arXiv:2603.09642v1 [cs.DC] 10 Mar 2026

边缘 SoC 上稀疏模型的多DNN推理

Jiawei Luo, Di Wu, Simon Dobson,BlessonVarghese计算
机科学学院，圣安德鲁斯大学，英国苏格兰
{jl425,dw217,simon.dobson,bv6}@st-andrews.ac.uk

摘要

现代边缘应用日益需要多 DNN 推理系统在异构处理器上执行任务，从而同时从并发执行以及将每个模型匹配到最合适的加速器中获得性能提升。然而，现有系统对每个任务仅支持单一模型（或少量稀疏变体），这阻碍了这种匹配的效率，并导致较高的服务等级目标违约率。我们为多 DNN 推理系统引入模型拼接，通过重组稀疏模型中的子图来创建模型变体，且无需重新训练。我们提出了一个演示系统 SparseLoom，表明模型拼接可以部署到片上系统上。我们通过实验表明，与最先进的多 DNN 推理系统相比，SparseLoom 可将 SLO 违约率降低高达 74%，将吞吐量提升高达 2.31×，并将内存开销平均降低 28%。

关键词

多DNN推理、稀疏模型、异构处理器、模型拼接

1 引言

现代边缘设备应用通常需要并行运行多种任务 [15, 22, 23]。例如，增强现实（AR）应用通过执行多个并发任务来提升用户体验（如图 1 的任务层所示），包括语音识别 [32], 图像分类 [43, 49], 活动识别 [36], 和情感分类 [33]。这些深度学习任务对底层硬件提出了不同的要求。

此外，系统可能受到多个服务级别目标（SLO）的约束——例如，延迟与准确率的优先级可能会随着时间不断变化 [1, 2, 7, 14, 29]。因此，每项任务都可以从多个稀疏变体中进行配置（如图 1 的模型层所示）。这些稀疏变体通过剪枝 [6, 53] 或量化 [13,54]从同一个基础模型派生而来。这些变体共同构成一个支持多样化 SLO要求的<样式 id='7'>稀疏模型库</样式>。例如，增强现实中的图像分类在实时AR标注时可能需要较低的延迟，但在背景场景分析时则可以容忍较高的延迟。部署与这些

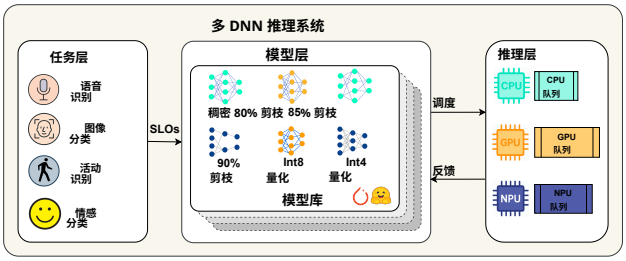


图 1：异构边缘片上系统上的多 DNN 推理增强现实应用场景。该应用在中央处理器、图形处理器和 NPU 上并行执行四项任务。每项任务都配置了一个稀疏模型库，以满足不同的 SLO 要求。

要求相匹配的不同变体，可以提升整体系统利用率和应用性能。

在推理层，这些模型通常部署在边缘片上系统（SoC）上，其中包括中央处理器（CPU）、图形处理器（GPU）和神经处理器（NPU）。为了充分发挥这些处理器的计算能力，模型通常会被划分为子图，并分配给异构处理器并行执行，从而提升整体推理吞吐量 [23, 24, 45]。

多 DNN 推理系统需要在可用设备上选择并调度合适的稀疏变体，并根据 SLO 反馈重新优化选择和放置。**理想情况下，这类系统在服务具有不同 SLO 的多个并发任务时，应尽可能降低 SLO 违规率并提高吞吐量。**

现有的多 DNN 推理系统可根据是否支持稀疏模型选择进行分类。大多数现有研究假设每个任务只有一个基础模型，无法从稀疏模型库中自适应地选择变体 [14, 15, 50]。这些系统无法利用多样的准确率与延迟的权衡来满足不同的 SLO 约束。近期研究基于强化学习等技术 [12, 44, 47] 引入模型选择器，以支持自适应变体选择。然而，我们的评估表明，仅从原始稀疏变体中进行选择，往往会导致较高的 SLO 违约率，这是由于可用变体数量有限（见第 2.1 节）。

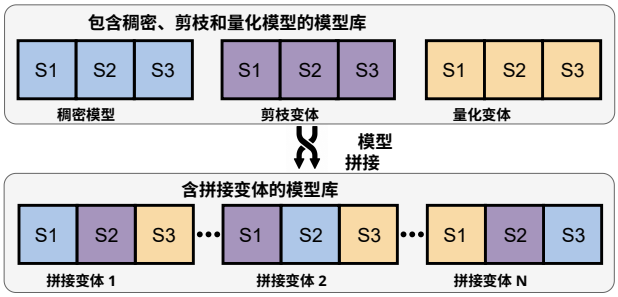


图 2：模型拼接：给定三个稀疏模型（稠密、剪枝、量化），每个模型都拆分为子图 S1–S3。拼接变体 1 将稠密模型（蓝色）中的 S1、剪枝变体（紫色）中的 S2，以及量化变体（橙色）中的 S3 组合在一起。

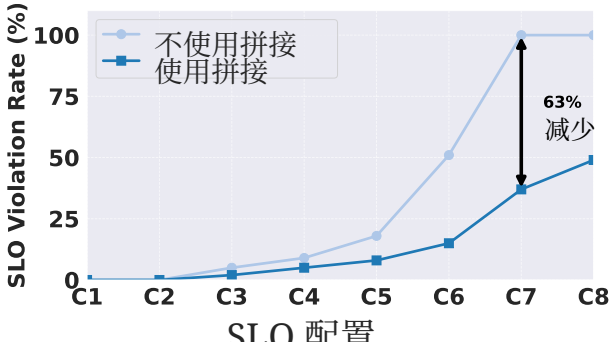


图 3：使用和不使用拼接时的 SLO 违规。拼接可显著降低 SLO 违规率。x 轴表示不同的 SLO 配置，索引越大（例如 C8），对应的 SLO 越具挑战性，其准确率和延迟要求也越严格。

为降低多 DNN 推理中的 SLO 违约率，我们引入了**模型拼接**。以往的方法通过分析单个层 [4] 并创建变体来实现拼接，但代价是需要重新训练 [37]。相比之下，所提出的模型拼接方法通过组合不同稀疏模型中的子图（即，连续的层块）来构建拼接变体，且无需重新训练。这些具有不同稀疏度级别和模式的异构变体提供了更多满足任务 SLO 的选择。因此，**模型拼接无需重新训练即可显著降低 SLO 违约率**（见第 2.1 节）。

在多 DNN 推理系统中集成模型拼接面临三项挑战：(1) 由于可能的拼接变体数量呈指数级增长，性能分析成本高昂；(2) 处理器放置不理想会导致吞吐量下降；(3) 预加载拼接变体会带来显著的内存开销。我们提出 SparseLoom，这是一种面向边缘 SoC 的整体式多 DNN 推理系统，可让模型拼接以更低的性能分析成本、更高的吞吐量和更低的内存开销投入使用。性能分析器模块采用准确率和延迟估算器来预测拼接变体的性能，从而降低性能分析成本。随后，稀疏性感知优化器 模块选择最优处理器放置顺序，并确定拼接变体。最后，热子图预加载器 模块根据给定的全局内存预算，预加载“热度”评分较高的子图，从而提高内存利用率。对多个目标系统、模型和工作负载进行的广泛评估表明，与最先进的多 DNN 推理系统相比，SparseLoom 将 SLO 违约率最多降低 74%，将吞吐量最多提升至 2.31× 倍，并使内存开销平均降低 28%。

2 动机与挑战

本节首先通过实证评估现有多 DNN 推理系统在采用和未采用模型拼接时的性能，以突出模型拼接在减少 SLO 违规方面的优势（第2.1节）。随后，我们指出将模型拼接融入现有系统时面临的三项关键挑战（第2.2节），这些挑战促使我们设计 SparseLoom。

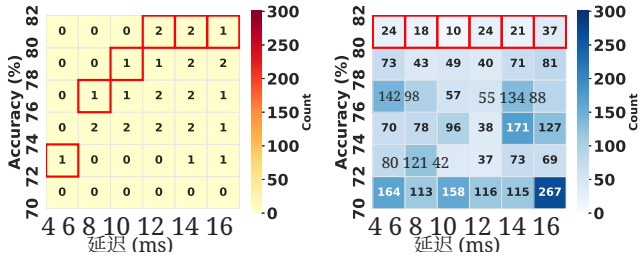
实验设置： 当前多 DNN 推理系统的性能在四个具有代表性的模型上进行了评估：ResNet101 [16], BERT-Base [9], ViT-Small [10], 和 Wav2vec2 [3],，分别对应图像分类、情感分类、活动识别和语音识别任务。对于每项任务，我们构建了一个由十种稀疏性变体组成的稀疏模型库，具体详见附录 A。所示结果均基于 Intel Ultra 7 265K SoC，其集成了中央处理器、图形处理器和 NPU [1]。

2.1 模型拼接的定义、范围与优势

现有的多 DNN 推理系统通常直接从稀疏模型库中选择模型变体。这些变体包括原始稠密模型及其剪枝和量化版本。相比之下，SparseLoom 采用模型拼接来生成新的变体。

定义：模型拼接是一种免训练的变体生成技术，通过组合来自同一基础模型的稀疏变体中层对齐的子图，创建新的模型变体。图 2 展示了模型拼接，其中来自原始模型库中模型的不同连续层块的子图（*S1–S3*）被组合起来，形成新的拼接变体。

[1]这些观察结果可推广至其他硬件平台（见第 5.1 节）。



(a) 未进行拼接。 **(b) 进行拼接。**

图 4：ResNet101 变体在准确率–延迟空间中的直方图；单元格计数表示密度，红色边框单元格标示帕累托前沿。模型拼接扩展了变体空间，并实现了优于原始稀疏模型的准确率–延迟前沿。

适用范围： (i) 稀疏变体通过压缩技术（即量化和剪枝）源自同一基础模型，并且 (ii) 稀疏变体采用相同的子图划分方案，以确保子图能够按照对齐的层位置重新组合（即层对齐子图）。

支持的压缩技术及其依据：非结构化剪枝（零值屏蔽）、结构化剪枝（改变架构）和量化等压缩技术之所以适用于模型拼接，是因为它们能够保留稀疏变体间逐层的输出分布。这一点已在模型压缩文献中得到充分研究，例如相似性分析理论（如 SVCCA [40]）对此进行了证明。此外，许多压缩方法在减少层参数时，会明确约束层输出分布保持相似，例如 BRECQ [27] 和 AMC [17]。因此，从由同一基础模型压缩而来的稀疏变体中拼接子图时，能够保持较高的准确率。

优势：模型拼接能够构建新的变体，从而以无训练方式增加可供选择的变体数量。然而，与直接从原始稀疏模型中进行选择相比，模型拼接能否降低 *SLO* 违规率，以及它如何改善准确率–延迟权衡，目前仍缺乏充分研究。

图 3 展示了在多个任务中采用和未采用模型拼接时的平均 SLO 违约率。具体而言，我们对基础模型和所有稀疏变体进行了基准测试，以测量其准确率和延迟。随后，我们将延迟范围扩大 ±20%，将准确率范围扩大 ±2%，并从扩展后的空间中均匀采样出八种 SLO 配置（C1–C8）。随着 SLO 变得更加严格（从 C1 到 C8），未采用模型拼接时的违约率不断上升，在最严格的配置（C8）下达到 100%。然而，在更严格的 SLO 下，模型拼接最多可使违约率降低 63%，其中 C7 配置下的改善幅度最大。

图 4 对比了准确率–延迟空间中原始稀疏变体（稠密、剪枝和量化）与拼接变体的分布；这些变体均源自 ResNet101。模型拼接不仅增加了可用变体的数量，还实现了更优的帕累托前沿（以红色边框单元格表示），从而为选择提供了更好的准确率–延迟权衡。此外，模型拼接甚至能够生成准确率更高或延迟更低的变体：4% 的拼接变体准确率超过了原始模型中的最高值，5% 的拼接变体延迟低于原始稀疏模型库中速度最快的模型。

2.2 模型拼接的挑战

本节将讨论在将模型拼接纳入多 DNN 推理系统时需要克服的挑战。

> **挑战 1：** 模型拼接引入的大量变体，对如何高效分析其准确率和延迟构成了严峻的可扩展性挑战。

所有变体（包括原始稀疏模型库中的模型和拼接变体）都要进行性能分析，以记录准确率和延迟等指标。然而，由于拼接变体数量庞大，性能分析成本随之增加。

表1展示了采用和不采用模型拼接时所需的性能分析运行总次数。一次性能分析运行是指对一个变体的准确率和延迟进行分析。处理器放置顺序是指子图变体在各处理器上执行的顺序。每个变体都需要在所有处理器放置顺序下进行性能分析。关于性能分析运行次数随任务数 T和变体数 V的增长情况：采用模型拼接时，性能分析运行次数会迅速增长——随任务数 T线性增长，并随每个任务的变体数 V呈指数增长。这凸显了模型拼接带来的可扩展性挑战，也说明了采用更高效性能分析策略的必要性。

对所有变体进行性能分析后，推理系统会根据SLO要求为每个任务选择合适的变体。具体而言，系统会选择能够同时满足准确率和延迟约束的变体。为满足延迟约束，还需要考虑每个处理器上子图的放置顺序，因为每个子图的执行时间高度依赖于处理器类型。

> **挑战2：** 执行具有不同稀疏类型的拼接变体子图时，处理器放置顺序往往无法优化延迟，导致总体吞吐量较低。

表1：有无模型拼接时的剖析复杂度。符号说明：T：任务数；V：每个任务的变体数；S：每个变体的子图数；P：处理器数。

方法	在没有拼接	With 拼接
处理器放置顺序	$P!$	$P!$
变体总数	$T \cdot V$	$T \cdot V^S$
准确率分析运行	$T \cdot V$	$T \cdot V^S$
延迟分析运行	$T \cdot V \cdot P!$	$T \cdot V^S \cdot P!$
总分析运行	$T \cdot V \cdot (P! + 1)$	$T \cdot V^S \cdot (P! + 1)$

处理器放置顺序： 当多个任务的拼接变体在异构处理器上并行执行时，其子图通常按照固定顺序分配给处理器 [23, 45]。例如，对于包含三个子图的变体，系统可能始终将第一个子图分配给 NPU，第二个分配给图形处理器，第三个分配给中央处理器。现有的多 DNN 推理系统采用这种固定放置策略，以简化放置，并减少运行时重新调度的开销 [45]。

表 2 展示了采用不同处理器放置顺序执行一组选定的 ResNet101 拼接变体时的延迟。对于三个处理器（中央处理器、图形处理器和 NPU），共有六种可能的放置顺序[2]。结果表明，最优放置顺序会因不同的稀疏变体而异。更重要的是，广泛采用的 NPU–GPU–CPU（N-G-C）放置顺序 [23, 45]始终无法达到最优延迟，因而降低了总体吞吐量（参见第 5.3 节）。

最优处理器放置顺序： 确定最优处理器放置顺序颇具挑战，因为 (i) 它取决于为每个任务选择的拼接变体；(ii) 不同任务可能需要不同的放置顺序，因为所选变体的类型各不相同。因此，必须结合所有任务的最终变体选择来确定处理器放置顺序。这就需要对处理器放置顺序和变体选择进行自动化联合优化。

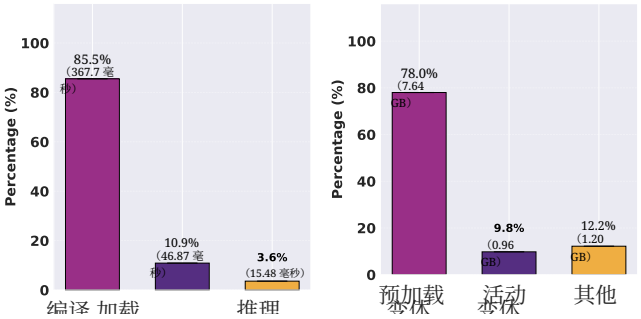
> **挑战3：** 为避免运行时切换延迟，预加载所有拼接变体会带来显著的内存开销。

切换变体的延迟开销： 在执行调度任务期间监控是否违反服务级别目标。如果检测到违规，系统会切换到模型库中的另一个变体，以满足服务级别目标。这包括编译新变体并将其加载到
目标处理器的内存。图5a比较了延迟

[2]仅采用不重叠的放置顺序，以最大限度提高并行性。

表 2：采用不同处理器放置顺序的拼接的ResNet101变体的延迟。最低延迟以蓝色突出显示。P：剪枝，Q：量化，D：稠密；C：中央处理器，G：图形处理器，N：NPU。

变体 / 顺序	P-Q-P	P-P-Q	D-D-P	D-P-Q	Q-P-D	P-D-Q
N-G-C	12.05	16.91	14.77	17.73	18.25	16.99
C-G-N	11.01	13.40	14.45	15.56	20.27	13.48
G-C-N	13.20	13.69	13.51	12.14	12.17	15.54
G-N-C	12.98	14.22	13.49	14.57	13.63	16.51
N-C-G	15.72	11.93	17.39	12.01	13.79	15.73
C-N-G	13.72	10.77	15.40	12.88	18.21	12.51
最佳顺序	C-G-N	C-N-G	G-N-C	N-C-G	G-C-N	C-N-G



(a) 延迟开销。 **(b) 内存开销。**

图 5：(a) 延迟分解，包括编译、加载和推理。(b) 内存分解，包括活动变体、预加载变体和其他。

添加新变体时编译、加载和推理的耗时。加载开销很大，最高占总时间的 96.4%。具体而言，编译和加载分别比推理耗时 23.7× 和 3× 倍。

预加载变体： 为减少运行时重新调度期间的切换开销，现有系统通常会预先编译所有变体，并从磁盘预加载到目标内存中 [12, 26]。虽然这种方法可以隐藏运行时的切换延迟，但对内存的需求很大。图 5b 展示了运行时内存使用量；预加载变体需要大量内存。因此，需要高效地预加载变体以实现切换。

简而言之，模型拼接通过提供更丰富的变体空间，减少了 SLO 违规，但也带来了必须解决的非平凡挑战，才能在多DNN推理系统上实现高效部署。

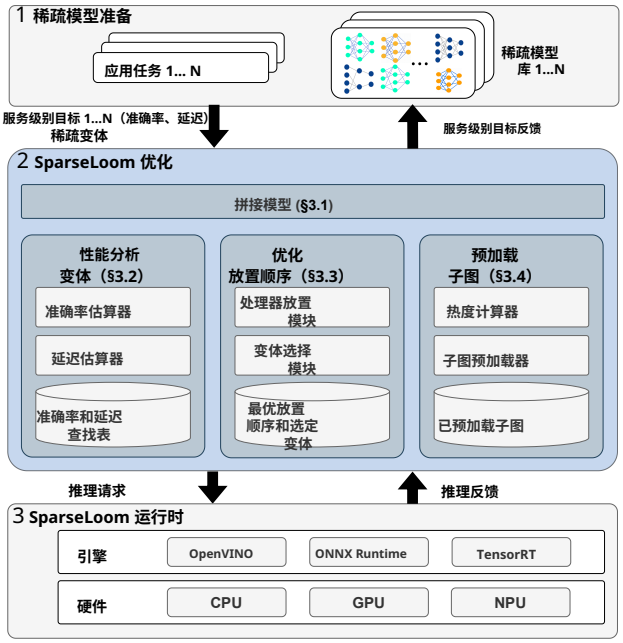


图 6：SparseLoom 中多DNN推理的三个阶段。

3 SparseLoom 的设计

SparseLoom 引入了模型拼接这一核心技术，并通过三个创新模块协同应对分析成本高、处理器放置欠佳以及子图预加载占用内存过多等挑战。

图6展示了SparseLoom中多DNN推理的三个阶段。① **稀疏模型准备：**应用中的每个任务都配备了特定于任务的SLO，以及通过将基础模型压缩为不同稀疏变体而生成的稀疏模型库。② **SparseLoom优化：**SparseLoom首先拼接每个任务的稀疏模型中的变体（第3.1节）。随后，它使用准确率和延迟估算器对拼接变体进行剖析，生成精度–延迟查找表（第3.2节）。接下来，SparseLoom确定最优处理器放置顺序，并为每个任务选择最优拼接变体（第3.3节）。为降低预加载的内存开销，子图预加载器根据热度度量加载子图，该度量同时刻画子图的频率与唯一性（第3.4节）。③ **SparseLoom 运行时：**在SparseLoom确定最优处理器放置顺序并为每个任务选择拼接变体之后，它将推理请求分发到多个处理器上的推理引擎，并收集反馈。

3.1 从稀疏变体拼接子图

这 T 个任务中的每一个都关联 V 个稀疏变体，其中每个变体 $v^{t,i}$ 由 S 个子图$\{s^{t,i}_j\}^S_{j=1}$组成。SparseLoom以无训练方式，通过组合这些原始变体中的子图来构建拼接变体。每个拼接变体 $\tilde{v}^{t,k}$ 同样由 S 个子图$\{\tilde{s}^{t,k}_j\}^S_{j=1}$组成，其中每个 $\tilde{s}^{t,k}_j$ 继承自某个 $s^{t,i}_j$。这在原始变体与拼接变体之间形成了自然映射，因为拼接变体直接复用原始变体中的子图。该映射的形式化定义如下：

$$\tilde{s}^{t,k}_j = s^{t,M[j,i]}_j, \quad \text{where } k = M[j,i]. \tag{1}$$

$M[j,i]$ 表示映射函数，并指定在拼接变体 k中，位置 j 处的子图取自变体 i 中相同位置的子图。SparseLoom 使用的符号说明见附录 B。

3.2 估算拼接变体的准确率和延迟

准确率和延迟估算器：准备好拼接变体后，SparseLoom 会测量每个 $\tilde{v}^{t,k}$的准确率 $A(\tilde{v}^{t,k})$和延迟$Lat(\tilde{v}^{t,k},\vec{p})$。然而，由于变体数量庞大（第2.2节中的挑战 1），通过穷举分析直接测量这些指标并不切实际。为解决这一问题，SparseLoom 估算准确率和延迟，以加快性能分析。

准确率估算器利用与拼接变体 $A(\tilde{v}^{t,k})$共享子图的原始变体的准确率，预测该拼接变体的准确率。其基础方法源于以下观察：子图级性能可以迁移——高准确率变体中的子图更有可能生成准确率较高的拼接变体。具体而言，我们首先评估每个原始变体 $v^{t,i}$的准确率 $A(v^{t,i})$，并将该准确率赋给其所有组成子图，如下所示：

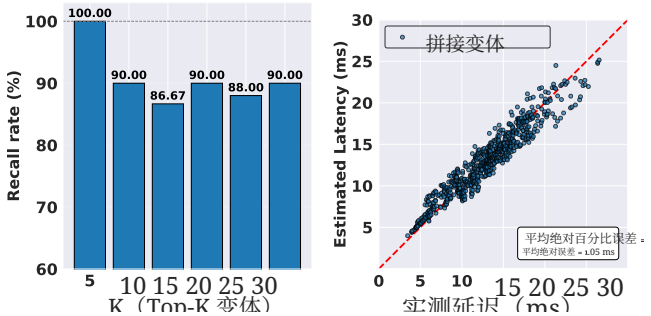
$$X(s^{t,i}_j) = A(v^{t,i}), j \in [1,S] \tag{2}$$

为预测 $A(\tilde{v}^{t,k})$，我们将公式 1 中的子图映射代入公式 2，得到：

$$A(\tilde{v}^{t,k}) = A(\{s^{t,M[j,i]}_j\}^S_{j=1}) \tag{3}$$

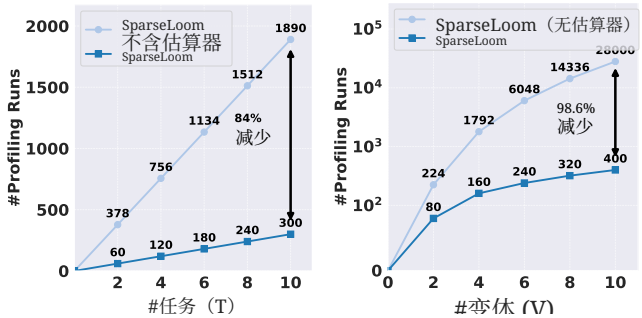
基于公式 2 和公式 3，我们构建一个预测函数 $f(\cdot)$，根据特征 $X(\{s^{t,M[j,i]}_j\}^S_{j=1})$ 估算准确率 $A(\tilde{v}^{t,k})$。我们将此估算器表述为一个监督回归任务，其目标如下：

$$\min_\theta \sum^N \left(A(\tilde{v}^{t,k}) - f_\theta(X(\{s^{t,M[j,i]}_j\}^S_{j=1}))\right)^2 \tag{4}$$



(a) 准确率估算器 (b) 延迟估算器

图 7：所提估算器的评估。(a) 准确率估算器：在准确率最高的变体上的 Top-K召回率。(b) 延迟估算器与真实值的对比。



（a）改变 T（P=3，S=3，V=3） （b）改变 V（P=3，S=3，V=4）

图8：在改变 T 和 V 时，使用和不使用估算器进行 SparseLoom 分析运行的情况。

其中，θ 表示 $f(\cdot)$ 的参数，N 表示训练样本的数量。我们使用基于一小组经过性能分析的拼接变体训练的 XGBoost 来实现 f_θ（。）。

对于延迟估算，我们通过将拼接变体 $\tilde{v}^{t,k}$ 中各子图在相应处理器上的延迟相加，来近似其端到端延迟，其中处理器放置顺序为 $\vec{p}$：

$$Lat(\tilde{v}^{t,k}, \vec{p}) \approx \sum_{j=1}^{S} Lat(s_j^{t,M[j,i]}, p_j) \tag{5}$$

其中，延迟 $(s^{t,M[j,i]}_j, p_j)$ 表示子图 $s^{t,M[j,i]}_j$ 在处理器 p_j 上的实测延迟。未考虑子图间通信开销，因为实证证据表明，在共享内存片上系统上，由于处理器访问统一内存空间，这部分开销相对较低（可忽略不计）。

.....有多准确？ 性能分析器？图7a展示了 SparseLoom 中准确率估算器的评估结果
我们报告使用准确率估算器识别具有 Top-K准确率的拼接变体时的 Top-K召回率。结果表明，SparseLoom 平均能够检索出真实 Top-K 变体的 90.78%。图7b展示了估算延迟与真实延迟的对比。平均绝对误差（MAE）为 1.05 毫秒，平均绝对百分比误差（MAPE）为 8.9%。实验结果证实，所提出的估算器具有较高的 Top-K召回率和较低的延迟误差。

分析成本分析： 我们进一步分析 SparseLoom 的分析成本。公式6给出了总分析成本，它由两部分组成：对每个任务的全部V 个变体进行准确率估计，以及对 P 个处理器上的每一个 S子图进行延迟估算。

$$C_{\text{SparseLoom}} = \underbrace{T \cdot V}_{\text{Accuracy estimation cost}} + \underbrace{T \cdot S \cdot V \cdot P}_{\text{Latency estimation cost}} \tag{6}$$

图8展示了 SparseLoom 的分析成本，以证明其可扩展性。当增加任务数量（T）时，与穷举分析相比，SparseLoom 可将总分析成本最多降低 84%。更重要的是，SparseLoom 随每个任务的变体数（V）线性扩展，成本减少最高可达 98%。

3.3 优化处理器放置顺序并选择拼接变体

基于拼接变体的性能分析结果，SparseLoom 随后确定一个全局处理器顺序 $\vec{p} = \{p_1、\ldots\ldots、p_S\}$，供所有拼接变体共用，以便在每个位置 j 分配子图$s^{t,i}_j$。采用全局处理器顺序可减少流水线气泡和运行时重新调度的开销 [45]。然而，现有多DNN系统未能优化处理器执行顺序，导致延迟未达到最优水平（第2.2节中的挑战2）。

解决方案概要： SparseLoom 联合优化处理器放置顺序和最终变体选择。我们首先对所有满足SLO要求的拼接变体进行性能分析，这些变体涵盖所有可能的处理器放置顺序；然后，通过选择所有变体和任务的平均延迟最低的顺序，确定最佳处理器顺序。基于这一最优顺序，我们再为每个任务选择该顺序下延迟最低的变体，确定最终变体。

算法1概述了两个步骤。优化器首先为每个任务选择同时满足准确率和延迟SLO的拼接变体。任务 t的经过筛选的拼接变体集合记为 Θ^t，并将其作为后续优化的输入。接下来，优化器遍历每个候选处理器顺序 $\vec{p} \in \Omega$。对于每个任务 t，它选择针对 $\vec{p}$延迟最低的拼接变体 $\tilde{v}^{t,k} \in \Theta^t$。随后计算所有任务的平均延迟 $L(\vec{p})$，并选择使 $L(\vec{p})$最小的处理器顺序作为最优放置顺序$\vec{p}^*$。最后，给定所选的全局放置顺序 $\vec{p}^*$，

算法1： 优化处理器放置顺序
并选择拼接变体

输入： $\{SLO^t_{acc}, SLO^t_{lat}$ $^{A(\tilde{v}^{t,k}),\ 纬度(\tilde{v}^{t,k},\ \vec{p})}\}_{t=1}^{T}, \Omega$
输出： $\vec{p}^*, \{\tilde{v}^{t,*}\}_{t=1}^{T}$

// 稀疏性感知处理器放置
对于 $t=1$ 至 T **执行**
  $\Theta^t = \{\tilde{v}^{t,k} \mid A(\tilde{v}^{t,k}) \geq SLO^t_{\text{acc}},\ 延迟(\tilde{v}^{t,k},\ \vec{p}) \leq SLO^t_{\text{lat}};\ \exists \vec{p} \in \Omega\}$
结束
查找使 $\vec{p}^* \in \Omega\ (\vec{p})$ 取得最小值的 L;
  $L(\vec{p}) = \frac{1}{T}\sum_{t=1}^{T} \min\{Lat(\tilde{v}^{t,k},\ \vec{p})\}_{\tilde{v}^{t,k} \in \Theta^t}$
// 最终变体选择
对 $t=1$ 至 T **执行**
  $\tilde{v}^{t,*} = 参数最小化_{\tilde{v}^{t,k} \in \Theta^t} Lat(\tilde{v}^{t,k} | \vec{p}^*)$
结束
返回 $\vec{p}^*$ 和 $\{\tilde{v}^{t,*}\}_{t=1}^{T}$

优化器通过为每个任务选择针对 $\vec{p}^*$执行延迟最低的拼接变体 $\tilde{v}^{t,k} \in \Theta^t$，来确定最终变体。

SparseLoom 对处理器放置和变体选择进行联合优化，使其能够确定一个考虑子图不同稀疏性的最优处理器顺序。因此，优化器可最小化各任务的平均执行延迟并提高吞吐量。

图9展示了在第三个位置执行的所有子图的热度分数（即 $\{s^k_{j=3}\}_{k=1}^{V^S}$）。排名前四的子图具有更高的分数，反映出它们要么出现频率较高，要么具有显著的独特性。这表明，在内存受限时，预加载排名前四的子图可能满足大多数 SLO 配置。

总之，SparseLoom 引入模型拼接以扩展变体空间，并集成三个关键模块，以应对性能分析成本、处理器放置和子图预加载方面的挑战。这些组件共同使 SparseLoom 成为一种适用于异构边缘SoC上多DNN推理的高效解决方案。

3.4 变体的预加载子图

任务的 SLO要求发生变化时，可能需要在运行时切换变体，因为原变体可能不再满足更新后的约束。为降低变体切换造成的延迟，必须进行预加载。然而，预加载所有变体的全部子图会大量占用内存（第2.2节中的挑战3）。为解决这一问题，SparseLoom 根据热度度量预加载子图。子图的热度

算法 2： 热子图预加载

输入： $\{\Theta^t(\sigma)\}_{t\in T,\sigma\in\Psi}, Mem_{\text{budget}}; Mem(s^{t,i}_j)$
输出： $\{\Phi^t\}_{t=1}^{T}$
// 计算热度分数
初始化 $H[s^{t,i}_j] \leftarrow 0$ 对于所有 t,i,j;
对于 $t=1$ 至 T，**执行**
  对于 $j=1$ 至 S，**执行**
    对于 $i=1$ 至 V，**执行**
      $H[s^{t,i}_j] \leftarrow \sum_{\sigma\in\Psi} \frac{出现(s^{t,i}_j, \Theta^t(\sigma))}{|\Theta^t(\sigma)|}$;
    end
  end
结束
// 内存预算下的贪心预加载
初始化 $\Phi^t \leftarrow \emptyset$ 对于所有 t; $M \leftarrow 0$;
对于 $t=1$ 到 T**执行**
  对于 $j=1$ 到 S**执行**
    将 $\{s^{t,i}_j\}_{i=1}^{V}$ 按 $H[s^{t,i}_j]$ 降序排序;
    对于每个 $s^{t,i}_j \in \{s^{t,i}_j\}_{i=1}^{V}$ **do**
      如果 $s^{t,i}_j \notin \Phi^t$ ***and*** $M + Mem(s^{t,i}_j) \leq Mem_{budget}$ **然后**
        $\Phi^t \leftarrow \Phi^t \cup \{s^{t,i}_j\}$;
        $M \mathrel{+}= Mem(s^{t,i}_j)$;
        中断;
      end
    end
  end
结束
返回 $\{\Phi^t\}_{t=1}^{T}$

$s^{t,i}_j$ is defined as

$$\text{Hotness}\left(s^{t,i}_j\right) = \sum_{\sigma\in\Psi} \frac{\text{Occur}\left(s^{t,i}_j, \Theta^t(\sigma)\right)}{|\Theta^t(\sigma)|}, \tag{7}$$

其中，$\Theta^t(\sigma)$ 表示满足任务 t 的配置 σ 的准确率和延迟要求的拼接变体集合，而 Ψ 表示满足$\sigma \in \Psi$的所有 SLO 配置的集合。此处，$\text{Occur}(s^{t,i}_j, \Theta^t(\sigma))$ 表示 $\Theta^t(\sigma)$ 中包含子图 $s^{t,i}_j$ 的变体数量，$|\Theta^t(\sigma)|$ 表示 $\Theta^t(\sigma)$ 中的变体总数。

热度分数同时反映了在满足不同 SLO 的各变体中，某一子图的频率与独特性。一个子图在以下任一情况下会获得较高的热度分数：*(i)* 它在各变体中被广泛复用（高频率），或 *(ii)* 它是唯一（或少数几个之一）能够满足某一 *SLO* 配置的子图（独特）。

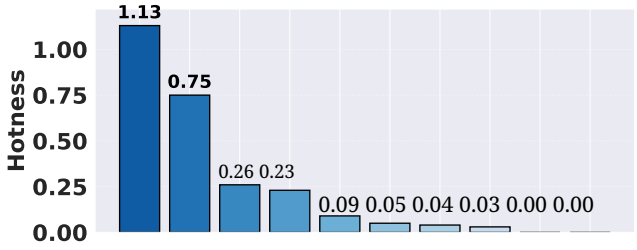


图9：第三个位置上所有子图的热度分数，其中前四个子图占主导地位。

算法 2 给出了 热子图预加载器 的底层步骤。首先计算每个子图 $s^{t,i}_j$ 的热度分数 $H[s^{t,i}_j]$。然后，针对每个任务 t，在全局内存预算 Mem_{budget}下构建预加载子图集 Φ^t。在每个子图位置 j，候选子图按 $H[s^{t,i}_j]$ 降序排序，预加载器在累计内存消耗不超过预算的前提下，贪心地将它们加载到 Φ^t 中。该策略确保每个位置上最常被复用且关键的子图被优先预加载。

4 实现

SparseLoom 使用 PyTorch [39]以 Python 实现。它支持两种在边缘 SoC 上部署的配置：适用于英特尔片上系统的 OpenVINO [21]，以及适用于英伟达片上系统的 ONNX Runtime [31] 与 TensorRT [35]。

英特尔片上系统上的 OpenVINO：OpenVINO 被用作英特尔片上系统上的推理引擎，因为它针对这类硬件进行了专门优化。OpenVINO 支持 FP16 和 INT8量化，也支持对非结构化剪枝和结构化剪枝进行加速。为了在英特尔片上系统上使用 OpenVINO 执行子图，SparseLoom 使用 OpenVINO API 编译模型：Core().compile_model(model, device_name)。为了通过中央处理器、图形处理器和 NPU 实现并行子图执行，SparseLoom 使用 OpenVINO 的异步接口，在不同线程中启动独立的推理请求：infer_request.start_async()。

在英伟达片上系统上使用 TensorRT 的 ONNX Runtime：对于英伟达片上系统，SparseLoom 采用 ONNX Runtime 和 TensorRT 作为推理引擎。ONNX Runtime 提供统一接口，可在图形处理器和 ARM CPU 上执行子图。TensorRT 支持低精度计算（例如 FP16 和 INT8）及结构化剪枝，可实现高性能图形处理器执行。要在英伟达片上系统上运行子图，SparseLoom 首先将每个子图导出为 ONNX模型。对于图形处理器执行，它会实例化一个 InferenceSession，并使用 TensorrtExecutionProvid er 或 CUDAExecutionProvider。对于中央处理器执行，则使用 CPUExec utionProvider。子图推理通过 ort.InferenceSession(model, providers) 在线程中执行，以支持并发执行。

5 评估

本节中，我们在三个边缘平台和四个数据集–模型对上评估 SparseLoom，并将其与六种基线方法进行比较。主要观察结果如下：

- **端到端性能提升**：与其他基线方法相比，SparseLoom 将 SLO 违约率降低了最多 74%，并将吞吐量提升了最多 2.31× 。
- **各个模块均表现出色**：借助高效的估算器，SparseLoom 将性能分析时间缩短了最多 99%；通过优化处理器放置，将吞吐量提升了最多 2× ；同时，在不影响 SLO 违约率的情况下，平均降低了 28% 的内存开销。

5.1 实验设置

平台： 表3展示了评估中使用的三个平台。这些平台涵盖了不同类型的边缘 SoC，从高端台式机到低功耗嵌入式设备。

任务、数据集和基础模型： 表4展示了四项任务及其对应的数据集和模型。我们涵盖了视觉、语言、活动识别和音频等多种任务，每项任务都配有具有代表性的数据集和基础模型。这些任务、数据集和模型常用于边缘端侧机器学习 [24, 45]。

稀疏模型库： 对于每项任务，我们构建了一个包含十种文献中常用变体的稀疏模型库 [12,44, 47], ，其中包括一个稠密基础模型和九个稀疏变体。不过，SparseLoom 并不要求变体数量固定，因为它通过模型拼接以指数级扩展变体空间，而不是依赖预先生成的变体。基础模型取自公开模型仓库（例如 Hugging Face），相应的稀疏变体则使用 OpenVINO 神经网络压缩框架（NNCF） [20] 和 ONNX Runtime [31]等模型压缩框架预先生成。附录 A 列出了所有变体，并详细介绍了其准备方式以及所需的软件和硬件支持。

指标： 测量的指标如下：*SLO* 违规率——我们将 SLO 违规率定义为未能满足延迟或准确率 SLO 的任务所占比例。为全面评估这一指标，我们报告所有任务到达组合的平均违规率。任务到达组合指所有任务的一种特定排列顺序。对于实验中使用的四项任务，共有 24 种组合。吞吐量 ——单位时间内完成的推理查询数。具体而言，我们反复运行四项任务，每项任务的批大小为 1，每次运行 100

个查询（记为一次运行），并将吞吐量计算为 $\frac{N}{T_{\text{total}}}$，其中 N 为完成的推理查询总数，T_{total} 为总执行时间。所有结果均取 10 次运行的平均值[3]。

基线设计：为了全面评估 SparseLoom，我们从两个维度对现有的多 DNN 推理系统进行分类。第一个维度是变体选择策略。系统可以为每个任务采用单一变体（SV），该变体可以是准确率最优（AO）或延迟最优（LO）；也可以支持每个任务的自适应变体（AV），以适应不同的 SLO 配置。代表性示例包括：SV-AO 类系统中的 Pipe-it [50], Pantheon [14], 和 RT-mDL [29]；SV-LO 类系统中的 Hetero^{2}Pipe [45], Band [23],和 OmniBoost [24]；以及 AV 类系统中的 Tango [47], ESIM [44], 和 NestDNN [12]。第二个维度是系统是否采用变体分区。分区（P）系统将变体拆分为子图，以便在处理器上并发执行 [23, 28, 45], ；而非分区（NP）系统则将每个变体视为一个单体单元 [14, 15, 50]。

这两个维度共同构成六种基线方法：**SV-AO-P**、**SV-AO-NP**、**SV-LO-P**、**SV-LO-NP**、**AV-P**和**AV-NP**，覆盖了最先进的多 DNN 推理系统的特征空间。SparseLoom 属于自适应变体分区（AV-P）类别。不过，它通过模型拼接超越了以往的系统。

SLO 配置：为在多种 SLO 下评估 SparseLoom 和其他基线方法，我们系统地构建了一组准确率–延迟配置。对于每项任务，我们首先测量所有稀疏变体的准确率和延迟，并记录观测到的准确率和延迟范围。随后，我们扩展这些范围：将延迟范围扩展 20%（从最低延迟的 80% 到最高延迟的 120%），将准确率范围扩展 ±2%（从最低准确率低 2% 到最高准确率高 2%）。基于扩展后的范围，我们分别从准确率范围和延迟范围中均匀采样五个点。这两个集合的笛卡尔积得到 $5 \times 5 = 25$ 个 SLO 配置。

例如，如果所有变体的观测准确率范围为 [85%，92%]，延迟范围为 [50ms，120 ms],，则扩展后的范围分别为 [83%，94%]（准确率）和[40ms，144 ms]（延迟）。随后，我们在准确率范围内均匀采样五个点，例如 {83%, 85.75%, 88.5%, 91.25%, 94%}，并在延迟范围内均匀采样五个点，例如 {40ms, 66ms, 92ms, 118ms, 144 ms}。最终的 SLO 配置由这两组点的笛卡尔积构成，共得到 25 对准确率–延迟组合。

5.2 端到端性能

较低的 SLO 违规率：图10展示了 SparseLoom 和其他基线方法在三个测试平台上并行运行四项任务时的 SLO 违规率。在所有基线方法中，SV-LO 方法（SV-LO-P 和 SV-LO-NP）的违规率最高，因为它们为每项任务采用单一变体，且仅针对延迟进行优化。相比之下，从稀疏变体中进行自适应变体选择可显著降低违规率。在所有三个测试平台上，SparseLoom 和 AV 方法（AV-P 和 AV-NP）的违规率均远低于 SV 方法（包括 SV-AO-P、SV-AO-NP、SV-LO-P 和 SV-LO-NP）。此外，SparseLoom 的表现进一步优于其他 AV 方法，凸显了将模型拼接融入自适应变体系统的优势。具体而言，与 SV 方法相比，SparseLoom 最多可将违规率降低 74%（在 Jetson Orin 上，相较于 SV-LO-NP）；与 AV 方法相比，最多可降低 24.7%（在台式机上，相较于 AV-NP）。总体而言，SparseLoom 始终能够减少 SLO 违约。

我们进一步评估了 SparseLoom 在准确率和延迟保证型 *SLO*下的表现，其中，准确率要求分别固定为所有变体中的最高值，或将延迟要求固定为最低值（详情见附录 D）。在准确率保证的 SLO 下，SparseLoom 始终能将 SLO 违约率最多降低 73.6%；在延迟保证型 SLO 下，最多降低 68.2%。这表明，即使在不允许任何准确率或延迟折衷的极端场景中，SparseLoom 仍然有效。

更高的推理吞吐量：图11展示了 SparseLoom 与其他基线方法在三个测试平台上的推理吞吐量。分区方法（SV-AO-P、SV- LO-P 和 AV-P）的吞吐量高于对应的非分区方法（SV-AO-NP、SV-LO-NP 和 AV- NP），这是因为子图可在多个处理器上并行执行。总体而言，SV 和 AV 方法的吞吐量相近；即使是贪心选择延迟最低变体的 SV-LO-P，相较于其他基线方法也仅带来微幅提升。相比之下，SparseLoom 的吞吐量显著更高：与笔记本电脑上的 SV-AO-NP 相比，最高提升 2.31×；与台式机上表现最佳的基线方法（SV-LO-P）相比，最高提升 1.53×。这些提升源于模型拼接，它能提供更多拼接变体和优化的放置顺序（见第 5.3 节）。

总而言之，SparseLoom 始终优于最先进的基线方法，在降低 SLO 违约率的同时提高推理吞吐量。在各种边缘 SoC 上，它最多可减少 74% 的 SLO 违约，并将吞吐量提高至 2.31× 倍。

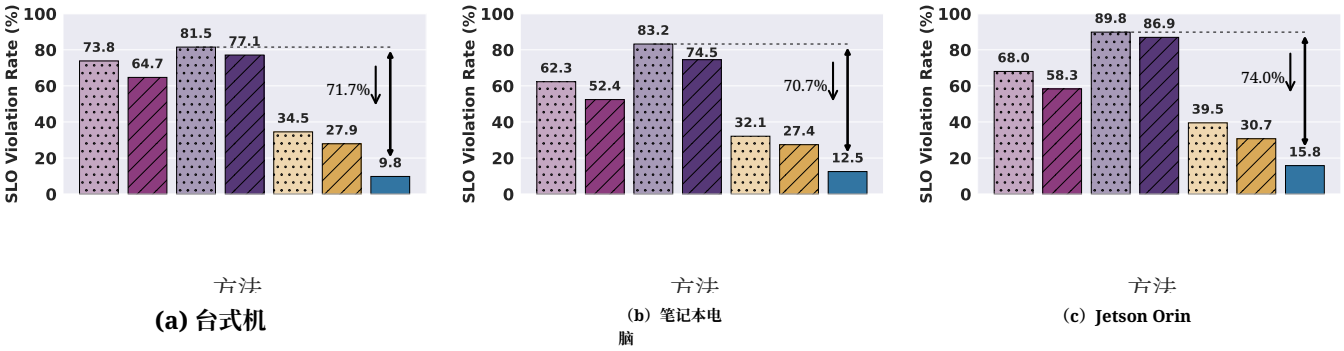
5.3 各个模块的评估

在本节中，我们进一步评估 SparseLoom 的三个主要模块。具体而言，我们考察：(i) 使用和不使用估算器时，准确率和延迟的性能分析时间；(ii) 处理器放置优化对吞吐量的影响，

[3]由于多次运行的结果方差较小，因此未显示误差线。

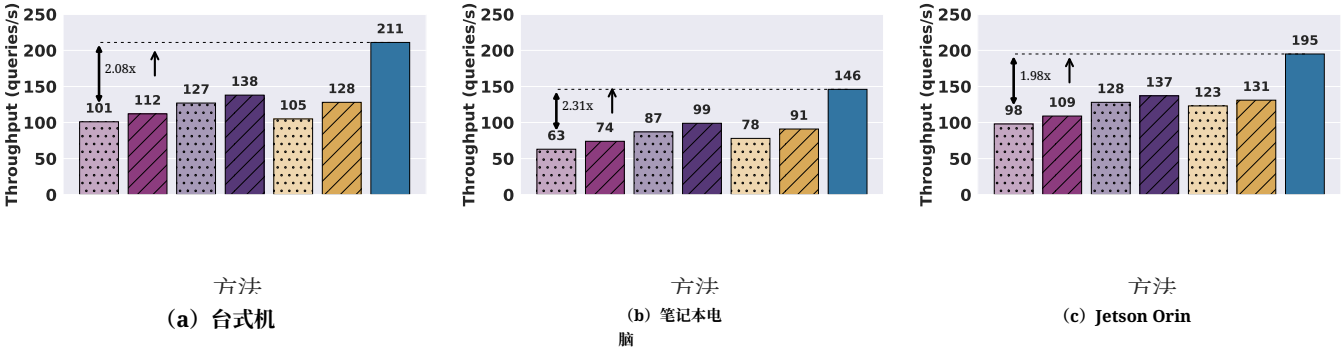
表3：评估中使用的平台。

平台	CPU	GPU	NPU
台式机（英特尔酷睿 Ultra 7 265K）	x86-64；20 核	4-X^{e}核	英特尔 AI Boost
笔记本电脑（英特尔酷睿 Ultra 5 135U）	x86-64；12 核	4-X^{e}核	英特尔 AI Boost
英伟达 Jetson AGX Orin（MAXN 模式）	ARM Cortex 12 核 2048 核 Ampere；64 个 Tensor 核		N/A



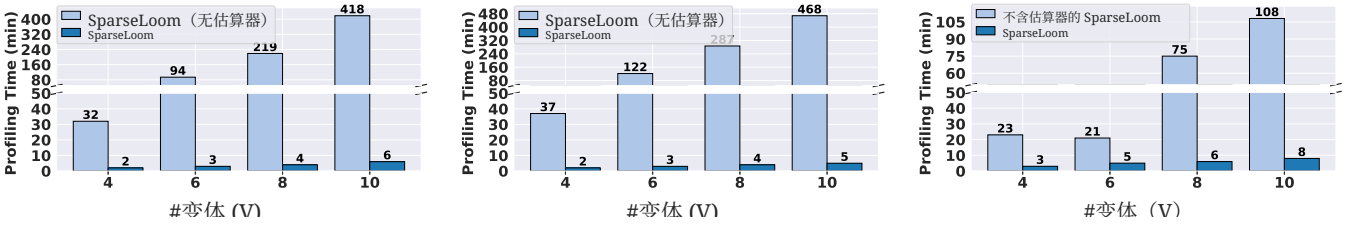
(a) 台式机 **(b) 笔记本电脑** **(c) Jetson Orin**

图10：SparseLoom 与基线方法在三种片上系统上的 SLO 违约率。SparseLoom 始终实现了低于所有基线方法的违约率，减少幅度最高达 74%。



（a）台式机 **（b）笔记本电脑** **（c）Jetson Orin**

图11：SparseLoom 与基线方法在三种片上系统上的推理吞吐量。SparseLoom 始终实现了比所有基线方法更高的吞吐量，提升幅度最高达 2.31×。

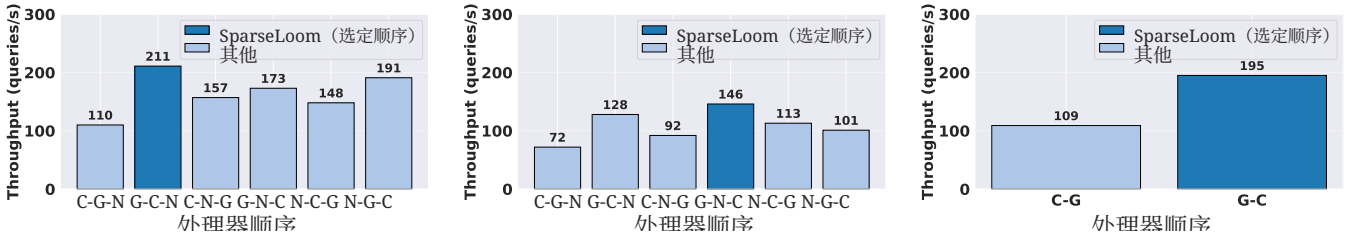


(a) 台式机 **(b) 笔记本电脑** **(c) Jetson Orin**

图12：SparseLoom 使用和不使用估算器时的性能分析时间（分钟）。在所有片上系统上，SparseLoom 都能将分析成本降至 8 分钟以内（最多减少 99%）。

以及 (iii) 在不同内存预算下，子图预加载对 SLO 违规的影响。

SparseLoom 的性能分析开销SparseLoom：图12比较了 SparseLoom 在三个测试平台上使用和不使用准确率与延迟估算器时的性能分析时间。随着



(a) 台式机 (b) 笔记本电脑 (c) Jetson Orin

图 13：处理器放置顺序对推理吞吐量的影响（C = 中央处理器，G =图形处理器，N = NPU）。SparseLoom 会自动选择最佳放置方式，使吞吐量最高提升 2× 。

表 4：用于评估的任务、数据集和模型。所选任务和模型能够代表第 1 节讨论的增强现实应用场景。

任务类型	数据集	基础模型
图像分类	ImageNet-1K [8]	ResNet-101 [16]
情感分类	SST-2 [46]	BERT-Base [9]
人体活动识别	HAR [30]	ViT-Small [10]
语音识别	LibriSpeech ASR [38]	Wav2vec2 [3]

变体数量增加，不使用估算器时的性能分析开销呈指数增长。例如，在包含 10 个变体（即我们实验中采用的设置）时，在笔记本电脑上不使用估算器进行性能分析约需 468 分钟，而使用估算器后，SparseLoom 将时间缩短至 5 分钟，减少了 99%。此外，随着变体数量增加，SparseLoom 使用估算器时的性能分析时间仅略有增长，展现出良好的可扩展性。

处理器放置优化： 图 13 展示了不同处理器顺序下的推理吞吐量。吞吐量差异显著：台式机上为 110 至 211，笔记本电脑上为 72 至 146，Jetson Orin 上为 109 至 195，最高可达 2× 差异。不同测试平台的最优顺序也各不相同：台式机上为 G–C–N，笔记本电脑上为 G–N–C，Jetson Orin 上为 G–C。这些结果凸显了 SparseLoom 所提供的优化处理器放置的重要性。借助其稀疏性感知优化器，SparseLoom 可自动选择最佳顺序，以最大化吞吐量。

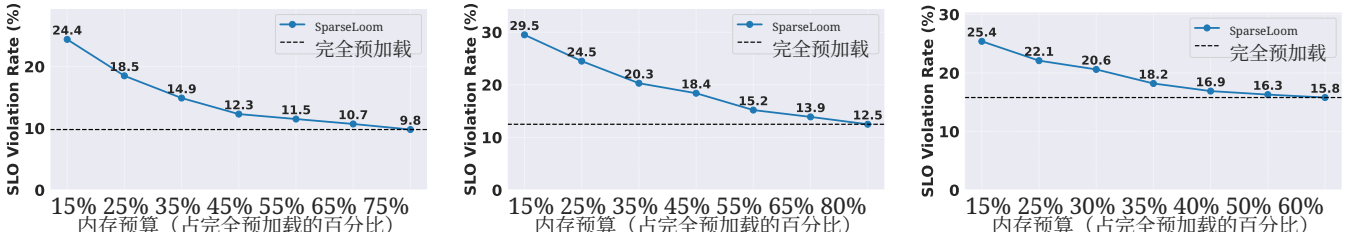
子图预加载：我们进一步评估了 SparseLoom 的 *Hot-SubgraphsPreloader* 在不同内存预算下的表现。图 14 展示了 SparseLoom 在三种片上系统上采用不同内存预算时的 SLO 违规率。在此情形下，我们将内存预算定义为完全预加载所有子图所需内存的占比，并将这种方式称为完全预加载。在给定硬件平台上，完全预加载能够达到最低的 SLO 违规率。正如预期，随着内存预算缩小，SparseLoom 的 SLO 违规率会上升，因为能够预加载的拼接变体更少。然而，仅使用完全预加载所需内存的 15%，SparseLoom 在台式机、笔记本电脑和 Jetson Orin 上的违规率便分别达到 24.4%、29.5% 和 25.4%，已经优于使用全部内存的 AV 方法。预算为 55% 时，其 SLO 违规率与完全预加载相当，差距在 2.7% 以内。若要达到与完全预加载相同的违规率，SparseLoom 在台式机、笔记本电脑和 Jetson Orin 上分别可减少 25%、20% 和 40% 的内存使用量。这些结果表明，*Hot-Subgraphs Preloader* 能够在维持相当的 SLO 性能的同时，大幅降低内存使用量。

5.4 讨论

处理器间执行开销 在 SparseLoom 中包括：(i) 将中间激活值从一个处理器传输到下一个处理器所产生的通信成本；以及 (ii) 由数值精度差异引起的格式转换成本。例如，当拼接变体按图形处理器–中央处理器–NPU 的顺序执行时，图形处理器使用 FP32 精度，而中央处理器和 NPU 使用 INT8，中间激活值必须从图形处理器传输到中央处理器，再从中央处理器传输到 NPU，同时输出精度也必须从 FP32 转换为 INT8。因此，会产生通信开销和格式转换开销。

我们通过实测发现，这种处理器间的执行延迟几乎占总推理延迟的 5%。我们实验中使用的处理器采用统一内存架构 [18, 19, 34],，格式转换的开销相对较小，并由推理引擎自动处理。

划分后的子图数量也会影响推理延迟。增加子图数量可以实现更细粒度的调度。然而，并行度并不会无限提升，因为它受可用处理器数量的限制。在我们的实验中，模型子图的数量等于处理器数量。这种配置也普遍应用于现有的多 DNN 推理系统 [24, 25, 45]。



(a) 台式机 (b) 笔记本电脑 (c) Jetson Orin

图 14：内存预算（完全预加载所有子图所需内存的百分比）对 SLO 违约的影响。SparseLoom 平均将内存用量降低 28%，且不提高违约率。

动态电压与频率调节（DVFS）的影响。在所有实验中，我们均采用默认的 DVFS 策略，未进行任何手动调整。然而，我们没有观察到 DVFS 在执行期间主动调节处理器频率。触发 DVFS 时，它可能会降低处理器频率，从而增加所有变体的推理延迟。这会影响所有基线方法，而不仅仅是 SparseLoom。为减轻这一影响，SparseLoom 可以采用一种简单机制，在新的 DVFS 条件下重新分析所有变体，以确定最佳处理器顺序。

对SparseLoom 大型基础模型的适用性。SparseLoom 面向边缘级深度神经网络，包括卷积神经网络和基于 Transformer 的模型。即使采用轻量级深度神经网络，由于边缘资源的计算能力有限，多深度神经网络并行执行仍会导致较高的 SLO 违约率。相比之下，大型基础模型（例如大型语言模型）仍难以部署在边缘 SoC 上，通常以单个多模态模型而非多个深度神经网络的形式提供服务。优化此类模型的执行效率不在 SparseLoom 的研究范围内；它们的瓶颈本质上不同，主要受极大的内存需求和内存访问开销所限 [52]。

6 相关工作

我们将 SparseLoom 与三类多 DNN 推理系统和变体生成方法进行比较，附录 C 总结了它们的主要特征。

第 ***1*** 类-单处理器上的任务级调度系统：这类系统通过优化单个处理器上多个深度神经网络任务的执行顺序，以满足延迟约束。代表性系统包括 Pipe-it [50]、Pantheon [14], 和 REEF [15]。它们通常采用优先级调度和期限感知排序等技术，以最大限度地利用有限的计算资源。这类系统的一个主要局限是缺乏对异构处理器的支持，导致计算资源利用不足。

类别 ***2***-异构处理器上的子图放置系统：这类系统将 DNN 划分为子图，并将其分配给异构处理器。具有代表性的系统包括 Hetero^{2}Pipe [45], Band [23], 和 OmniBoost [24]。这些方法通过在不同处理器上并行执行子图来提高系统吞吐量。然而，它们假设每个任务只有一个模型，因此不支持选择多个变体来适应不同的 SLO。

类别 ***3***-面向多样化 *SLO* 的多变体选择系统：这类系统支持推理框架从多个模型变体中进行选择，以满足不同的准确率与延迟要求。代表性工作包括 Tango [47]、ESIM [44], 和 NestDNN [12]。通常，这些系统会准备一组准确率和延迟性能各不相同的模型变体，并依据性能分析数据、历史性能和启发式规则进行选择，以平衡延迟与准确率。然而，由于可用模型变体数量有限，它们支持多样化 SLO 的能力往往受到限制。

变体生成方法 例如自适应稀疏性 [42]和逐层剪枝 [41]，可以生成少量模型变体（通常为数十个），与 SparseLoom 所采用的模型拼接方法类似。然而，这些方法需要额外的计算资源和校准数据集来进行重新训练。相比之下，SparseLoom 无需数据和训练，且能够显著扩大变体空间。此外，这些方法生成的变体单独评估时可能满足 SLO，但在多 DNN 执行时却可能违反 SLO。SparseLoom 超越了变体生成方法，通过提供一个涵盖变体生成、选择和放置的整体系统来实现这一点。

7 结论

在本文中，我们提出了 SparseLoom，这是首个利用模型拼接来扩展模型变体空间并降低 SLO 违约率的多 DNN推理系统。为高效支持拼接，SparseLoom 开发了轻量级性能分析器、吞吐量优化的处理器放置模块，以及内存高效的子图预加载器。在各种边缘 SoC 上的评估表明，与最先进的系统相比，SparseLoom 最多可减少 74% 的 SLO 违规，将吞吐量最多提升 2.31×，并将内存开销平均降低 28%。

参考文献

[1] Sohaib Ahmad、Hui Guan、Brian D Friedman、Thomas Williams、Ramesh K Sitaraman 和 Thomas Woo。2024。Proteus：一种具有精度扩展功能的高吞吐量推理服务系统。载于第 *29* 届 *ACM* 编程语言与操作系统架构支持国际会议论文集，第 *1* 卷。318–334。[2] Sohaib Ahmad 、 Hui Guan 和 Ramesh K Sitaraman。2024。Loki：一种支持硬件和精度扩展的机器学习推理流水线服务系统。载于第 *33* 届高性能并行与分布式计算国际研讨会论文集。267–280。[3] Alexei Baevski、Yuhao Zhou、Abdelrahman Mohamed 和 Michael Auli。2020。wav2vec 2.0：一种用于自监督学习语音表示的框架。神经信息处理系统进展 33（2020），12449–12460。[4] Yamini Bansal、Preetum Nakkiran 和 Boaz Barak。2021。重新审视模型拼接以比较神经表示。神经信息处理系统进展 34（2021），225–236。[5] Tianqi Chen 和 Carlos Guestrin。2016。XGBoost：一种可扩展的树提升系统。载于第 *22* 届 *ACM* 知识发现与数据挖掘国际会议论文集。785–794。[6] Hongrong Cheng、Miao Zhang 和 Javen Qinfeng Shi。2024。深度神经网络剪枝综述：分类、比较、分析与建议。*IEEE*模式分析与机器智能汇刊 （2024）。[7] Daniel Crankshaw、Gur-Eyal Sela、Xiangxi Mo、Corey Zumar、Ion Stoica、Joseph Gonzalez 和 Alexey Tumanov。2020。InferLine：面向预测服务流水线的延迟感知资源配置与扩展。载于第 *11* 届 *ACM*云计算研讨会论文集。477–491。[8] Jia Deng、Wei Dong、Richard Socher、Li-Jia Li、Kai Li 和 Li Fei-Fei。2009。ImageNet：一个大规模分层图像数据库。载于*IEEE*计算机视觉与模式识别会议论文集。248–255。[9] Jacob Devlin、Ming-Wei Chang、Kenton Lee 和 Kristina Toutanova。2019。BERT：用于语言理解的深度双向Transformer预训练。载于计算语言学协会北美分会会议论文集。4171–4186。[10] Alexey Dosovitskiy、Lucas Beyer、Alexander Kolesnikov、Dirk Weissenborn、Xiaohua Zhai、Thomas Unterthiner、Mostafa Dehghani、Matthias Minderer、Georg Heigold、Sylvain Gelly 等。2021。一张图像胜过 16×16 个词：用于大规模图像识别的 Transformer。国际学习表征会议 （2021）。[11] Bailey J Eccles、Philip Rodgers、Peter Kilpatrick、Ivor Spence 和 Blesson Varghese。2024。DNNShifter：一种面向边缘计算的高效 DNN剪枝系统。未来一代计算机系统 152（2024），43–54。[12] Biyi Fang、Xiao Zeng 和 Mi Zhang。2018。NestDNN：面向连续移动视觉的资源感知多租户设备端深度学习。载于第 *24* 届年度国际移动计算与网络会议论文集。115–127。[13] Amir Gholami、Sehoon Kim、Zhen Dong、Zhewei Yao、Michael W Mahoney 和 Kurt Keutzer。2022。高效神经网络推理的量化方法综述。载于低功耗计算机视觉。Chapman and Hall/CRC，291–326。[14] Lixiang Han、Zimu Zhou 和 Zhenjiang Li。2024。Pantheon：移动边缘GPU上的可抢占多DNN推理。载于第 *22* 届年度国际移动系统、应用与服务会议论文集。465–478。[15] Mingcong Han、Hanze Zhang、Rong Chen 和 Haibo Chen。2022。并发 GPU 加速 DNN 的微秒级抢占推理。收录于第*16*届*USENIX*操作系统设计与实现研讨会论文集。539–558。[16] Kaiming He、Xiangyu Zhang、Shaoqing Ren 和 Jian Sun。2016。用于图像识别的深度残差学习。收录于*IEEE*计算机视觉与模式识别会议论文集。770–778。[17] Yihui He、Ji Lin、Zhijian Liu、Hanrui Wang、Li-Jia Li 和 Song Han。2018。AMC：面向移动设备的模型压缩与加速自动机器学习。收录于欧洲计算机视觉会议论文集。784–800。[18] 英特尔公司。2023年。英特尔酷睿Ultra 5处理器135U。网址：https://www.intel.com/content/www/us/en/products/sku/237328/intel-core-ultra-5-processor-135u-12m-cache-up-to-4-40-ghz/specifications.html。[19] 英特尔公司。2023年。英特尔酷睿Ultra 7处理器265K。网址：https://www.intel.com/content/www/us/en/products/sku/241063/intel-core-ultra-7-processor-265k-30m-cache-up-to-5-50-ghz/specifications.html。[20] 英特尔公司。2025年。神经网络压缩框架（NNCF）。网址：https://github.com/openvinotoolkit/nncf。[21] 英特尔公司。2025年。OpenVINO工具套件。网址：https://github.com/openvinotoolkit/openvino。[22] Venkatraman Iyer、Sungho Lee、Semun Lee、Juitem Joonwoo Kim、Hyunjun Kim 和 Youngjae Shin。2023。面向多模型设备端人工智能推理的自动化后端分配。*ACM*计算系统测量与分析论文集7，3（2023），1–33。[23] Joo Seong Jeong、Jingyu Lee、Donghyun Kim、Changmin Jeon、Changjin Jeong、Youngki Lee 和 Byung-Gon Chun。2022。Band：异构移动处理器上的协同多DNN推理。收录于第*20*届移动系统、应用与服务国际年会论文集。235–247。[24] Andreas Karatzas 和 Iraklis Anagnostopoulos。2023。OmniBoost：在多DNN工作负载下提升异构嵌入式设备的吞吐量。收录于第*60*届*ACM/IEEE*设计自动化会议论文集。1–6。[25] Andreas Karatzas、Dimitrios Stamoulis 和 Iraklis Anagnostopoulos。2025。RankMap：面向异构嵌入式设备的优先级感知多深度神经网络管理器。收录于欧洲设计、自动化与测试会议论文集。1–7。[26] Seulki Lee 和 Shahriar Nirjon。2020。通过神经网络权重虚拟化实现快速、可扩展的内存中深度多任务学习。收录于第*18*届移动系统、应用与服务国际会议论文集。175–190。[27] Yuhang Li、Ruihao Gong、Xu Tan、Yang Yang、Peng Hu、Qi Zhang、Fengwei Yu、Wei Wang 和 Shi Gu。2021。BRECQ：通过块重建突破训练后量化的极限。第九届国际学习表征会议（2021）。[28] Neiwen Ling、Xuan Huang、Zhihe Zhao、Nan Guan、Zhenyu Yan 和 Guoliang Xing。2022。BlastNet：利用双块实现跨处理器实时DNN推理。收录于第*20*届*ACM*嵌入式网络传感器系统会议论文集。91–105。[29] Neiwen Ling、Kai Wang、Yuze He、Guoliang Xing 和 Daqi Xie。2021。RT-MDL：支持边缘平台上的实时混合深度学习任务。收录于第*19*届*ACM*嵌入式网络传感器系统会议论文集。1–14。[30] Meet Nagadia。2021。人体动作识别（HAR）数据集。网址：https://www.kaggle.com/datasets/meetnagadia/human-action-recognition-har-dataset。[31] 微软公司。2025年。ONNX Runtime工具套件。网址：https://github.com/microsoft/onnxruntime。[32] Mohammad Reza Mirzaei、Seyed Ghorshi 和 Mohammad Mortazavi。2014。增强现实环境中的视听语音识别技术。*The Visual Computer* 30，3（2014），245–257。

[33] Aadi Swadipto Mondal、Yuang Zhu、Kaushal Kumar Bhagat 和 Nasser Giacaman。2024。交互式教育应用用户评论分析：一种情感分析方法。交互式学习环境32，1（2024），355–372。[34] 英伟达公司。2022年。NVIDIA Jetson AGX Orin 系列技术简报。https://www.nvidia.com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf。[35] 英伟达公司。2025年。NVIDIA TensorRT：高性能深度学习推理。https://developer.nvidia.com/tensorrt。[36] Xiaomin Ouyang、Xian Shuai、Jiayu Zhou、Ivy Wang Shi、Zhiyuan Xie、Guoliang Xing 和 Jianwei Huang。2022。Cosmo：利用小规模数据进行多模态人体活动识别的对比融合学习。收录于第*28*届年度国际移动计算与网络会议论文集。324–337。[37] Zizheng Pan、Jianfei Cai 和 Bohan Zhuang。2023。可拼接神经网络。收录于*IEEE/CVF*计算机视觉与模式识别会议论文集。16102–16112。[38] Vassil Panayotov、Guoguo Chen、Daniel Povey 和 Sanjeev Khudanpur。2015。LibriSpeech：基于公有领域有声读物的自动语音识别语料库。收录于*IEEE*国际声学、语音与信号处理会议论文集。5206–5210。[39] Adam Paszke、Sam Gross、Francisco Massa、Adam Lerer、James Bradbury、Gregory Chanan、Trevor Killeen、Zeming Lin、Natalia Gimelshein、Luca Antiga 等。2019。PyTorch：一种命令式风格的高性能深度学习库。神经信息处理系统进展32（2019）。[40] Maithra Raghu、Justin Gilmer、Jason Yosinski 和 Jascha Sohl-Dickstein。2017。SVCCA：用于深度学习动态与可解释性的奇异向量典型相关分析。神经信息处理系统进展30（2017）。[41] Youngmin Ro 和 Jin Young Choi。2021。AutoLR：深度网络微调中的逐层剪枝与学习率自动调优。收录于*AAAI*人工智能会议论文集，第35卷。2486–2494。[42] Victor Sanh、Thomas Wolf 和 Alexander Rush。2020。移动剪枝：通过微调实现自适应稀疏性。神经信息处理系统进展33（2020），20378–20389。[43] Mahadev Satyanarayanan 和 Nigel Davies。2019。通过边缘计算增强认知。*Computer*52，7（2019），37–46。[44] Tanmoy Sen 和 Haiying Shen。2024。在边缘设备上弹性调度多个DNN推理任务，以实现高效率和高准确率。收录于第*21*届*IEEE*移动自组网与智能系统国际会议论文集。185–193。[45] Yuhao Shen、Zichen Wang、Tianyi Wang、Chaojie Gu、Zhenyu Wen、Yuanchao Shu 和 Cong Wang。2025。Hetero^{2}Pipe：针对协同执行减速的异构移动处理器多DNN推理流水线。收录于*IEEE*分布式计算系统国际会议论文集。[46] Richard Socher、Alex Perelygin、Jean Wu、Jason Chuang、Christopher D Manning、Andrew Y Ng 和 Christopher Potts。2013。情感树库上的语义组合性递归深度模型。收录于自然语言处理实证方法会议论文集。1631–1642。[47] Zain Taufique、Aman Vyas、Antonio Miele、Pasi Liljeberg 和 Anil Kanduri。2024。Tango：异构边缘平台上的低延迟多DNN推理。收录于第*42*届*IEEE*计算机设计国际会议论文集。300–307。[48] Neural Magic 团队。2025。DeepSparse：面向CPU的稀疏性感知深度学习推理运行时。https://github.com/neuralmagic/deepsparse。[49] Rajendra Thilahar 和 R Sivaramakrishnan。2019。用于增强现实中的特征选择与图像分类的模糊神经遗传方法Reality Systems。《国际机器人与自动化期刊》8, 3 (2019), 194–204。[50] SiqiWang、GayathriAnanthanarayanan、Yifan Zeng、NeerajGoel、Anuj Pathania 和 Tulika Mitra。2019。嵌入式 ARM big.LITTLE 多核处理器上的高吞吐量 CNN 推理。《IEEE 集成电路与系统计算机辅助设计汇刊》39, 10 (2019), 2254–2267。[51] Di Wu、Qi Tang、Yongle Zhao、Ming Zhang、Ying Fu 和 Debing Zhang。2020。Easyquant：通过尺度优化实现训练后量化。arXiv 预印本 arXiv:2006.16669 (2020)。[52] Zhenliang Xue、Yixin Song、Zeyu Mi、Xinrui Zheng、Yubin Xia 和 Haibo Chen。2024。Powerinfer-2：在智能手机上实现快速大语言模型推理。ACM SIGOPS 第30届操作系统原理研讨会论文集 (2024)。[53] Huanrui Yang、Hongxu Yin、Maying Shen、Pavlo Molchanov、Hai Li 和 Jan Kautz。2023。利用 Hessian 感知显著性进行全局视觉 Transformer 剪枝。收录于《IEEE/CVF计算机视觉与模式识别会议论文集》。18547–18557。[54] Zhihang Yuan、Chenhao Xue、Yiqi Chen、Qiang Wu 和 Guangyu Sun。2022。Ptq4vit：通过双均匀量化对视觉 Transformer 进行训练后量化。收录于《欧洲计算机视觉会议》。Springer，191–207。

A 稀疏变体

所用变体：表5展示了评估中使用的完整稀疏变体集合。对于每项任务，我们构建了一个包含十种变体的稀疏模型库，以涵盖多种稀疏模式。该集合包括一个稠密基础模型、一个适用于英特尔片上系统的量化模型（INT8）、两个适用于 Jetson Orin 的量化模型（FP16 和 INT8） [51], ，其余变体则通过剪枝获得 [11] ——适用于英特尔片上系统的八个剪枝模型，以及适用于 Jetson Orin 的七个剪枝模型——均以稠密基础模型为基础。

创建变体： 在英特尔片上系统上，我们使用 OpenVINO 的神经网络压缩框架（NNCF），通过结构化剪枝（改变架构）、非结构化剪枝（使用零掩码） [6], 和量化来生成稀疏变体。在英伟达片上系统上，我们采用 ONNX 框架来生成剪枝（结构化）变体和量化变体。这些压缩框架已在现实世界中得到广泛采用，可在几分钟内为每个模型生成适用于边缘部署的稀疏变体。所有压缩方法均在训练后应用，因此需要数百个校准样本。在我们的实验中，这些变体存储在持久化存储中（例如磁盘上），涵盖全部四项任务的所有稀疏模型库，总计 7.6 GB。

不同压缩类型的软件/硬件支持：结构化剪枝不依赖特定硬件或软件。然而，非结构化剪枝依赖稀疏加速软件（例如 DeepSparse [48]），且独立于底层硬件平台。量化需要软件和硬件同时支持。大多数具有代表性的片上系统（例如 NVIDIA Jetson 系列）均支持 8 位量化。在我们的实验中，Intel Ultra 5 和 Ultra 7 平台支持全部三种压缩类型，而 Jetson Orin 不支持非结构化剪枝。

表5：各平台上的稀疏变体。稀疏模式 **表示模型参数的稀疏性级别和数值精度。注：** x**% 的稀疏性表示** x**% 的权重经过剪枝。**

平台	变体类型	稀疏模式
英特尔片上系统（台式机 / 笔记本电脑）	基础模型	稠密（FP32）
	量化模型	稠密（INT8）
	非结构化剪枝	90%，FP32
		85%，FP32
		80%，FP32
		75%，FP32
		70%，FP32
		65%，FP32
	结构化剪枝	40%, FP32
		50%, FP32
NVIDIA Jetson	基础模型	稠密（FP32）
	量化模型	稠密（FP16）
		稠密（INT8）
	结构化剪枝	20%，FP32
		30%，FP32
		35%，FP32
		40%，FP32
		45%，FP32
		50%，FP32
		55%，FP32

B记法

本文使用的符号汇总于表6。

表6：符号说明。

符号说明	描述
t, T	任务索引和任务总数
i, V	变体索引和每个任务的变体数
j, S	子图索引和每个变体的子图数
k	拼接变体索引（共 V^S 个变体）
$v^{t,i}$	任务 t中的变体 i
$s^{t,i}_{j}$	任务 t中变体 i的子图 j
$_{\vec{v}^t,k}$	任务 t中的拼接变体 k
$\tilde{s}^{t,k}_{j}$	任务 t中拼接变体 k的子图 j
$_{\vec{v}^t,*}$	任务 t的最优拼接变体
p_j	分配给子图 j的处理器
$\vec{p}$	放置顺序 $\{p_1, \ldots, p_S\}$
$\vec{p}^*$	所有任务的最优放置顺序
Ω	所有 $P!$ 放置顺序的集合
延迟（$s^t,_j{}^i$, p_j）	$s^t,_j{}^i$ 在 p_j上的延迟
延迟（$\tilde{v}^t,^k$, $\vec{p}$）	$\tilde{v}^t,^k$ 的延迟（适用于 $\vec{p}$）
A（$\tilde{v}^t,^k$）	$\tilde{v}^t,_k$的准确率
SLO^t_{acc}	任务 t的准确率服务等级目标
SLO^t_{lat}	任务 t的延迟服务等级目标
Θ^t	同时满足 SLO^t_{acc}和 SLO^t_{lat}的变体
$Mem(s^t,_j{}^i)$	内存开销 $s^t,_i{}^j$
Mem_{budget}	总内存预算
Φ^t	任务 t的缓存的子图
σ	一个 SLO 配置
Ψ	SLO 配置集合

C 现有方法与 SparseLoom 的比较

表 7 比较了现有的多DNN推理系统类别、变体生成方法与 SparseLoom 在各项关键能力上的差异。第 1 类系统针对单处理器上的任务调度进行优化，第 2 类系统侧重于异构处理器上的子图放置，第 3 类系统则支持针对不同 SLO 的模型变体选择。然而，这三类系统仅提供单个模型或数十种变体，且不支持稀疏性感知放置和内存高效预加载。变体生成方法能够生成细粒度稀疏变体，但需要校准数据或重新训练，且规模仍然有限。相比之下，SparseLoom 无需数据和训练，支持数千种拼接变体，并整合了变体生成、选择、稀疏性感知放置和内存高效预加载，从而实现高效的多DNN推理。

D 在精度和延迟保证型 SLO 下比较 SparseLoom 与基线方法

我们将准确率保证的 *SLO* 定义为：准确率要求固定为所有变体中的最高值，同时改变延迟阈值。相反，延迟保证型 *SLO* 将延迟要求固定为最低值，同时改变精度阈值。

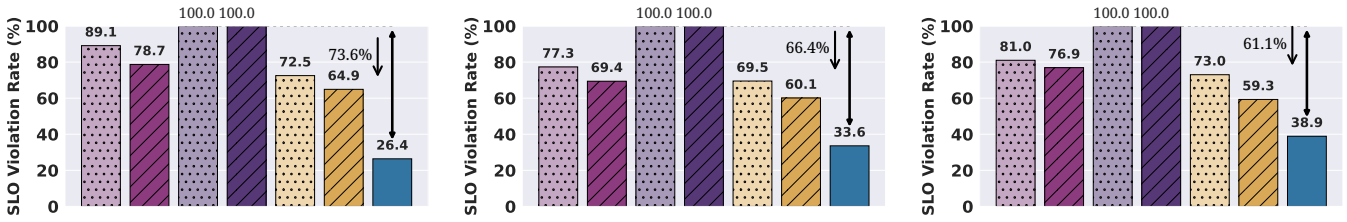
例如，假设所有变体的精度范围为 $[85\%, 92\%]$，延迟范围为 $[50\,\text{ms}, 120\,\text{ms}]$。对于准确率保证的 SLO，我们将准确率要求固定为 92%，并在 50 到 120 ms 之间均匀生成五个延迟阈值（例如，$\{50, 67.5, 85, 102.5, 120\}$ ms）。相反，对于延迟保证型 SLO，我们将延迟要求固定为 50 ms，并在 85% 到 92% 之间均匀生成五个精度阈值（例如，$\{85\%\ 86.75\%\ 88.5\%, 90.25\%, 92\%\}$）。

这些 SLO 刻画了要求无精度折衷 或 无延迟折衷的场景，反映了优先考虑准确率或延迟之一而非另一项的应用。

图 15 和图 16 展示了评估章节中介绍的三个硬件平台上 SparseLoom 和所有基线方法的 SLO 违约率。在准确率保证型和延迟保证型 SLO 下，SparseLoom 始终能够降低 SLO 违约率，降幅分别高达 73.6% 和 68.2%。这表明，SparseLoom 不仅能降低所有 SLO 的平均违约率，在准确率或延迟要求严格的场景中也依然有效。

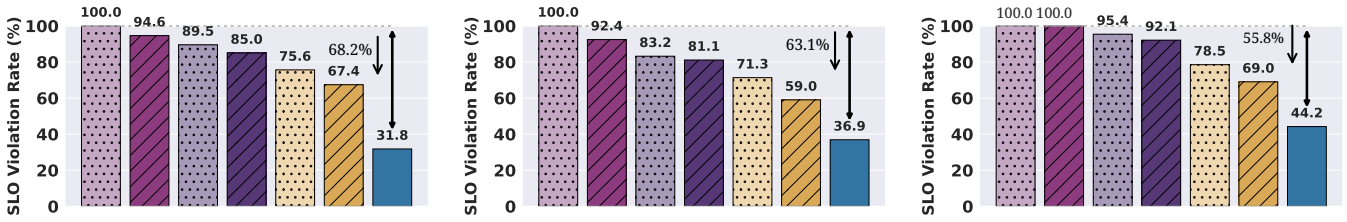
表 7：不同类别的多 DNN 推理系统、变体生成方法和 SparseLoom 的主要特征。类别 *1*：优化单处理器上任务级调度的系统；类别 *2*：优化异构处理器上子图放置的系统；类别 *3*：支持为多样化 SLO 选择多个变体的系统。变体生成方法可生成细粒度稀疏变体。

特征	类别 1 [14, 15, 50]	类别 2 [23, 24, 45]	类别 3 [12, 44, 47]	变体生成方法 [41, 42]	SparseLoom
无需数据和训练的变体生成	✗	✗	✗	✗	✓
多变体选择	✗	✗	✓	✗	✓
支持的模型变体数量	1	1	10的规模	10的规模	1000的规模
异构子图放置	✗	✓	✗	✗	✓
优化的稀疏性感知放置	✗	✗	✗	✗	✓
内存高效预加载	✗	✗	✗	✗	✓



(a) 台式机 **(b) 笔记本电脑** **(c) Jetson Orin**

图 15：准确率保证的 SLO 下，SparseLoom 与基线方法在三种片上系统上的 SLO 违规率。与所有基线方法相比，SparseLoom 始终实现更低的违规率，最多减少 73.6%。



(a) 台式机 **(b) 笔记本电脑** **(c) Jetson Orin**

图 16：延迟保证型 SLO 下，SparseLoom 与基线方法在三种片上系统上的 SLO 违规率。与所有基线方法相比，SparseLoom 始终实现更低的违规率，最多减少 68.2%。