Xenos：以数据流为中心的优化，加速边缘设备上的模型推理

Runhua Zhang*、Hongxu Jiang*、Fangzheng Tian*、Jinkun Geng+、Xiaobin Li*、
Yuhang Ma*、Chenhui Zhu*、Dong Dong*、Xin Li*、Haojie Wang†
*北京航空航天大学，+斯坦福大学，†清华大学

arXiv:2302.00282v1 [cs.DC] 1 Feb 2023

摘要

边缘计算正逐渐成为模型推理的热门应用场景。然而，由于缺乏高度优化的推理框架，边缘设备（例如多核DSP、现场可编程门阵列等）上的推理性能受到效率低下的影响。以往的模型推理框架主要采用以算子为中心的开发方式，对基于边缘的推理加速不足。此外，以算子为中心的框架还会带来大量持续开发和维护成本。

本文提出 Xenos，可自动对计算图进行以数据流为中心的优化，并从两个维度加速推理。在垂直维度上，Xenos 开发了算子链接技术，通过重构算子间数据流来改善数据局部性。在水平维度上，Xenos 开发了 DSP 感知算子拆分技术，以提高多个 DSP 单元之间的并行度。我们的评估证明，垂直和水平数据流优化的有效性分别使推理时间减少了 21.2%–84.9% 和 17.9%–96.2%。此外，Xenos 的性能也优于广泛使用的 TVM，快 3.22×–17.92× 倍。我们还将 Xenos 扩展为一种分布式解决方案，称为 d-Xenos。d-Xenos 使用多个边缘设备协同执行推理任务，与单一设备相比，速度提升了 3.68×–3.78× 倍。

关键词：边缘设备推理、以数据流为中心的优化、DNN优化、数据局部性

1 引言

如今，边缘设备已得到广泛应用，并逐渐成为深度学习应用的普遍场景 [14, 16,27]。这些边缘设备的硬件资源配置各不相同（例如内存、计算能力等），通常还要求具备实时响应能力，即推理时长不能过长。现有解决方案（例如 TVM [8]）在这些平台上执行模型推理的效率低下，无法满足实时响应要求。

深入研究了众多典型的模型推理工作流后，我们发现推理效率低下主要有两个原因。

(1) 低效的数据流调度。模型推理的数据流调度会严重破坏内存局部性。以典型的 CNN 推理为例，每完成一层的推理，计算算子便会将特征图输出到共享内存区域。这些特征图元素将作为下一层的输入，并被馈送到多个 DSP单元进行后续推理计算。然而，数据布局（由前一层输出）与数据访问顺序（由下一层所需）并不匹配。换言之，DSP单元读取数据时，并未按照先前写入的顺序进行。因此，在读取特征图时，DSP单元的数据局部性较差，且需要进行不必要的大量读操作，从而带来显著开销并延长推理时间。

(2) 硬件无关并行。不同的边缘设备通常配备异构计算资源和内存层次结构。固定的模型划分方案忽视了资源状况，无法适应内存层次结构，和/或不能充分利用计算资源。在模型推理过程中，参数需要频繁换入换出。只有少数数字信号处理器（DSP）计算单元（DSP核/切片 [1]）处于活动状态并承担计算任务，而大多数单元则处于空闲状态，等待依赖数据。这类划分方案会浪费大量计算能力，且无法取得令人满意的性能。

我们研究了现有的框架（例如 TVM [8],TASO [17] 和 PET [26]），但发现它们在上述两个问题上的优化非常有限。这种优化对于基于边缘设备的推理而言并不足够。首先，典型框架采用简单的基于枚举的搜索策略，尝试算子融合/拆分的不同组合，最终确定一种计算图优化方案。这种搜索策略没有考虑边缘设备的资源状况，并且随着算子数量增加，效率会降低。例如，TASO 和 PET 中基于枚举的搜索策略在实际应用中只能处理少于 5 个算子的情况，这限制了它们的应用范围。其次，现有优化框架忽略了推理过程中的数据局部性，导致代价高昂的缓存未命中。

, ,
.

[1]多核数字信号处理器使用“DSP 核”一词，而 FPGA 使用“DSP切片”。在下文中，我们使用“DSP单元”作为通用术语。

受现有优化框架缺陷的启发，我们开发了一种新颖的解决方案 Xenos，以加速边缘设备上的模型推理。与现有框架不同，Xenos 基于以数据流为中心的优化，而非以算子为中心的优化。具体而言，Xenos 能够节省以往工作所忽略的两方面开销。

首先（垂直方向上），Xenos 重构相邻算子之间的推理数据流，并开发算子链接技术来优化计算图。在运行推理模型之前，Xenos 会扫描整个计算图，并修改相邻算子之间的数据流。定制后的数据流信息会写入计算图的元数据中，并馈送给 Xenos 的推理引擎。运行时，推理引擎可以按照后续算子读取的顺序写入中间结果（前一个算子的输出）。因此，推理过程能够实现更好的数据局部性。

其次（水平方向上），Xenos 将硬件信息（DSP 单元和内存层次结构）纳入考量，并开发 DSP 感知的算子拆分技术，以管理跨 DSP 单元的数据流。Xenos 不会像 TASO 和 PET 那样枚举所有可能的算子拆分方案，而是通过启发式方法在 DSP 单元之间划分特征图，以实现高并行度，并拆分算子参数，使其适配专用 L2 内存，从而实现高效的数据访问。因此，Xenos 能够更快地确定高性能方案，将模型部署到边缘设备上。

我们的贡献总结如下。

• **框架**。Xenos 是一个完整的端到端框架，专注于以数据流为中心的优化，而非以算子为中心的优化。从性能角度来看，我们的方法能够对基于边缘设备的模型推理进行更深入的优化（即垂直和水平数据流优化），而这些优化此前的研究均未涉及。从维护和开发角度来看，我们的方法无需引入新算子即可优化计算图，从而在持续开发过程中节省大量编程工作。

•**分布式推理**。我们还将 Xenos 从单节点推理扩展到分布式推理，我们

调用 d-Xenos。d-Xenos 面向单个边缘设备无法高效处理的大规模推理工作负载。d-Xenos 集成了用于参数同步的带宽最优环形全归约算法，并以模型并行方式进行推理，从而有效加速推理计算。

• **评估**。我们在不同平台上进行了对比实验，结果表明，Xenos 分别可将推理时间缩短 21.2%–84.9% 和 17.9%–96.2%。Xenos 的表现也优于 TVM，快 3.22×–17.92×。在分布式推理方面，与单设备基线相比，d-Xenos 实现了 3.68×–3.78× 倍的加速。

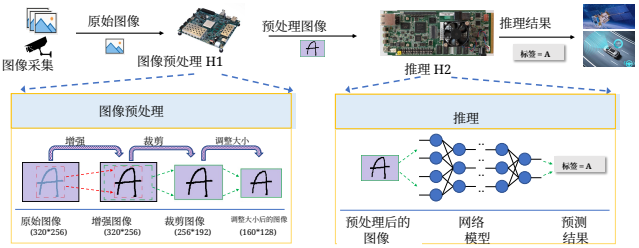


图1。推理工作流的完整堆栈。**2 背景与动机**

2.1 模型推理工作流

图1展示了边缘设备上模型推理的典型工作流程，包含三个主要部分，即图像采集模块、图像预处理模块和推理模块。图像采集模块从图像采集设备或用户输入中收集图像。随后，这些图像将在硬件 H1 上进行预处理，以满足推理模块的要求，包括调整尺寸和图像增强。最后，这些预处理图像将被发送到硬件 H2 上的推理模块，由其输出供应用程序使用的最终推理结果。

工作流必须快速执行，才能满足实时响应能力的要求。在三个模块中，推理模块往往是延迟瓶颈；根据我们的实验，它通常占总执行时间的 60% 以上。因此，推理模块成为提升系统响应能力的关键，Xenos 可用于加速该模块。

2.2 推理计算中的数据局部性

在执行推理计算时，由于不同算子的数据布局不匹配，DSP 单元通常会以非连续的顺序读取数据。我们以深度可分离卷积为例来说明这一点：它由深度卷积和后续的逐点卷积组成。

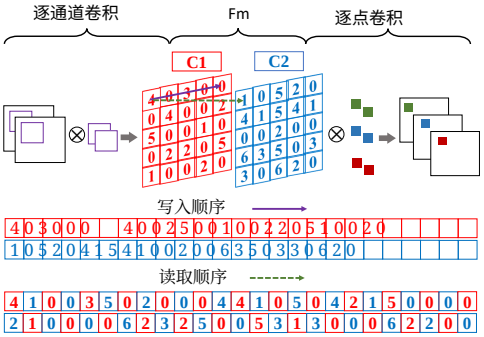


图2. 局部性较差时低效的数据流调度 图2中，特征图 *Fm*有 2 个通道 *C*1和 *C*2，它是深度卷积的输出，也是逐点卷积的输入。深度卷积按宽度优先顺序写入 *Fm* ，而逐点卷积按通道优先顺序读取 *Fm* ，导致数据

这两个算子之间的数据布局不匹配。因此，简单地对模型进行分区会导致推理过程的数据局部性较差，因为每个 DSP 单元在写入和读取特征图时都需要不同的数据块。

2.3 硬件资源异构性

在边缘设备上执行推理任务时，将整个模型置于高层内存（例如 L2内存）中能够提高效率。由于边缘设备上的内存容量有限，这种策略在实践中并不容易实现。尽管现有工作 [13, 15, 23] 已提出了不同的剪枝方法，但模型规模仍然过大，无法适配边缘设备的内存层次结构，从而导致推理任务的性能严重下降。例如，作为专为边缘硬件设计的轻量级推理模型，MobileNet [15] 仍有许多层的特征图和内核大小超过 L2内存的容量（例如 TMS320C6678 上的 512KB）或共享内存的容量（例如 TMS320C6678 上的 4MB ）。直接执行该模型而不进行分区，将导致显著的低效率。

由于边缘设备种类繁多，资源配置各异，且运行的模型大小不同，因此不存在适用于所有场景的“万能”分区方案。人工调优的方案高度依赖专业经验，效率可能很低，还会导致部署周期漫长。现有自动化解决方案，如 TASO [18] 和 PET [26], , 会在庞大的搜索空间中枚举所有可能的分区方案，从而延长部署时间。此外，由于算法复杂度较高，这些框架受计算图规模的限制，因此会错失许多优化机会。因此，我们希望设计一种垂直数据流优化方法，将硬件资源信息纳入考量，并自动生成理想的分区方案，从而充分利用计算能力，同时良好适配内存层次结构。

2.4 现有缺陷与我们的动机

回顾现有框架后，我们发现它们主要采用以算子为中心的优化，即通过将算子替换为融合或拆分后的算子来优化计算图，从而提高计算效率。我们认为，这类优化策略在全局层面并不充分，也无法消除由以下因素导致的性能开销
数据局部性受损且存在硬件异构性。因此，我们转而开发一个新的框架 Xenos，并从数据流视角实现其优化。与现有工作相比，Xenos 具有以下优势。

(1) Xenos 拥有丰富的算子库，且每个算子都支持多种数据流模式。对于未充分优化的计算图，Xenos 可以自动为每个算子选择数据流模式，以提升推理性能。与以往工作采用的算子中心

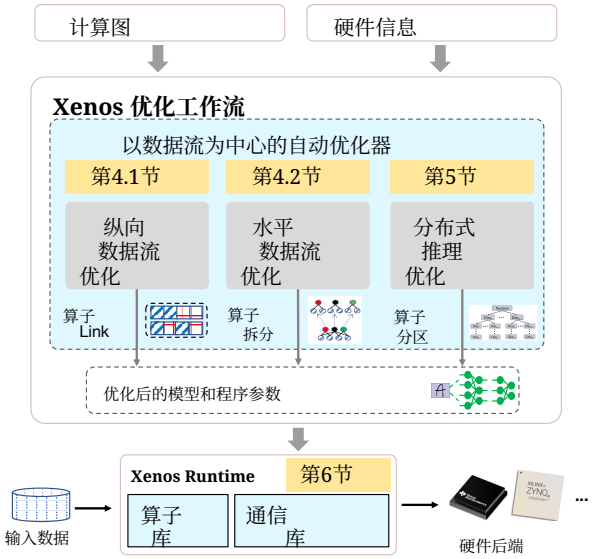


图 3. Xenos 的架构

优化相比，Xenos 以数据流为中心的方法能够进行更深入的优化，并便于持续维护。

（2）Xenos 通过算子链接技术保留数据局部性。在推理计算过程中，Xenos 可以自动推导出算子的最优数据流模式，使得（a）算子输出的中间参数的写入顺序与（b）后续算子的读取顺序相匹配。因此，Xenos 可以避免整个推理过程中代价高昂的缓存未命中。

（3）Xenos 通过感知 DSP 的算子拆分技术，充分利用计算和内存资源。Xenos 将输入张量分区到多个 DSP 单元上，并进一步拆分，使其适配每个 DSP 单元的私有内存。因此，该技术提高了推理并行度，并降低了数据获取开销。

3 架构概述

图 3 展示了 Xenos 的架构。给定一个尚未优化的模型，Xenos 会根据边缘设备的硬件信息执行自动优化流程。随后，Xenos 输出一个与原始模型等效的优化后的模型。运行时，输入数据和优化后的模型都会被送入 Xenos Runtime。Xenos Runtime 利用多个 DSP 单元，基于其高性能算子库和通信库执行推理任务。

与典型框架（TASO 和 PET）一样，Xenos 的优化流程会在预处理阶段进行算子融合，以减少一些低效计算（图 3 中未显示）。然而，这种基础优化不足以实现高性能推理。因此，

Xenos 提供了两项关键技术，即算子链接和 DSP感知算子拆分，分别用于实现垂直和水平数据流优化（§4）。此外，Xenos 还支持分布式推理（§5）：它可以利用多个边缘设备分担工作负载，共同执行一项耗时的推理任务。

4 Xenos设计

本节介绍 Xenos 在单个边缘设备上的详细设计。具体而言，Xenos 通过算子链接技术（§4.1）优化算子间的数据流，并通过 DSP 感知算子拆分（DOS）技术（§4.2）优化跨多个 DSP 单元的数据流。所有这些优化都可以由 Xenos 自动执行（§4.4）。

4.1 垂直：算子连接

推理过程中，Xenos Runtime 在完成一个算子的计算任务后，应按照理想布局将特征图输出到共享内存中，以便在后续算子的推理计算过程中，能够以良好的局部性顺序读取这些数据。否则，数据访问会因缓存未命中而产生严重开销。Xenos 采用算子连接技术来保持数据局部性。我们以图4中的一个链接算子（Conv1x1 + 2×2平均池化）为例说明该技术。

在图4中，输入特征图由前一个算子生成，该算子有4个输出通道，因此输入特征图由四个矩阵组成（以不同颜色标示）。未经数据流优化时，这四个矩阵按行依次存入内存，红色矩阵最先存入，蓝色矩阵最后存入。然而，在推理计算过程中，融合算子每次都需要从每个矩阵读取 1×1 特征图，然后在卷积后对每个 2×2方形区域求平均。如图4右侧的放大图所示，数据流每次都会经过这四个矩阵（1×1卷积计算）。在每个矩阵内，数据流遵循之字形模式（平均池化计算），从而形成重构后的数据流。比较未优化和优化后的数据流可以看出，未优化的数据流在每次数据访问时都会产生强制性缓存未命中。相比之下，优化后的数据流与推理期间的写入/读取顺序完全匹配，因此能够最大限度地

数据局部性更好，数据也能以更高的效率获取。

为保持理想的数据局部性，前一个算子在输出特征图时，需要了解后续算子的数据访问模式。借助这种了解，Xenos 可以根据访问模式修改数据布局，而不是简单地逐个输出矩阵。Xenos 使用算子链接技术来实现这一点，具体细节如下所述。

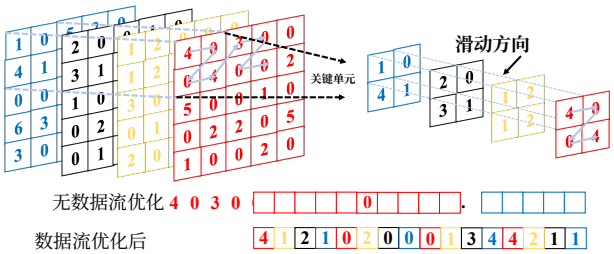


图4。用于优化数据流的算子链接 w

在运行推理模型之前，Xenos 会扫描计算图，并生成用于描述计算图中数据流的元数据。随后，Xenos 分析这些元数据，识别出可能破坏数据局部性的特定模式（即一系列相邻算子）。找到此类低效模式后，Xenos 会修改元数据，以改变这些相邻算子之间的数据流。元数据会被提供给推理引擎。在运行时，推理引擎可以从元数据中获知后续算子的数据访问模式。因此，它能够依据经过优化且保持数据局部性的数据流来写入特征图（见图4）。

值得注意的是，在执行标准卷积计算时，算子链接技术也可能造成数据冗余，因为它会复制特征图的部分参数，以避免后续算子回溯。不过，这种内存上的牺牲是值得的，因为重构带来的性能收益超过了额外的内存开销，并且有效加速了推理工作流（见 §7.2）。

4.2 水平：DSP感知算子拆分

Xenos 采用 DOS 对推理工作负载进行分区，以提高并行度。DOS 侧重于两个方面。首先，它需要将特征图分区到多个 DSP 单元中，使多个 DSP 单元能够分担推理工作负载。其次，它需要将算子参数拆分到边缘设备的内存层次结构中，从而提高参数获取效率。

4.2.1 特征图划分。Xenos 从三个维度划分特征图 [2]，分别是输入特征图高度（inH）、输入特征图宽度（inW）和输出特征图通道（$outC$）。我们不采用基于输入通道（inC）的划分，因为基于 inC 的划分需要进行额外的归约，并会带来更多计算开销。考虑到特征图存储在容量为 4 MB 的共享内存中，远大于典型模型的输入通道规模，因此通常无需沿输入通道进行划分。

Xenos 优先采用基于 $outC$的分区方式，因为其复杂度较低：Xenos 只需将卷积核参数分配给

[2]作为一种特殊情况，当特征图过大而无法存入共享内存时，Xenos 会先将特征图切片，作为预处理步骤，然后由 DOS 继续划分切片后的特征图。

不同的 DSP 单元，并将这些卷积核参数存放在 DSP 单元的 L2内存中。所有 DSP 单元都可以访问位于共享内存中的特征图。另一方面，基于 inH和基于 inW的方案会在基于 $outC$的分区之后，对剩余的特征图进行分区，而且通常需要特殊处理边界行/列。只有在内核无法均匀分配到各个 DSP 单元时，DOS 才会进一步按 inH/inW进行分区。如果经过三重分区后仍存在负载不均衡，DOS 会将剩余的特征图（工作负载）随机分配给不同的 DSP 单元。

虽然算子划分过程将推理工作负载分配到多个数字信号处理器单元上，但单个数字信号处理器单元仍可能无法高效地执行推理任务，因为分配给它的算子参数过大，无法装入 L2内存（例如，CNN模型的内核尺寸非常大）。为解决这一问题，Xenos 需要根据数字信号处理器单元的内存资源，进一步拆分算子参数。下面将介绍算子参数拆分。

4.2.2 算子参数拆分。Xenos 将尺寸较大的算子参数拆分成较小的块，以便将其放入数字信号处理器单元的专用L2内存中。这样可以提高参数获取效率，并缩短推理时间。

Xenos 在拆分参数时，会按照一定优先级处理各个维度，以确保拆分后引入的计算开销最小。以常见的CNN模型为例，其参数包含四个维度，即输出通道（K）、输入通道（C）、卷积核高度（R）和卷积核宽度（S）。沿 K 维度拆分不会引入额外计算，而沿其他三个维度拆分则需要额外的归约操作来汇总这些维度上的结果，从而产生额外的计算开销。因此，Xenos 会优先尝试沿 K 维度拆分参数；如果仅沿 K 维度拆分仍不足以使参数装入 L2内存，则再依次尝试沿 C、R 和 S 维度拆分。

方程 1 给出了基于输出通道的拆分示例。由于较大的参数（W and B）无法放入 L2内存，Xenos 会对算子进行细粒度拆分： W被拆分为 W_1 和 W_2，而 B被拆分为B

$_1$ 和 B_2。之后，可以将参数分布到两个不同 DSP 单元的 L2内存中。方程 y_0 和 y_1可以在两个 DSP 单元上并行联合执行，也可以在一个 DSP 单元上依次执行。随后，输出 $y_1(x_i)$ 和 $y_2(x_i)$ 会自动合并，无需执行任何数据布局转换算子。其他类型也可以采用类似方式拆分工作。

$$
\begin{aligned}
& y(x_i) = Wx_i + B \\
& \Downarrow \\
& y_1(x_i) = W_1x_i + B_1 \\
& y_2(x_i) = W_2x_i + B_2
\end{aligned}
\tag{1}
$$

4.3 示例优化

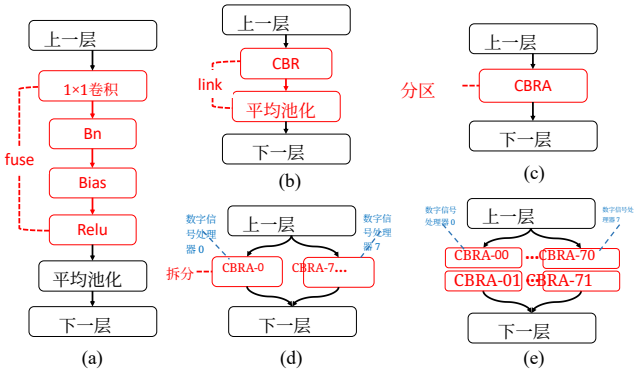


图5。 Xenos优化示意图

图5展示了Xenos如何优化MobileNet [15]计算图中一个部分的示例。Xenos首先对算子Conv1x1（卷积）、Bn（批归一化）、Bias（偏置）和Relu（ReLU）执行典型的算子融合，生成融合算子CBR（图5(a)）；然后将CBR与AvgPooling（平均池化）链接起来，形成算子CBRA（图5(b)）。接下来，Xenos使用DOS技术将特征图分区到各个DSP单元上（图5(d)）。此后，算子参数仍然过大，无法容纳到每个DSP单元的L2内存中，因此Xenos继续将算子参数拆分成更小的数据块（图5(e)），最终完成优化。

4.4 自动优化

Xenos已配备自动化流程，可同时执行垂直优化和水平优化。无需任何人工调优工作，Xenos即可识别计算图中的适当模式（表1），并将优化不足的数据流替换为优化后的版本。自动化为开发人员部署优化后的模型带来了极大便利。我们测量了多个典型推理模型上自动优化的耗时（表2），将优化不足的模型转换为优化后的模型需要0.11毫秒–0.91毫秒。

5 分布式推理

我们预计，推理工作负载很快就会超出单个边缘设备的处理能力。ResNet-101 (60.2M) [12], Bert (340M ~481000M) [9] 和 GPT-3[24](175000M) 等近期模型，几乎无法用于单设备推理，因此分布式推理成为必然。

表 1. 自动模式识别

连接	示例
ConvX ->ConvY	1.卷积 3×3 -> 卷积1×1 2.卷积 5×5 -> 卷积1×1 3.卷积 7×7 ->卷积1×1 4.卷积 3×3 -> 卷积3×3 . . .
ConvX ->ConvY -> Z池化 ConvX ->Z池化 -> ConvY	1.卷积3×3 ->卷积1×1 -> 平均池化 2.卷积3×3 ->卷积1×1 -> 最大池化 3.卷积1×1 ->平均池化 ->卷积3×3 4.卷积1×1 ->最大池化 ->卷积3×3 . . .
ConvX- > { · · · – > ConvY 卷积Z	快捷连接[12]
MatmulX ->MatmulY	MatA * MatB ->MatC * MatD

因此，我们扩展了 Xenos，以利用多个设备进行协同推理计算，并将 Xenos 的分布式版本称为 d-Xenos。

算法1 枚举划分方案

输入： $dset$–待划分的维度集合

$bestShm, bestTime = \varnothing, +\infty$
对 $shm \in$ permutation(dset) **执行**
  $execTime$ =性能分析(shm)
  若 $execTime < bestTime$ **则**
    $bestShm, bestTime = shm, execTime$
返回 $bestShm$

d-Xenos 支持两种方法在多个边缘设备之间同步参数，即基于参数服务器（PS）的同步 [20] 和环形全归约同步 [22]。d-Xenos 的核心思想是结合模型并行计算与多个设备之间的参数同步，从而缩短整体推理时间。不过，我们注意到，与 Xenos 相比，如果 d-Xenos 仍遵循 §4.2.1 中描述的优先级来划分特征图，则可能无法实现最优推理时间。以卷积算子为例，回想一下，卷积的 DOS 优先采用基于 $outC$ 的划分，而不是基于 inH-/inW 的划分。原因在于，基于 $outC$ 的划分可以充分利用不同数字信号处理器单元之间的共享内存，并避免因基于 inH-/inW 的划分而产生的填充开销。然而，对于 d-Xenos，不同的边缘设备之间并不共享内存，因此很难判断基于 $outC$ 的划分是否优于基于 inH-/inW 的划分。为解决这一问题，d-Xenos 使用基于树的遍历算法来确定划分策略。

通用枚举算法如算法 1 所示。d-Xenos 会针对推理过程中涉及的特征图或算子参数中的各个维度，枚举所有可能的分区方案组合；如果有

总共有 d 个维度。例如，矩阵乘法有三个维度（m、 n和 k，其中 k 由特征图和参数共享

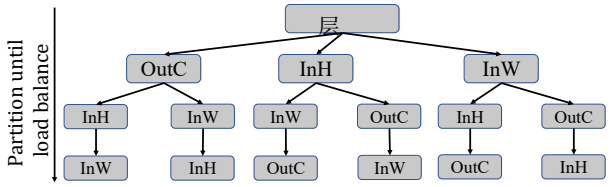


图 6. d-Xenos 中分区方案的枚举

），卷积有七个维度（$batch$、 inC、 inH、 inW、 $outC$、 r和 s，其中inC 由特征图和参数共享）。然而，正如我们在 §4.2.1 中讨论的，Xenos 只沿特定维度进行分区，因此 d-Xenos 也选择这些维度进行分区，因为我们通过经验发现，基于其他维度的分区（例如，卷积算子的 inC维分区）在分布式环境中也无法实现更好的性能。因此，如图 6 所示，d-Xenos 在划分卷积时仍聚焦于沿 inH、 inW和$outC$ 进行分区。随后，d-Xenos 根据这些方案的执行时间性能分析结果，从中选择最佳分区策略。由于在特定平台上部署时，每个模型只需进行一次这样的搜索，因此搜索成本可以接受。

6 实现

要使用 Xenos，用户需要提供推理模型的计算图，以及边缘设备的硬件特定信息，例如内存层次结构、数字信号处理器单元的数量等，以便 Xenos 执行硬件自适应优化。优化后的模型将生成为由 Xenos Runtime 支持的可执行文件，其中包含高效的算子库，以及用于内存管理和可扩展通信的中间件。目前，Xenos 支持多种边缘设备，包括多核 DSP 硬件 [2],赛灵思 U 系列硬件 [4],赛灵思 Zynq 系列硬件 [5],等。

6.1 算子库

现有框架通常提供高级编程语言（例如 Python）的应用程序接口，并向程序员隐藏底层实现细节。然而，在使用边缘硬件平台时，程序员需要使用底层算子和编程语言（例如 C/C++ 和汇编语言）来实现自己的功能。

我们的目标是为普通工作人员提供丰富的算子库，该库效率高，且便于在这些边缘硬件平台上实现他们的高级功能。因此，我们首先使用低级语言（C/C++ 和汇编语言）实现各种类型的算子（例如，具有不同卷积核大小和不同步幅的标准卷积、深度可分离卷积、最大池化/全局池化、拼接等）。与此同时，我们持续

表2。 自动耗时

	MobileNet	SqueezeNet	ShuffleNet	ResNet18	CentreNet	LSTM	Bert-S
耗时（秒）	0.11	0.14	0.36	0.24	0.18	0.64	0.91

表3. 算子与优化

算子	描述
x.add	逐元素加法
x.mul	逐元素乘法
x.mac	乘积累加
x.卷积	卷积（卷积核大小、步幅、填充等）
x.matmul	矩阵乘法
x.gampool	全局池化 / 平均池化 / 最大池化
x.transpose	矩阵转置
x.concat	多个张量的拼接
x.split	将一个张量拆分为多个张量
x.cbr	融合卷积-批归一化-ReLU算子
x.cbrm	链接的CBR-最大池化算子
x.cbra	链接的CBR-平均池化算子

迭代完善算子库，以提升其易用性和兼容性。Xenos 提供的基本算子可概括为表3。如今，不同的机器学习框架往往提供数百种算子。为了支持以算子为中心的优化策略，算子的开发和维护成本可能很高，因为每增加一种优化策略，通常就需要设计更多算子。相比之下，Xenos 的算子支持算子之间的自定义数据流。为实现垂直和水平优化，Xenos 只需自定义算子间数据流，而无需增加更多算子，从而降低了维护成本，也便于在未来开发中扩展。

6.2 可扩展通信中间件

图像数据通常需要在推理前进行一系列预处理操作。通常，预处理模块和推理模块不会在同一设备上执行。因此，我们需要在二者之间搭建桥梁。为实现模块化和可扩展性，我们将通信原语封装为独立的中间件。该中间件兼容用于嵌入式应用的 SRIO 协议技术以及传统的以太网协议。同时，我们的通信中间件集成了流水线和批量传输机制，以实现高吞吐量。此外，为降低实时延迟，我们还定制了高效的打包/解包功能。

7 实验评估

我们旨在通过评估回答以下问题：（1）Xenos 能为模型推理工作流带来多大程度的加速?

（2）Xenos 能为典型算子带来多大的加速?

（3）Xenos 完成自动优化需要多长时间?

（4）使用 Xenos 进行推理时，可以节省多少硬件资源?

（5）与单节点版本相比，分布式 Xenos（d-Xenos）能实现多大幅度的加速?

7.1 实验设置

测试平台：我们采用两个测试平台进行评估：(1) 一台多核数字信号处理器设备，配备 2 个 TMS320C6678 型节点和一个高速图像采集器，两者通过 SRIO 直接连接；(2) 一台 ZCU102 型 FPGA 设备，其代码由英特尔高级综合（HLS）编译器生成。为简洁起见，我们分别称其为 TMS320C6678 和 ZCU102。

基准测试：我们选择 7 种典型模型作为基准测试，即 MobileNet、SqueezeNet、ShuffleNet、ResNet18、CentreNet、LSTM 和 Bert。

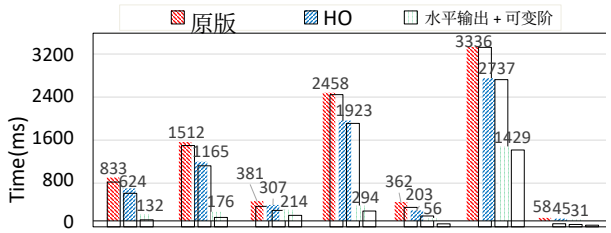
基线：首先，我们针对 Xenos 开展消融研究：将同时采用水平优化（HO）和垂直优化（VO）的完整 Xenos 方案与两个基线进行比较，以展示 Xenos 两种策略的优势。如图 7 所示，其中一个基线既不采用 HO，也不采用 VO，我们称之为原始基线。另一个基线仅采用 HO，我们称之为 HO 基线。随后，我们在 ZCU102 上使用相同的基准测试，将 Xenos 与 TVM [8], 进行比较。此外，我们还将 Xenos 与 GPU 基线进行比较，在该基线上，我们使用 PyTorch 在 NVIDIA RTX 3090 GPU 上运行相同的基准测试。

指标： 我们在评估中研究了推理时间开销和硬件资源开销。对于推理时间开销，我们对每个推理工作负载运行 1000 次，并报告平均值。对于资源开销，我们比较 TMS320C6678 上 L2、SRAM 和 DDR 的开销；并比较 ZCU102 上 DSP 3、触发器 [4] 和查找表 [5] 的开销。

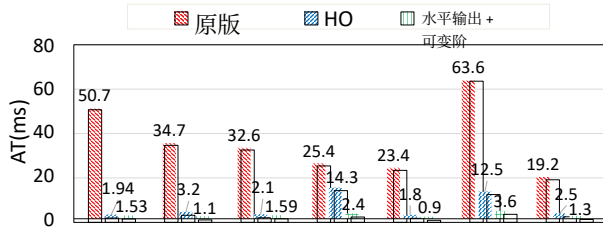
7.2 推理时间比较

TMS320C6678。 在图7(a)中，与原版相比，水平输出可将推理时间缩短17.9%-43.9%，这体现了更高计算并行度带来的加速效果。我们进一步通过比较 HO 基线与完整的 Xenos 方案，评估可变阶带来的性能优势。

[3]数字信号处理器（参见 DSP切片）由高速乘法器电路组成，用于在 FPGA 中执行高速乘加运算。[4]触发器（Flip Flop）是一种只能存储 1 个二进制位的存储单元，可用作时序逻辑电路的存储元件。[5]查找表（Look-up Table）是用于实现组合逻辑电路功能的模块。



(a) TMS320C6678



(b) ZCU102

图 7. 推理时间比较

可以看出，可变阶还能进一步将推理时间缩短30.3%-84.9%，这体现了推理过程中良好数据局部性带来的性能提升。**ZCU102**。 图7(b)也展示了水平输出和可变阶在 ZCU102 上带来的性能优势。同样，与原始基线相比，水平输出可将推理时间缩短80.4%-96.2%。与 HO 基线相比，可变阶还能进一步将推理时间缩短21.2%-83.3%。

对比图7(a)和图7(b)可以很容易发现，水平输出（HO）在 TMS320C6678 上带来的推理时间缩减更多，而可变阶（VO）在 ZCU102 上的贡献更大。原因主要有两个方面。

（1）可变阶（VO）在 TMS320C6678 上比在 ZCU102 上更有效。这是因为 ZCU102 使用大量查找表资源来实现数据映射，因此即使不使用 VO，内存访问效率也已经很高。相比之下，TMS320C6678 未配备此类工具，因此当推理工作负载破坏数据局部性时，内存访问效率会受到严重影响。不过，VO 通过数据流重构帮助保持数据局部性，因此成为主要贡献因素。

(2) 与 TMS320C6678 相比，HO 在 ZCU102 上更有效。这是因为 ZCU102 的 DSP 单元远多于 TMS320C6678。TMS320C6678 只有 8 个 DSP 单元，而 ZCU102 可以分配数千个 DSP 单元参与计算。因此，模型分区和并行执行的管理对于提高 ZCU102 的效率变得更加重要。由于原始基线没有采用合适的分区方案，无法充分利用丰富的计算资源。相比之下，HO 配合优化算子，能够实现对

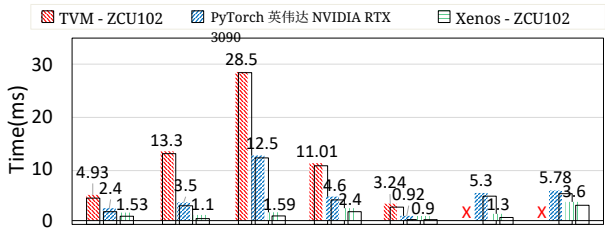


图8. TVM 与 PyTorch 的推理时间比较

DSP计算资源的高利用率，从而显著缩短推理时间。

与其他基线比较。 我们进一步将 Xenos 与另外两个基线进行比较。首先，我们在 ZCU102 上运行 TVM，使用相同的推理模型[3]。其次，我们还报告了 GPU 环境下的推理性能：使用 PyTorch，在 NVIDIA RTX 3090 GPU 上运行相同的模型。图8显示，Xenos 的性能也显著优于另外两个基线。具体而言，与 GPU 基线相比，Xenos 实现了 1.02×–1.87× 倍的加速；与 TVM 基线相比，则实现了 3.22×–17.92× 倍的加速。在所有基准模型中，除长短期记忆网络（LSTM）和 Bert-S 6 外，TVM 在相同硬件上的表现远逊于 Xenos，因为 TVM 未能在推理过程中充分利用硬件信息。

7.3 典型算子的微基准测试

表4. 典型算子的加速比。CBR 是卷积-批归一化-ReLU 的缩写。

算子	Xenos 优化	加速比
CBR-最大池化 $224 \times 224 \times 24$ $3 \times 3 \times 3 \times 224$	算子链接	3.3×
CBR-平均池化 $7 \times 7 \times 1024$ $1 \times 1 \times 1024 \times 1024$	算子链接	2.3×
全连接 $1 \times 1 \times 1536$ $1 \times 1 \times 1536 \times 1000$	算子拆分	2.25×
CBR $112 \times 112 \times 32$ $1 \times 1 \times 32 \times 64$	算子拆分	2.6×

我们选取典型算子，并研究 Xenos 对它们的加速效果。我们在 TMS320C6678 上对卷积神经网络中的典型算子进行了微基准测试，表5展示了 Xenos 带来的加速比。

[6]TVM 无法在边缘硬件上运行长短期记忆网络/Bert-S，因为 TVM 依赖赛灵思提供的开发套件来运行推理模型，但赛灵思的开发套件不支持在 ZCU102 上运行长短期记忆网络/Bert-S

表5. 典型算子的加速比。CBR 是卷积-批归一化-ReLU 的缩写。

算子	优化	加速比
全连接 $1\times1\times1536$ $1\times1\times1536\times1000$	算子拆分	2.25×
CBR $112\times112\times32$ $1\times1\times32\times64$	算子拆分	2.6×
全连接 $1\times1\times1536$ $1\times1\times1536\times1000$	-	1×
CBR $112\times112\times32$ $1\times1\times32\times64$	-	1×

7.4 自动时间开销

我们希望评估 Xenos 完成推理模型自动优化所需的时间。表2列出了优化典型推理模型的耗时。大多数优化可在 0.1–0.2 秒内完成。对于 Bert-S 等复杂模型，Xenos 的自动优化耗时更长，但仍不到 1 秒。

7.5 资源开销比较

7.5.1 TMS320C6678 上的比较。图 9 展示了在 TMS320C6678 上运行 MobileNet 时的资源开销比较。其他 6 个模型呈现出类似趋势，因此受篇幅限制，我们省略了它们。由于存储开销的差异主要由水平输出而非可变阶导致，我们省略了中间基线，只比较原版和完整的 Xenos 方案。

原版的 DDR 开销来自特征图和参数（例如权重）。在推理过程中，特征图大小和参数大小都可能超过 SRAM 的存储容量。如图9(c)所示，初始 ~22毫秒内出现的 DDR 突发主要由输出特征图导致。当时输入特征图占用了 SRAM，因此 SRAM 不足以同时容纳输入特征图和输出特征图。在 ~163毫秒结束时，出现突发是因为 MobileNet 中一个尺寸较大的卷积层，其参数大小超过 4MB，无法存放在 L2内存或基于SRAM的共享存储器中。

7.5.2 ZCU102 上的比较。图10分别展示了 MobileNet 和 SqueezeNet 在 ZCU102 上的资源开销，可以看出，HO 和 VO 都有助于降低 ZCU102 上的资源开销。其余 5 个模型呈现出与 MobileNet 类似的趋势，因此这里省略。

HO 可实现更高的并行度，并提高资源利用效率。一些推理操作的计算能够更快完成，因此相应的资源（例如 DSP单元）可以被释放并重新用于其他推理操作，而无需分配更多资源。与原版相比，HO 可以用更短的时间完成相同的工作负载，同时资源成本更低。

VO 也有助于降低资源成本。由于采用了重构后的数据流，数据访问速度更快，进而减少了资源空闲时间（例如，DSP单元无需长时间等待输入特征图）。因此，资源利用率得以提高，推理引擎也不必向边缘设备申请不必要的额外资源。

然而，我们注意到，在资源成本方面，SqueezeNet 的表现趋势与其他模型有所不同。具体而言，HO 并未降低 DSP 单元的成本（甚至略有增加）。最终我们发现，SqueezeNet 的网络结构很容易并行化，因此能够受益于 HLS 默认提供的优化。换言之，HLS 在代码生成过程中也集成了一些优化机制，这些机制已帮助原版充分利用 DSP 单元，使得 HO 几乎没有进一步优化的空间。因此，HO 并未给 SqueezeNet 带来明显的性能提升。

7.6 分布式 Xenos（d-Xenos）

我们使用 4 台 TMS320C6678 设备评估 d-Xenos。我们为 d-Xenos 配备了环形全归约同步算法，该算法已被证明在参数同步方面具有最优带宽 [22]。我们在 d-Xenos 上运行了三个模型（MobileNet、ResNet 和 Bert）。评估结果如图 11 7 所示。我们总结了两点主要结论：

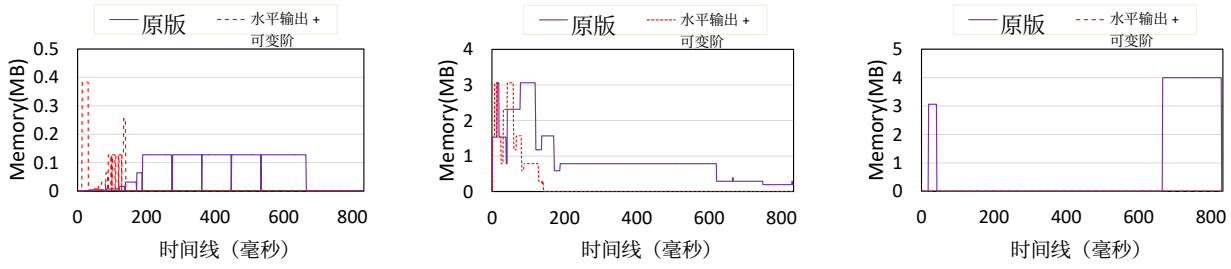
（1）环形全归约同步比传统的基于参数服务器的同步更高效。采用基于参数服务器的同步时，由于参数同步占据了大部分开销，推理时间甚至可能比单设备推理更长。相比之下，采用环形全归约同步时，d-Xenos 实现了显著的加速。

（2）不存在一种“万能”的分区方案，能够实现最优推理时间。因此，单一模式的分区方案（即基于 *inH*、*inW* 和 *outC* 的方案）都无法达到最优性能。相比之下，我们的剖析驱动方法会为相关算子选择最合适的分区方案，因此，混合分区方案（Ring-Mix）能为 d-Xenos 带来最大的性能优势。

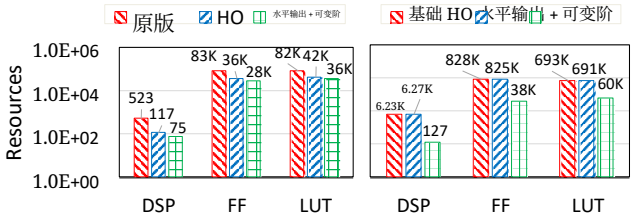
8 讨论：Xenos 与 TASO 和 PET 的比较

TASO [18] 和 PET [26] 被认为是与 Xenos 最相关的工作。熟悉相关领域的读者可能会想知道，如果

[7]由于 Bert-L 和 Bert-S 模型只有一个通道，因此不采用基于 *inC*–/*outC*– 的分区方案。ResNet-101/Bert-L/Bert-S 的单节点推理耗时远长于其他基线，因此我们未将其纳入图11。



(a) MobileNet L2 成本 (b) MobileNet SRAM 成本 (c) MobileNet DDR 成本



(a)MobileNet 资源成本 (b)SqueezeNet 资源成本

图10. ZCU102 上的资源成本比较

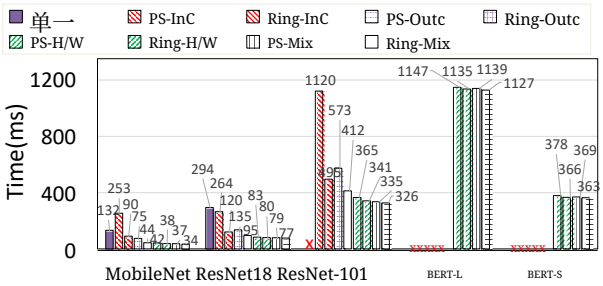


图11. d-Xenos 推理时间

如果配备相同的算子库，*TASO* 和 *PET* 能否搜索出与 Xenos相当（甚至更好）的解决方案？理论上，如果为 TASO 或 PET 配备合适的代价函数，并给予无限的时间和计算能力，其搜索算法（本质上是深度优先搜索，即 DFS）就应该能够输出一种方案，实现最优的内存布局和模型划分。然而，由于两个原因，这种情况并不切实际。首先，尽管声称可以自定义代价函数，但要为内存布局定义代价函数并不容易。TASO 和 PET 只是将执行时间作为代价函数，而且没有针对内存布局的代价函数定义指南，因此无法像 Xenos 的垂直优化那样保留数据局部性。更重要的是，TASO 和 PET 的搜索算法只是基于 DFS 枚举所有候选项，并未考虑硬件平台的任何先验知识。因此，其搜索空间很容易急剧膨胀。即使经过剪枝，基于搜索的优化也只能处理数量很少的算子。具体而言，TASO 的基于搜索的优化最多只能处理 4 个算子，而 PET 在实际应用中最多只能处理 5 个算子，从而限制了它们的应用范围。

源成本比较（MobileNet）

另一方面，Xenos 将先验知识——包括硬件的资源信息和算子的数据信息——作为指导，因此其水平和垂直优化能够避免搜索空间爆炸，并高效地找到近似最优方案。不过，TASO/PET 与 Xenos 的优化方法并不互斥，TASO/PET 的自动搜索算法也可以由 Xenos 继承，以发现更多优化方案。我们将此类优化方法的整合作为未来工作。

9 相关工作

图级优化 现有机器学习框架将神经网络模型表示为计算图，并通过图级优化来优化机器学习任务。TensorFlow [7]、XLA [6]、TensorRT [1]、MetaFlow [19]和 DNNFusion [21] 通过领域专家设计的变换规则来优化计算图。TASO [18]和 PET [26]进一步采用超优化技术，自动执行图级优化，从而显著扩大优化空间，同时减少人力投入。此外，在缺乏硬件知识时，单纯应用上述优化技术无法取得理想的性能。因此，Xenos 采用架构感知方法（DSP感知算子拆分和算子链接）来进行更深入的优化。

代码生成 Halide [25] 提出了一种用于张量程序的领域专用编程语言，并提出了“计算+调度”模型，以解耦各个优化阶段。受 Halide 启发，TVM [8] 采用了类似的优化模型，并提出一种基于学习的方法，自动搜索给定调度的超参数，从而生成高效的代码。Ansor [28] 和 FlexTensor [29] 探索了不同的调度策略，以进一步扩大 TVM 的搜索空间。

边缘硬件上的推理 AOFL并行化[30]通过融合卷积层，并根据计算资源的可用性和网络条件动态选择最佳并行度，从而提升基于边缘的推理效率。相比之下，Xenos 更关注边缘设备上的资源（内存和计算）条件

（HO），并对算子间数据流（VO）进行更深入的优化。Mema[10] 改进了调度策略，使多个推理任务能够在不增加额外边缘资源的情况下运行。虽然 Xenos 目前专注于加速单一推理任务，但我们相信，经过一些调整，其策略也能兼容多任务场景。 [11] 提出了一种协同方案，利用云端资源和边缘设备共同承担一项大型推理任务。Xenos 利用云端资源加速基于边缘的推理将是一个有趣的研究方向，我们将其留作未来工作。

10 结论与未来工作

我们提出了 Xenos，它融入了以数据流为中心的优化策略，以加速基于边缘的推理。我们在两个典型平台——多核 DSP（TMS320C6678）和 FPGA（ZCU102）上，使用 7 个基准测试进行了全面实验。评估结果证明了 Xenos 的垂直和水平数据流优化的有效性，同时也表明，在相同设置下执行基于边缘的推理时，Xenos 的性能优于 TVM。我们还开发了一个初步的分布式版本（d-Xenos），与单一设备相比，其加速比可达到 3.68×–3.78×。

Xenos 的未来发展主要包括两个方面：(1) 我们将继续优化 Xenos 的分布式版本，以提高多个边缘设备之间协同推理的效率。(2) 我们将考虑引入 TASO/PET 的方法，以增强 Xenos 的优化策略。

参考文献

[1] Nvidia TensorRT。https://developer.nvidia.com/tensorrt。[2] TI TMS320C6678。https://www.ti.com/product/TMS320C6678。[3] TVM-Deploy。http://tvm.apache.org/docs/how_to/deploy/vitis_ai。html?highlight=vitis。[4] Xilinx U 系列。https://www.xilinx.com/products/boards-and-kits/ alveo.html。[5] Xilinx ZCU102。https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html。[6] XLA：面向机器学习的优化编译器。https://www. tensorflow.org/xla。[7] Martín Abadi、Paul Barham、Jianmin Chen、Zhifeng Chen、Andy Davis、Jeffrey Dean、Matthieu Devin、Sanjay Ghemawat、Geoffrey Irving、Michael Isard 等。TensorFlow：一个大规模机器学习系统。载于 第 *12* 届 {*USENIX*} 操作系统设计与实现研讨会（{*OSDI*} *16*），第 265–283 页，2016年。[8] Tianqi Chen、Thierry Moreau、Ziheng Jiang、Lianmin Zheng、Eddie Yan、Haichen Shen、Meghan Cowan、Leyuan Wang、Yuwei Hu、Luis Ceze 等。{TVM}：一种用于深度学习的自动化端到端优化编译器。载于 第 *13* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 18*），第 578–594 页，2018年。[9] Jacob Devlin、Ming-Wei Chang、Kenton Lee 和 Kristina Toutanova。Bert：用于语言理解的预训练双向 Transformer。*arXiv* 预印本 *arXiv:1810.04805*，2018年。[10] Jeroen Galjaard、Bart Cox、Amirmasoud Ghiassi、Lydia Y Chen 和 Robert Birke。MEMA：多个深度模型的快速推理。载于 *2021 IEEE* 普适计算与通信国际会议研讨会及其他附属活动（*PerCom Workshops*），第 281–286 页。IEEE，2021年。[11] Ramyad Hadidi、Jiashen Cao、Micheal S Ryoo 和 Hyesoon Kim。物联网设备上的深度神经网络协同执行。*arXiv* 预印本 *arXiv:1901.02537*，2019年。[12] Kaiming He、Xiangyu Zhang、Shaoqing Ren 和 Jian Sun。用于图像识别的深度残差学习。载于 *IEEE* 计算机视觉与模式识别会议论文集，第 770 778 页，2016年 – 。[13] Yihui He、Ji Lin、Zhijian Liu、Hanrui Wang、Li-Jia Li 和 Song Han。AMC：面向移动设备的自动化机器学习模型压缩与加速。载于 欧洲计算机视觉会议（*ECCV*）论文集，第 784–800 页，2018年。[14] Jacob Hochstetler、Rahul Padidela、Qi Chen、Qing Yang 和 Song Fu。面向车载边缘计算的嵌入式深度学习。载于 *2018 IEEE/ACM* 边缘计算研讨会（*SEC*），第 341–343 页。IEEE，2018年。[15] Andrew G Howard、Menglong Zhu、Bo Chen、Dmitry Kalenichenko、Weijun Wang、Tobias Weyand、Marco Andreetto 和 Hartwig Adam。MobileNets：面向移动视觉应用的高效卷积神经网络。*arXiv* 预印本 *arXiv:1704.04861*，2017年。[16] Yutao Huang、Xiaoqiang Ma、Xiaoyi Fan、Jiangchuan Liu 和 Wei Gong。当深度学习遇上边缘计算。载于 *2017 IEEE* 第 *25* 届网络协议国际会议（*ICNP*），第 1–2 页。IEEE，2017年。[17] Zhihao Jia、Oded Padon、James Thomas、Todd Warszawski、Matei Zaharia 和 Alex Aiken。TASO：通过自动生成图替换优化深度学习计算。载于 第 *27*62，2019 47届 *ACM* 操作系统原理研讨会，第 – 页。[18] Zhihao Jia、Oded Padon、James Thomas、Todd Warszawski、Matei Zaharia 和 Alex Aiken。TASO：通过自动生成图替换优化深度学习计算。载于 第 *27*62，2019 47届 *ACM* 操作系统原理研讨会，第 – 页。[19] Zhihao Jia、James Thomas、Todd Warzawski、Mingyu Gao、Matei Za-haria 和 Alex Aiken。通过放宽约束优化深度神经网络计算图替换。载于*2*届系统与机器学习会议论文集，SysML'19，2019年。[20] Mu Li、David G. Andersen、Jun Woo Park、Alexander J. Smola、Amr Ahmed、Vanja Josifovski、James Long、Eugene J. Shekita 和 Bor- Yiing Su。使用参数服务器扩展分布式机器学习。载于第*11*届 *USENIX* 操作系统设计与实现会议论文集，OSDI'14，第583–598页，美国，2014年。USENIX 协会。[21] Wei Niu、Jiexiong Guan、Yanzhi Wang、Gagan Agrawal 和 Bin Ren。Dnnfusion：通过先进的算子融合加速深度神经网络执行。载于第*42*届 *ACM SIGPLAN* 编程语言设计与实现国际会议论文集，第883–898页，2021年。[22] Pitch Patarasuk 和 Xin Yuan。面向工作站集群的带宽最优全归约算法。《并行与分布式计算杂志》，69(2):117–124，2009年。[23] Antonio Polino、Razvan Pascanu 和 Dan Alistarh。通过蒸馏和量化压缩模型。*arXiv* 预印本 *arXiv:1802 05668*，2018年。。[24] Alec Radford、Karthik Narasimhan、Tim Salimans 和 Ilya Sutskever。通过生成式预训练提升语言理解。2018年。[25] Jonathan Ragan-Kelley、Connelly Barnes、Andrew Adams、Sylvain Paris、Frédo Durand 和 Saman Amarasinghe。Halide：一种用于优化图像处理流水线中的并行度、局部性和重新计算的语言与编译器。载于第*34*届 *ACM SIGPLAN* 编程语言设计与实现会议论文集，PLDI '13，2013年。[26] Haojie Wang、Jidong Zhai、Mingyu Gao、Zixuan Ma、Shizhi Tang、Liyan Zheng、Yuanzhi Li、Kaiyuan Rong、Yuanyong Chen 和 Zhihao Jia。{PET}：通过部分等价变换和自动校正优化张量程序。载于第*15*届 *USENIX* 操作系统设计与实现研讨会 *(OSDI 21*54，2021 37*)*，第 – 页。[27] Haoxin Wang 和 Jiang Xie。基于用户偏好的能耗感知移动增强现实系统与边缘计算。载于*IEEE INFOCOM 2020-IEEE* 计算机通信会议，第1379–1388页。IEEE，2020年。[28] Lianmin Zheng、Chengfan Jia、Minmin Sun、Zhao Wu、Cody Hao Yu、Ameer Haj-Ali、Yida Wang、Jun Yang、Danyang Zhuo、Koushik Sen 等。Ansor：为深度学习生成高性能张量程序。载于第*14*届 {*USENIX*} 操作系统设计与实现研讨会 *(*{*OSDI*} *20)*，第863–879页，2020年。[29] Size Zheng、Yun Liang、Shuo Wang、Renze Chen 和 Kaiwen Sheng。FlexTensor：面向异构系统张量计算的自动调度探索与优化框架。载于第二十五届编程语言与操作系统架构支持国际会议论文集，第859 873页，2020 –。[30] Li Zhou、Mohammad Hossein Samavatian、Anys Bacha、Saikat Majum- dar 和 Radu Teodorescu。在异构边缘设备上自适应并行执行深度神经网络。载于第*4*届 *ACM/IEEE* 边缘计算研讨会论文集，第195–208页，2019年。