≥16GB 内存。无需图形处理器/张量处理器（基线数据已预先收集）。我们还提供图形处理器和张量处理器脚本。• **运行时状态：** 确定性分析模型（无随机性）。
- **执行：** 支持多线程（--jobs <n>）
- **指标：** 计算利用率（%）、周期细分（输入加载、权重加载、计算、输出至数据流、输出存储、指令获取）、指令字节数（MINISA 与微指令）、指令归约比率、延迟（μs）
- **输出：** 18 张 PDF 图表、6 个 CSV 数据文件（基准测试摘要、指令对比、图形处理器/张量处理器对比、利用率/归约/内存摘要）
- **实验：** 50 个工作负载 × 9 种阵列配置 = 450个评估点（阶段 1）；所有工作负载的图形处理器/张量处理器对比（阶段 2）；21 张出版级图表（阶段 3）
- **大约需要多少磁盘空间？：** ∼500MB
- **准备工作流需要多长时间（大约）？：** 5–10 分钟（设置 conda 环境）
- **完成实验需要多长时间（大约）？：** 完成全部评估（阶段 1）需要 3–10 小时；分析和绘图（阶段 2–3）需要 <2 分钟 • **是否公开？：** https://github.com/maeri-project/FEATHER • **是否已存档？：** https://doi.org/10.5281/zenodo.18921947

C. 描述

MINISA 框架可在任何支持 Python 的设备上运行。请参阅开源 GitHub 仓库中的 README.md 或已归档的 DOI 工件，以安装 MINISA 框架并运行：(1) 映射与布局协同搜索，(2) 布局约束的映射搜索，(3) 与图形处理器 / 张量处理器的对比。我们提供了 50 个通用矩阵乘法工作负载作为基准测试套件，如 §VI-A 所述。我们还提供了预先收集的图形处理器和张量处理器上的结果。

D. 评估与预期结果

(1) FEATHER+ 映射器，用于搜索在 $AH \times AW$-FEATHAER+ 下处理通用矩阵乘法/卷积工作负载的最优映射和布局选择；(2) 周期精确的解析性能模型，配备 5 引擎异步执行模拟器；(3) 图形用户界面，用逐周期动画展示 FEATHER+的工作方式；(4) 指令压缩分析，对比 MINISA 与显式微配置；(5) 与图形处理器（NVIDIA RTX 5090）和张量处理器（Google TPU v6e-8）基线的延迟比较。(1) 在 9 种配置下对全部 50 个工作负载进行（映射、布局）联合搜索：python -m minisa evaluate (2) FEATHER+图形用户界面：python -m minisa gui (3) 与张量处理器/图形处理器对比：python -m minisa analyze (4) 指令开销比较（MINISA 与微指令基线）：python -m minisa compare (5) 在 $AH \times AW$-FEATHAER+ 下对通用矩阵乘法/卷积进行（映射、布局）联合搜索：python -m minisa search (6) 在 $AH \times AW$-FEATHAER+ 下进行布局约束的映射搜索（通用矩阵乘法/卷积）：python -m minisa search --layout-constrained

（7）绘制图表：python -m minisa plot

E. 实验自定义

- **阵列配置：** 更改 --ah 和 --aw 以评估不同 FEATHER+尺寸。对于方形阵列，使用相同的 --aw（AW=AH）。
- **工作负载：** 提供包含不同工作负载的 CSV 文件。--csv 文件需包含 category、name、M、K、N 列。

F. 映射器设计空间

我们在表 VII 中总结了映射器设计空间，并应用启发式方法尽早剪除缺乏前景的候选项，以加快搜索。对于 50 个工作负载（$AH{=}AW{=}16$），在搭载 M5 Pro 的 MacBook Pro 上使用 16 个并行作业，联合映射和布局搜索可在 17 分钟内完成。