无线神经处理中的多模态深度神经网络工作负载编排

Sai Xu，*IEEE* 会士，加 i-Kit Wong，*IEEE* 会士，Yanan Du，*IEEE* 会员，Hyundong Shin，*IEEE* 会士 *E*

arXiv:2603.02109v1 [eess.SP] 2 Mar 2026

摘要——**在边缘推理中，无线资源分配与加速器级深度神经网络（DNN）调度尚未实现端到端协同优化。无线传输与加速器级DNN执行之间缺乏协调，妨碍了二者的高效重叠，导致端到端推理时延增加。为解决这一问题，本文研究了无线神经处理（WNP）中的多模态深度神经网络工作负载编排。WNP是一种将无线传输与多核加速器执行整合到统一端到端流水线中的范式。首先，我们为多模态深度神经网络执行构建了统一的通信-计算模型，并制定了相应的优化问题。其次，我们提出了 O-WiN 框架，通过两个紧密耦合的阶段——基于仿真的优化和运行时执行——对 WNP 中的深度神经网络工作负载进行编排。第三，我们设计了两种算法：RTFS 和 PACS。RTFS 按顺序调度通信与计算，而 PACS 则将二者交错执行，通过重叠无线数据传输与加速器级DNN执行来实现流水线并行。仿真结果表明，在模态异构性较高时，PACS 的性能显著优于 RTFS；它通过通信与计算重叠更有效地掩蔽无线时延，从而凸显了通信-计算流水线在加速 WNP 中多模态深度神经网络执行方面的有效性。**

索引词—**无线神经处理、流水线并行、深度神经网络、调度、边缘人工智能、边缘推理。**

I. 引言

IMAGINE 这样一种场景中，来自多个分布式传感器节点的各类数据流通过无线方式卸载到边缘服务器。收集后，这些异构数据流被输入单个多模态深度神经网络（DNN）进行推理 [1]。由于端到端推理时延由计算平台上的深度神经网络执行时间和无线传输时间共同决定，因此通信与计算同等关键 [2], [3]；此外，在频谱资源有限的情况下，某些场景中的总体时延往往受无线传输限制（如图6所示）。然而，无线资源分配和加速器级深度神经网络调度通常是孤立设计的。更确切地说，它们仅通过一个覆盖层框架进行耦合和协调，而非通过底层端到端跨域领域协同优化 [2], [4]。受这一观察的启发，本文研究无线神经处理（WNP）这一新范式。该范式将无线传输与多核加速器上的深度神经网络执行相结合，从而实现通信与计算之间的细粒度流水线并行。

WNP旨在将无线通信与神经处理整合到单一执行流水线中，突破将无线通信仅视为外部 I/O 操作的传统观念。在这一统一的执行流程中，数据传输与算子级神经处理紧密交错。这种时间并行性使 WNP 区别于现有的通信-计算协同设计方法 [5],，后者依赖粗粒度资源协作。因此，WNP 有望缓解“无线墙”问题，为 DNN工作负载带来更低的端到端延迟、更高的吞吐量和更高的能效。作为一个新兴研究领域，WNP 的一个基本问题是如何编排 DNN工作负载，使无线数据传输与多核加速器执行在时间上对齐，从而实现无缝的端到端通信–计算集成。

A. 相关工作

这项工作本质上具有跨学科性质，连接了边缘人工智能（AI）推理与加速器级DNN执行。因此，我们将相关文献分为两类，并分别回顾各自的最新进展。

1）边缘人工智能推理：从边缘AI推理的角度来看，深度神经网络推理通常被视为计算即服务（CaaS）抽象下的任务：深度神经网络被建模为可调度工作负载，而对加速器级算子执行和系统约束（例如内存和带宽）的关注有限。在边缘人工智能领域，协同推理已成为一个重要的研究方向。其核心思想是以逐层或逐图的方式划分大型深度神经网络模型，并在终端设备和计算中心上联合执行划分后的片段 [6]。为减少上行通信开销，通常会传输分区点处的中间激活值或特征表示，而非原始输入 [7], [8]。由于上行通信成本通常随传输的特征表示大小而增加，结合特征压缩可以显著降低通信开销，并实现低时延边缘推理 [9], [10], [11], [12]。为进一步缓解设备—边缘协同推理中的通信瓶颈，面向任务的通信 [13], [14], [15], [16]应运而生，它不再追求忠实地传递数据，而是只传输完成任务所需的信息

S. Xu 和 K. K. Wong 任职于英国伦敦 WC1E 7JE 伦敦大学学院电子与电气工程系。K. K. Wong 还任职于韩国京畿道龙仁市 17104 庆熙大学电子工程系。（电子邮件： {sai.xu,kai-kit.wong}@ucl.ac.uk）。Y. Du 任职于英国谢菲尔德 S10 2TN 谢菲尔德大学电气与电子工程学院。（电子邮件：yanan.du@sheffield.ac.uk）。H. Shin 任职于韩国京畿道龙仁市 17104 庆熙大学电子与信息融合工程系（电子邮件：hshin@khu.ac.kr）。C

对应 ding au thor: Kai-Kit W ong.

表 I 主要符号

符号	描述	符号	描述
$\mathcal{G}=(\mathcal{V},\ \mathcal{E})$	多模态DNN的计算有向无环图	$\mathcal{V},\ \mathcal{E}$	$\mathcal{G}$的顶点集和边集
pred(v)、succ(v)	作业 v的前驱集/后继集	$(u\to v)\in\mathcal{E}$	优先约束：v 只能在 u完成后开始
$\mathcal{K}=\{1,\dots,K\}$	模态集合（传感器节点）	$\mathcal{G}_k=(\mathcal{V}_k,\ \mathcal{E}_k)$	模态 k 的子图
$\mathcal{C}=\{1,\dots,C\}$	加速器核心集合	$s_v,\ f_v$	作业 v 的开始/完成时间
$y_{v,c}\in\{0,1\}$	任务到核心的分配指示变量	$p_{v,c}$	作业 v 在核心 c 上的处理时间
$\mathcal{F}_c$	核心 c 上的可行调度	B_{tot}	片上网络带宽预算总量
$b_c(t)$	核心 c 在时间 t的带宽使用量	$\mathcal{T}=\{0,\dots,T-1\}$	无线时隙集合
$\mathcal{F}=\{1,\dots,F\}$	OFDMA 子载波集合	Δ	时隙时长
$x_{k,f,t}\in\{0,1\}$	RB分配：节点 k 在时隙 t使用资源块 f	$\rho_{k,f,t}$	节点 k 在时隙 t上资源块 f 的上行速率
D_k	模态 k的切片大小	$B_k(t;\mathbf{x})$	节点 k 在时隙 t结束时的累计接收比特
$t_k^{\text{arr}}(\mathbf{x})$	切片 k 在边缘服务器的到达时隙索引	$r_k(\mathbf{x})$	切片 k 在边缘服务器的到达（释放）时间
T_{makespan}	端到端推理总时长	t_{start}	RTFS（等待全部）中的计算开始时间
$R_k(t;\mathbf{x})$	时隙 t 开始时切片 k 的剩余负载	$\hat{u}_k(t)$	节点 k 在时隙 t 的预测有效上行速率
$\bar{u}_k(t)\ u_{k,\ f}(t)$	节点 k 在时隙 t 的平均/瞬时速率	$\hat{T}_k(t)$	切片 k 的预测剩余传输时间（RTFS）
$L_{v,c},\quad b_{v,c}$	作业 v 在核心 c上的剖析延迟和带宽需求	$B_{\max}$	每核片上网络带宽上限
$\overline{L}_v$	作业 v 在各核心上的平均延迟	排名$_u$（v）	列表调度中的向上排序值
$\sigma(v)$	仿真器中的作业状态（未调度/运行中/已完成）	$\mathcal{Q}_{\text{ready}}$	就绪作业队列
$v_c,\ \omega_c$	核心 c 上运行的作业及其剩余工作量	α_c	分配给核心 c的带宽份额（事件步）
δ	距下一个作业完成的事件步长	$\mathcal{P}$	轻量级预测器（PACS）
$\hat{A}_k(t)$	切片 k 在时间 t 的预测到达时隙（PACS）	$S_v(t)$	子图 $\mathcal{V}_k$ 的预测器释放时间（PACS）
$\hat{F}_v(t)$	作业 v 的预测最早完成时间（PACS）	o	有向无环图的输出节点
ϵ	小正数常量（数值稳定性）		
$G(k)$	为切片 k 分配一个资源块（RB）所带来的预测完工时间缩减（PACS）		

从而在保持任务性能并提升鲁棒性的同时，降低上行开销和时延。

2）加速器级*DNN*执行：在计算机体系结构领域，近年来一个活跃的研究方向是在多核加速器上高效执行异构DNN工作负载。通常，调度设计空间极其庞大，涵盖算子执行顺序和核心分配决策。为以较低的计算复杂度获得高质量调度，研究人员提出了多种调度策略。在数据中心场景中，AI-MT [17]采用手工设计的层级调度器进行多DNN推理。MAGMA [18] 和 COMB [19] 采用遗传算法加速调度探索，COMB 还进一步引入了内存感知优化。MoCA [20]通过动态内存管理提升多DNN性能。在边缘场景中，Herald [21]和 DREAM [22] 面向实时多DNN执行；Herald 采用异构数据流设计，而 DREAM 则使用自适应在线调度器来应对时变工作负载。DySta [23] 专为稀疏多DNN工作负载设计。TaiChi [24]提出了一种结合图神经网络与强化学习的框架，以在多核加速器上高效调度多DNN工作负载。

以往工作要么将边缘AI推理与深度神经网络执行割裂开来，要么将通信-计算协同设计局限于粗粒度的高层协调。从系统角度看，无线信道在推理过程中本质上承担着数据供给通路的作用。从功能上说，它可以抽象为深度学习加速器的“远程内存”，具有带宽受限、时延较高且动态变化的特性。因此，我们认为，应将无线数据流纳入统一的数据流建模与优化框架，以实现加速器级DNN执行——这一方法对于进一步提升边缘AI推理性能具有巨大潜力。

B. 主要贡献

为弥合无线资源分配与加速器级DNN执行联合优化方面的研究空白，并推动对WNP进行系统探索，本文探讨了一种全面的解决方案，通过统一的通信-计算流水线来编排多模态深度神经网络工作负载。本文的主要贡献总结如下：

- 我们构建了一种适用于 WNP 中多模态深度神经网络执行的系统架构，能够联合考量无线数据传输与多核加速器上的算子级执行。该架构刻画了无线传输与算子级深度神经网络执行之间的内在交互关系。基于这一模型，我们构建了一个优化问题，旨在最小化 WNP 中多模态通信—计算编排的端到端推理时延。

- 我们提出了 O-WiN，这是一种模块化、可扩展的编排框架。它将统一的通信—计算流水线分解为由接口定义的功能模块，以实现解耦集成和系统化的跨模块协调。O-WiN 包含两个紧密耦合的阶段——基于仿真的优化和运行时执行：仿真阶段迭代改进通信策略和计算映射策略，随后将所得方案部署到运行时，以优化无线传输和加速器执行。

- 为解决 WNP 中的端到端推理时延最小化问题，我们提出了两种算法：(i) RTFS，一种顺序通信—计算启发式算法，优先进行上行链路传输，以在推理前尽量缩短无线传输时延；(ii) PACS，一种流水线启发式算法，通过将无线传输与算子执行重叠，在某个模态专属分支所需的最少输入数据

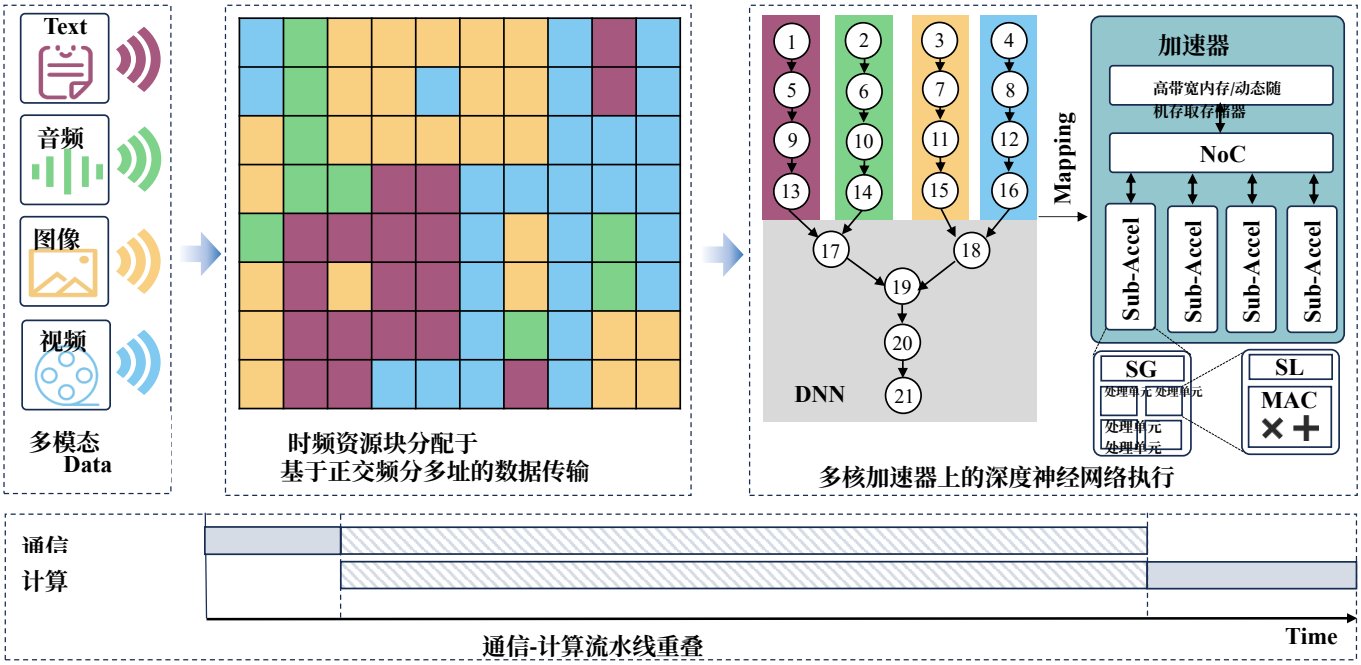


图 1. WNP 中多模态深度神经网络执行的系统架构。

对于该模态y 变成一个 v可用 .

- 我们通过仿真评估 RTFS 和 PACS，并报告端到端执行时间线以及每个核心的片上网络（NoC）带宽分配。我们还通过改变加速器核心数量、正交频分多址（OFDMA）子载波数量、无线时延缩放因子、片上网络带宽预算，以及压缩因子和切片令牌大小，进一步开展敏感性研究。

本文其余部分安排如下。第II节介绍系统模型，并对 WNP 中的多模态通信—计算编排问题进行建模。第III节提出框架 O-WiN，并详细说明各模块的功能及整体工作流程。第IV 节分别介绍两种启发式算法，以实现分阶段执行和流水线并行。第V节报告仿真结果，并讨论不同系统配置下的关键性能趋势。最后，第VI 节总结全文。

II. 系统模型与问题建模

图 1展示了 WNP 中多模态深度神经网络执行的系统架构。在该系统中，多模态深度神经网络的输入通过无线 uplink 信道从传感器节点传输到边缘服务器。数据到达边缘服务器后，便会触发多模态深度神经网络模型的推理，并将其算子映射到多核加速器上执行。为加速推理，可利用通信与计算之间的流水线并行来隐藏无线传输时延。表 I 总结了本文采用的主要符号。

A. 神经网络架构

边缘服务器部署了训练好的多模态深度神经网络，用于处理从多个传感器节点收集的异构数据，包括文本、音频、图像和视频。该模型通常由模态专属编码器组成，这些编码器将每种模态的输入映射为潜在特征表示。随后，通过融合机制（例如拼接后进行投影、基于注意力的融合或门控）整合这些特征，以获得统一表示。然后将统一表示输入任务专属头，生成预测。当不同模态提供互补且对齐良好的信息时，多模态深度神经网络通常比单模态模型具有更高的准确率和鲁棒性。

多模态深度神经网络推理工作负载建模为一个有向无环图（DAG），记为 $\mathcal{G} = (\mathcal{V}, \mathcal{E})$，其中每个顶点 $v \in \mathcal{V}$ 代表一个计算算子，每条有向边 $(u \to v) \in \mathcal{E}$ 表示数据依赖，即算子 v 只有在收到算子 u的输出后才能开始执行。这些算子按模态划分为专属编码器子图，每个子图处理来自一种模态的输入；其他子图则可表示融合模块和任务头。设 $\mathcal{K} = \{1, \ldots, K\}$ 和 $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{E}_k)$分别表示模态索引集，以及与模态 $k \in \mathcal{K}$对应的模态专属子图。

B. 多核加速器

该加速器由多个核心组成，这些核心称为子加速器，并排列成处理单元（PE）阵列。所有核心都通过片上网络访问共享系统带宽，其中系统带宽定义为主存（例如，高带宽内存/动态随机存取存储器）带宽

与主机到加速器（例如，PCIe）带宽两者中的较小值。每个处理单元（PE）都包含一个乘积累加（MAC）单元，用于计算部分和，以及一小块本地暂存器（SL），用于存储权重、激活值和中间结果。在核心级别，共享全局缓存（SG）会预取下一计算分块所需的数据。片上网络将这些预取的操作数分发到各处理单元（PE）的本地暂存器（SL），计算结果则从本地暂存器（SL）写回共享全局缓存（SG）。

为对层级计算如何在多核加速器上执行进行建模，作业和映射的定义如下：（i）作业是能够在底层硬件加速器上独立调度和执行的最小计算任务。根据层的大小和可用硬件资源，作业可以对应一个完整的层，也可以对应层的一部分（由分区得到）。它是系统中调度和计算的基本单位。（ii）映射是在空间和时间两个维度上，对多个核心中的作业分配、调度和优先级进行统筹安排的策略。映射定义了如何在硬件上协调计算与数据移动，以实现高效的资源利用和优化的性能。映射策略可以手动设计，也可以通过优化算法自动生成。

令 $\mathcal{C} = \{1, \ldots, C\}$ 表示加速器核心集合。对于每个作业 $v \in \mathcal{V}$，令 s_v 和 f_v 分别表示其开始时间和完成时间。我们引入二元分配变量$y_{v,\,c} \in \{0,1\}$，其中 $y_{v,\,c} = 1$ 表示作业 v 被分配到核心 $c \in \mathcal{C}$。每个作业必须且只能分配到一个核心：

$$\sum_{c \in \mathcal{C}} y_{v,c} = 1, \qquad \forall v \in \mathcal{V}. \tag{1}$$

如果在核心 c 上执行，作业 v 需要 $p_{v,\,c}$ 的处理时间，并且假设作业是非抢占式的。因此，作业 v 的完成时间为

$$f_v = s_v + \sum_{c \in \mathcal{C}} y_{v,c}\, p_{v,c}, \quad \forall v \in \mathcal{V}. \tag{2}$$

由于有向无环图的优先约束，作业只有在其所有前驱作业完成后才能开始：

$$s_v \geq f_u, \qquad \forall (u,v) \in \mathcal{E}. \tag{3}$$

由于硬件限制，一个核心在同一时间最多只能执行一个作业。为避免引入额外的整数变量和大-M 线性化，我们以可行性形式施加核心容量约束：

$$\{(s_v, f_v) \mid y_{v,c} = 1,\ v \in \mathcal{V}\} \ \in \mathcal{F}_c, \forall c \in \mathcal{C}, \tag{4}$$

其中，$\mathcal{F}_c$ 表示核心 c 上满足析取非重叠约束的可行调度集合。等价地，对于任意两个不同的作业 $u \neq v$，若有 $y_{u,\,c} = y_{v,\,c} = 1$，则 $f_u \leq s_v$ 或 $f_v \leq s_u$ 必须成立。

在多核加速器中，片上网络带宽由各处理器核共享，并可动态重新分配。在任何时间，所有处理器核的带宽使用量总和不得超过可用预算：

$$\sum_{c \in \mathcal{C}} b_c(t) \leq B_{\mathrm{tot}}, \quad \forall t. \tag{5}$$

其中，B_{tot} 和 $b_c(t) \geq 0$ 分别表示总带宽以及时间 t处理器核 $c \in \mathcal{C}$ 的带宽使用量。

C. 无线传输

为支持多节点同时高效接入，采用正交频分多址。在正交频分多址中，可用频谱被划分为一组正交子载波，并沿时间轴进行铺设，形成时频资源块（资源块）网格。由于位置相关的信道增益和空间异质干扰，不同节点在同一资源块上的上行传输速率各不相同；此外，同一节点的速率在不同资源块之间也可能存在显著差异。为在时变信道条件下最大化频谱效率，资源块会在节点之间进行动态分配。

令 $\mathcal{T} = \{0, \ldots, T-1\}$ 和 $\mathcal{F} = \{1, \ldots, F\}$ 分别表示时隙集合和正交子载波集合。切片是由单个模态生成的最小数据单元，且足以启动其对应模态专属子图的前向推理。作为基本处理单元，切片可以包含部分模态数据，也可以包含完整的数据实例，其大小由编码器的输入接口决定。假设每个传感器节点都关联一个唯一模态，并使用 $k \in \mathcal{K}$ 为对应于模态 k 的传感器节点编索引，从而在节点与模态之间建立一一映射。二元变量 $x_{k,\ f,\ t} \in \{0,1\}$ 表示资源块分配决策：若资源块 f 在时隙 t 分配给节点 k，则 $x_{k,\ f,\ t} = 1$；否则为 0。由于每个资源块最多只能分配给一个模态进行上行传输，因此必须满足以下互斥约束：

$$\sum_{k \in \mathcal{K}} x_{k,f,t} \leq 1, \qquad \forall f \in \mathcal{F},\ \forall t \in \mathcal{T}. \tag{6}$$

令 Δ 和 $\rho_{k,f,t}$ 分别表示时隙时长，以及节点 k 在时隙 t中资源块 f 上的可实现上行速率。节点k 在时隙 t期间可通过资源块 f 发送 $\Delta\,\rho_{k,f,t}$ 位。给定资源块分配 $\mathbf{x} = \{x_{k,f,t}\}$，截至时隙 t 结束时，边缘服务器从节点 k 接收到的总比特数为

$$B_k(t; \mathbf{x}) = \sum_{\tau=1}^{t} \sum_{f \in \mathcal{F}} x_{k,f,\tau}\, \Delta\, \rho_{k,f,\tau}. \tag{7}$$

令 D_k 表示节点 k的切片大小。该切片的到达时隙为

$$t_k^{\mathrm{arr}}(\mathbf{x}) = \min\Big\{ t \in \mathcal{T} :\ B_k(t; \mathbf{x}) \geq D_k \Big\}. \tag{8}$$

在边缘服务器处的相应到达时间为

$$r_k(\mathbf{x}) = t_k^{\mathrm{arr}}(\mathbf{x})\, \Delta. \tag{9}$$

由于边缘服务器处的到达时间同时也是其关联的模态专属子图的释放时间，因此 $\mathcal{G}_k = (\mathcal{V}_k,\ \mathcal{E}_k)$ 中的任何算子都无法在 $r_k(\mathbf{x})$ 之前启动。

$$s_v \geq r_k(\mathbf{x}), \qquad \forall k \in \mathcal{K},\ \forall v \in \mathcal{V}_k. \tag{10}$$

D. 并行性

完整的多模态推理过程不仅取决于 DNN层如何映射到硬件加速器上，还取决于输入数据的到达时间。因此，无线传输与深度神经网络执行之间的关键耦合因素是数据到达时间。在多模态DNN中，每个模态专属子图仅依赖于对应的输入数据，一旦数据可用便可立即开始执行。换言之，某个模态的数据一旦到达，其对应的子图便可开始执行，即使其他模态仍在传输中。例如，当子图的计算工作负载较大而输入数据大小相对较小时，执行过程受计算限制，优先传输该模态的数据可能有效降低整体时延。相反，当子图的计算工作负载较小而输入数据大小较大时，推迟传输其数据可能更有效。考虑到这些因素，要增强通信-计算流水线的并行性并避免加速器利用不足，就需要协调好通信-计算资源分配与任务调度。

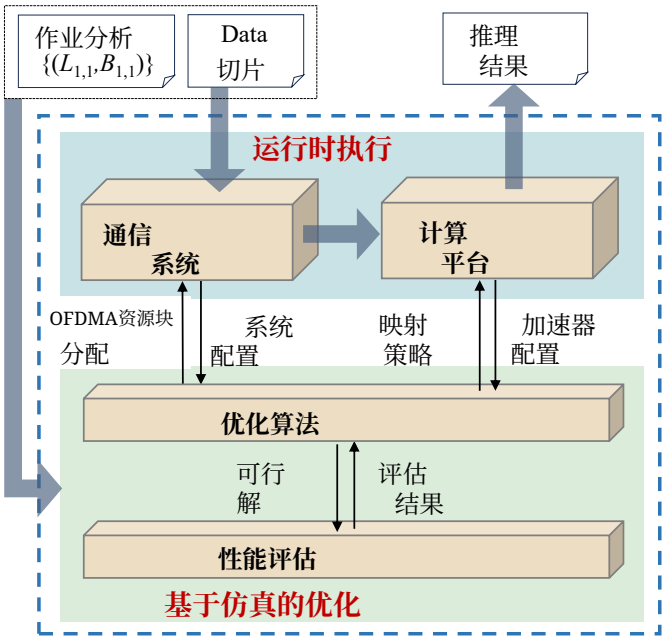


图 2. O-WiN：一种模块化、可扩展的编排框架，将统一的通信-计算流水线分解为接口明确定义的功能模块，以实现解耦集成和系统化的跨模块协调。

E. 问题建模

在推理过程中，通信与计算构成一条统一的执行流水线。我们采用端到端推理时延（即完工时间）作为性能指标，其定义如下：

$$T_{\text{makespan}} = \max_{v\in\mathcal{V}} \ f_v. \tag{11}$$

我们的目标是通过联合优化通信侧资源分配、计算侧资源分配和作业调度，最小化完工时间。该优化问题可表述为：

$$\min_{\mathbf{x},\mathbf{y},\mathbf{s},\mathbf{f}} \quad T_{\text{makespan}} \tag{12}$$

$$\text{s.t.} \quad (1),\ (2),\ (3),\ (4),\ (5),\ (6),\ (10), \tag{13}$$

where $\mathbf{y} = \{y_{v,c}\}_{v\in\mathcal{V},\, c\in\mathcal{C}}$, $\mathbf{s} = \{s_v\}_{v\in\mathcal{V}}$ and $\mathbf{f} = \{f_v\}_{v\in\mathcal{V}}$.

III. 框架

为了在 WNP 中实现多模态深度神经网络工作负载编排，必须建立一种统一且可复用的抽象，覆盖从数据准备、算法开发到系统执行和性能评估的整个流水线。本节介绍 O-WiN，这是一种模块化且可扩展的编排框架，它将端到端工作流分解为定义明确、由接口驱动的功能模块，从而支持各组件的解耦集成、系统化协同和未来扩展。O-WiN 的整体架构如图所示。

2.

A. 架构

O-WiN 由四个相互解耦、即插即用且接口定义明确的模块组成：

- 通信系统。 给定资源块分配决策$\{x_{k,f,t}\}$、每个资源块的上行速率 $\{\rho_{k,f,t}\}$以及切片大小 $\{D_k\}$，该模块确定每个切片何时在边缘服务器上被完整接收，并输出其到达（释放）时间 $\{r_k\}$。此外，它会将接收到的切片负载传递给下游计算平台模块，以便进行后续推理。

- 计算平台。 给定接收到的切片及其释放时间 $\{r_k\}$、多模态推理有向无环图$\mathcal{G} = (\mathcal{V},\ \mathcal{E})$，以及由任务到核心的分配 $\{y_{v,c}\}$和每核心执行调度 $\{\pi_c\}$ with $\pi_c \in \mathcal{F}_c$组成的映射策略，该模块执行推理，在汇点 o处生成输出，并报告由此产生的端到端完工时间 $T_{\text{完工时间}}$。

- 优化算法。 给定切片大小 $\{D_k\}$、每个资源块的上行速率 $\{\rho_{k,f,t}\}$、多模态推理有向无环图 $\mathcal{G} = (\mathcal{V},\ \mathcal{E})$、任务分析表 $\{(L_{v,\,c},\ b_{v,\,c})\}$[1]，以及通信和计算两方面的系统配置，该模块执行通信-计算联合优化，以最小化端到端完工时间 $T_{\text{完工时间}}$。它返回 (i) 资源块分配决策 $\{x_{k,f,t}\}$ 和 (ii) 映射策略，其中包括任务到核心的分配 $\{y_{v,c}\}$和每核心执行调度 $\{\pi_c\}$ with $\pi_c \in \mathcal{F}_c$。

- 性能评估。给定切片大小 $\{D_k\}$、每个资源块的上行速率 $\{\rho_{k,f,t}\}$、多模态推理有向无环图 $\mathcal{G} = (\mathcal{V},\ \mathcal{E})$、任务分析表 $\{(L_{v,\,c},\ b_{v,\,c})\}$、通信和计算两方面的系统配置，以及可行的协同优化解$(\{x_{k,f,t}\},\ \{y_{v,c}\},\ \{\pi_c\})$，该模块将执行完整的

[1]任务分析表 $\{(L_{v,\,c},\ b_{v,\,c})\}$ 可以通过以下方式构建：(i) 在真实硬件上进行实测分析；(ii) 使用周期级/跟踪级仿真器获得测量结果；或 (iii) 合成生成工作负载（例如，从预定义分布中随机采样）。

栈模拟，以模拟切片传输并获取 $\{r_k\}$，在由此产生的释放时间和调度方案下执行有向无环图，并报告端到端性能指标，包括 T_{makespan} 及其他相关统计数据（例如核心利用率和片上网络带宽使用率）。最终输出为最终解$(\{x^*_{k,f,t}\},\ \{y^*_{v,c}\},\ \{\pi^*_c\})$ 及其经验证的评估结果。

B. 工作流程

O-WiN 的端到端工作流由两个紧密耦合的阶段组成：基于仿真的优化和运行时执行。仿真阶段高效地搜索高质量的协同优化解$(\{x^*_{k,f,t}\},\ \{y^*_{v,c}\},\ \{\pi^*_c\})$，随后将其部署以驱动运行时推理流水线。

1) 基于仿真的优化：O-WiN 采用迭代式优化—评估循环，将优化算法模块与性能评估模块紧密耦合。在每次迭代中，优化器生成候选协同优化策略，然后由评估器对其进行端到端评估。评估器返回关键性能统计信息，包括端到端完工时间T_{makespan} 和资源利用率指标。在这些反馈信号的指导下，优化器不断更新$(\{x_{k,f,t}\},\ \{y_{v,c}\},\ \{\pi_c\})$ ，在满足系统约束的同时改进目标，直至获得可行的高质量解。最终生成的策略将被导出至运行时执行阶段，并部署为资源块分配和映射策略，从而闭合优化设计与运行时执行之间的循环。

2) 运行时执行：在运行时，模态切片首先由通信系统模块处理。根据已部署的资源块分配 $\{x^*_{k,f,t}\}$，通信模块确定切片释放时间 $\{r_k\}$，并将收到的切片载荷转发给计算模块。释放后，计算平台根据已部署的映射策略 $(\{y^*_{v,c}\},\ \{\pi^*_c\})$执行相应的模态专属子图。执行在汇点 o处终止，生成最终推理输出以及实际的端到端完工时间 T_{makespan}。

IV. 优化算法

优化问题(P1)将正交频分多址资源块分配与带优先约束的并行机调度结合起来，其中资源块分配决定了在多核加速器上执行深度神经网络的释放时间。鉴于(P1)属于NP难问题，且具有极其庞大的组合设计空间，我们提出了两种启发式优化算法[2]：（i）释放时间优先调度（*RTFS*），一种分阶段的先传输后计算启发式算法，在推理开始前优先进行上行链路传输，以尽量缩短释放时间；以及（ii）流水线感知协同调度（*PACS*），一种目标驱动启发式算法，通过交错安排通信与计算来尽量缩短完工时间。

[2]在 O-WiN 中，优化模块被视为由接口定义的黑盒组件，因此可以灵活集成不同的调度策略；任何改进的算法都可以替换我们的启发式算法，且只需极少的集成工作，无须修改其他模块。

A. RTFS算法

在RTFS中，通信与计算被解耦为两个顺序执行的阶段，并由全等待屏障隔开：只有在所有切片均已接收至边缘服务器后，计算才会开始。在第一阶段，通过资源块分配将所有切片传输至边缘服务器。在第二阶段，将作业映射到多核加速器上执行DNN工作负载，同时满足有向无环图优先约束和片上网络带宽约束。

1) 通信阶段

由于多核加速器上的深度神经网络执行要等到所有数据切片到达后才开始，因此计算开始时间由最晚的切片到达时间决定：

$$t_{\text{start}} = \max_{k\in\mathcal{K}}\ r_k(\mathbf{x}) = \Delta\cdot\max_{k\in\mathcal{K}} t_k^{\text{arr}}(\mathbf{x}). \tag{14}$$

为降低全等待时延，采用尾部导向目标，优先处理最慢的切片。定义切片 k 在时隙 t 开始时的剩余负载为

$$R_k(t;\mathbf{x}) = \max\{0,\ D_k - B_k(t-1;\mathbf{x})\}. \tag{15}$$

切片 k 的剩余传输时间为

$$\hat{T}_k(t) = \frac{R_k(t;\mathbf{x})}{\hat{u}_k(t)}, \tag{16}$$

其中，$\hat{}u_k(t)$ 表示节点 k 在时隙 t期间的预测有效上行传输速率。

根据未来信道统计信息，分两步估算预测有效上行传输速率 $\hat{}u_k(t)$。首先，对所有子载波上的瞬时上行传输速率取平均，得到时隙级平均速率：

$$\bar{u}_k(\tau) = \frac{1}{F}\sum_{f=1}^{F} u_{k,f}(\tau), \tag{17}$$

其中 $u_{k,f}(\tau)$ 表示节点 k 在时隙 τ期间于子载波 f 上的瞬时上行传输速率。接下来，将时隙 t 的预测有效上行传输速率定义为剩余时域内的后缀均值：

$$\hat{u}_k(t) = \frac{1}{T-t}\sum_{\tau=t}^{T-1}\bar{u}_k(\tau). \tag{18}$$

为保证数值稳定性，我们采用一个较小的下限 $\epsilon > 0$，使 $\hat{}u_k(t) \leftarrow \max\{\hat{}u_k(t)\ \epsilon\}$。在每个时隙 t内，以贪心方式依次分配子载波，以近似最小化最大尾部时间 $\max_{k\in\mathcal{K}}\hat{}T_k(t)$。对于每个子载波，通过以下方式执行一步更新：

$$R_k \leftarrow \max\{0,\ R_k - \Delta\,\rho_{k,f,t}\}. \tag{19}$$

随后，重新计算 $\max_{j\in\mathcal{K}}\hat{}T_j(t)$ ，并选择使最终最大尾部时间最小的节点。RTFS 的通信阶段总结见算法 1。

2）计算阶段

在计算阶段，到达 t_{start} 处的全体等待同步点后，所有依赖切片且特定于模态的子图将同时释放，使边缘服务器能够立即开始推理。此外，

算法1RTFS通信阶段

输入： $\mathcal{K}$, $\mathcal{F}$, Δ, $\{D_k\}_{k\in\mathcal{K}}$, $\{\rho_{k,f,t}\}$, $\{\hat{u}_k(t)\}$, $\epsilon > 0$
输出： $\{x_{k,f,t}\}$, $\{t_k^{\mathrm{arr}}\}$, t_{start}

$R_k \leftarrow D_k$, $t_k^{\mathrm{arr}} \leftarrow +\infty$, $\forall k \in \mathcal{K}$
$t \leftarrow 0$
当 $\exists k \in \mathcal{K} : R_k > 0$ **时执行**
  $\tilde{R}_k \leftarrow R_k$, $\forall k \in \mathcal{K}$
  $x_{k,f,t} \leftarrow 0$, $\forall k \in \mathcal{K}$, $f \in \mathcal{F}$
  对 $f \in \mathcal{F}$ **执行**
    $k^\star \leftarrow \perp$, 最佳 $\leftarrow +\infty$
    对所有 $k \in \mathcal{K}$ **执行**
      if $\tilde{R}_k \le 0$ **则**
        继续
      $b \leftarrow \Delta\rho_{k,f,t}$
      $\tilde{R}'_j \leftarrow \tilde{R}_j$, $\forall j \in \mathcal{K}$
      $\tilde{R}'_k \leftarrow$最大值$\{0,\ \tilde{R}_k - b\}$
      $\hat{T}_j \leftarrow \tilde{R}'_j /$ 最大值$\{\hat{u}_j(t),\ \epsilon\}$, $\forall j \in \mathcal{K}$
      对象 $\leftarrow$ 最大值$_{j\in\mathcal{K}}$ $\hat{T}_j$
      如果 对象 $<$ 最佳 **则**
        最佳 $\leftarrow$对象, $k^\star \leftarrow k$
    如果 $k^\star \neq \perp$ **则**
      $x_{k^\star,f,t} \leftarrow 1$
      $\tilde{R}_{k^\star} \leftarrow$最大$\{0,\ R_{k^\star} - \Delta\,\rho_{k^\star,f,t}\}$
  对于 $f \in \mathcal{F}$ **执行**
    寻找 k，使得$x_{k,f,t} = 1$（若无，则继续）
    $R_k \leftarrow$ 最大$\{0,\ R_k - \Delta\,\rho_{k,f,t}\}$
  对于所有 $k \in \mathcal{K}$ **执行**
    如果 $R_k = 0$ **和** $t_k^{\mathrm{arr}} = +\infty$，**则**
      $t_k^{\mathrm{arr}} \leftarrow t$
  $t \leftarrow t + 1$
$t_{\mathrm{start}} \leftarrow \Delta \cdot$ 最大值$_{k\in\mathcal{K}}\, t_k^{\mathrm{arr}}$
返回 $\{x_{k,f,t}\}$, $\{t_k^{\mathrm{arr}}\}$, t_{start}

在共享片上网络带宽预算 $B_{\max}$ 下，多核加速器上的深度神经网络执行被建模为事件驱动仿真器。

只有当某个作业的所有前驱作业都已完成时，该作业才会就绪。在每个决策点，都会按照类似于异构最早完成时间（HEFT）的基于秩的列表调度策略，将就绪作业分派给空闲核心。$v \in \mathcal{V}$ 的向上排序值定义为

$$\mathrm{rank}_u(v) \triangleq \overline{L}_v + \max_{k\in\mathrm{succ}(v)} \mathrm{rank}_u(k), \tag{20}$$

其中，$\overline{L}_v = \frac{1}{C}\sum_{c=1}^{C} L_{v,c}$ 表示作业 v 在所有处理器核上的平均执行延迟，succ(v) 表示有向无环图中 v 的后继集合；对于没有后继作业的出口作业，最大化项定义为零。直观而言，$\mathrm{rank}_u(v)$ 可作为从作业 v 开始的剩余关键路径代价的近似指标，因此可为在线调度提供优先级度量。在运行时，每当某个处理器核空闲时，调度器便构建就绪集，其中包含前驱作业均已完成且所需输入均已就绪的作业。随后，调度器从就绪集中选择向上排序值 $\mathrm{rank}_u(\cdot)$ 最大的作业，并将其分派到空闲处理器核上，进行非抢占式执行直至完成；若出现并列，可采用次级标准进行打破，例如选择较小的$\overline{L}_v$。

仿真器直接将时间推进到下一个完成事件，为所有运行中的作业更新剩余工作量，标记已完成的作业，并释放新就绪的后继作业。给定有向无环图 $\mathcal{G} = (\mathcal{V}, \mathcal{E})$，作业将在线分派到 C 个加速器核心上；当某个作业的所有前驱作业都已完成后，该作业便符合执行条件。此外，执行还会受到片上网络带宽争用的影响：当多个核心并发运行时，其作业会竞争共享带宽预算，从而影响作业进度及每个作业的完成时间。具体而言，每个作业都有一个带宽需求 $b_{v,c} \in (0, 1]$。平台提供全局共享资源预算 $B_{\max}$。当多个核心并发执行作业时，分配给核心 c上运行作业的实际带宽与其请求的 $b_{v_c,c}$成正比：

$$\alpha_c(t) = \begin{cases} \dfrac{b_{v_c,c}}{\sum_{k\in\mathcal{R}(t)} b_{v_k,k}} \cdot B_{\max}, & \mathcal{R}(t) \neq \varnothing, \\ 0, & \text{otherwise}, \end{cases} \tag{21}$$

其中，$\mathcal{R}(t)$ 是时间 t 正在运行作业的处理器核集合，而 v_k 表示当前分配给处理器核 k 的作业。

*B. PACS*算法

在 RTFS 中，等待全部作业的屏障常常会使计算资源处于闲置状态，从而可能延长端到端完工时间。相比之下，PACS 移除了这一屏障，并在分片到达时立即释放符合条件的依赖分片的作业，从而实现流水线式通信—计算重叠，并更有效地利用跨模态并发。

1）通信阶段

与 RTFS 不同，PACS 根据每个候选资源块分配决策对多核加速器上深度神经网络执行的下游影响进行评估，将无线资源分配与加速器侧执行明确耦合。然而，对于在线决策而言，通过对通信-计算流水线进行完整的重新仿真来估算每个候选方案的准确完工时间，代价过高。因此，采用轻量级预测器 $\mathcal{P}$ 近似预测最终汇节点的完成时间。预测器不会运行完整的多核执行仿真，而是将通信引发的分片到达与由离线分析得到的算子平均延迟参数化的粗粒度执行模型相结合。具体而言，为降低预测开销，预测器有意忽略处理器核竞争和排队效应。

在时隙 t，预测器 $\mathcal{P}$ 估算切片 k 的到达时间为

$$\hat{A}_k(t) = \begin{cases} t, & R_k(t) \le 0, \\ t + \dfrac{R_k(t)}{\hat{u}_k(t)}, & R_k(t) > 0. \end{cases} \tag{22}$$

相应入口作业的释放时间设为

$$S_v(t) = \max\{t, \hat{A}_k(t)\}, \qquad \forall v \in \mathcal{V}_k. \tag{23}$$

算法2RTFS计算阶段

输入： $\mathcal{V}, L_{v,c}$ 和$b_{v,c}$ 对于 $c \in \mathcal{C}$, $B_{\max}$, $\epsilon > 0$, t_{start}; op(v), 切片(v), 以及 $\{g_k\}_{k=1}^{K}$。
输出： $\{s_v\}_{v\in\mathcal{V}}$, $\{f_v\}_{v\in\mathcal{V}}$

对 $v \in \mathcal{V}$ **执行**
  $\overline{L}_v \leftarrow \frac{1}{C}\sum_{c=1}^{C} L_{v,c}$
对于所有 $v \in \mathcal{V}$，按逆拓扑序 **执行**
  如果 $\text{succ}(v) = \varnothing$，**则**
    $\text{rank}_u(v) \leftarrow \overline{L}_v$
  else
    排名$_u(v) \leftarrow \overline{L_v}$ + 最大值$_{k\in\text{succ}(v)}$排名$_u(k)$

(1) 初始化
$t_{\text{cur}} \leftarrow t_{\text{start}}$
$\sigma(v) \leftarrow 0, \quad \forall v \in \mathcal{V}$
$s_v \leftarrow \perp,\ f_v \leftarrow \perp,\ \forall v$
$\mathcal{Q}_{\text{ready}} \leftarrow \varnothing$
对于所有 $v \in \mathcal{V}$ **执行**
  如果pred(v) $=\varnothing$ **和** $\big(\text{op}(v) \neq \texttt{Embed}$ **或** $t_{\text{cur}} + \epsilon \geq g_{\text{slice}(v)}\big)$ **然后**
    $\sigma(v) \leftarrow 1$
    $\mathcal{Q}_{\text{ready}} \leftarrow \mathcal{Q}_{\text{ready}} \cup \{v\}$
$v_c \leftarrow$null $\omega_c \leftarrow 0,\ \forall c \in \mathcal{C}$
当 $\exists v \in \mathcal{V}:\ \sigma(v) \neq 3$ **执行**
(2) 将就绪作业调度到空闲核心
  $\mathcal{I} \leftarrow \{c:\ v_c =$空$\}$
  当 $\mathcal{I} \neq \varnothing$ **且** $\mathcal{Q}_{\text{ready}} \neq \varnothing$ **时执行**
    选择任意 $c^\star \in \mathcal{I}$ $\left(-\text{排名}_u(v) \quad \underline{L_v}\right)$
    $v^\star \leftarrow$ 参数 $\min_{v\in\mathcal{Q}_{\text{就绪}}}$
    如果 $s_{v^\star} = \perp$ **那么**
      $s_{v^\star} \leftarrow t_{\text{cur}}$
    $v_{c^\star} \leftarrow v^\star$
    $\omega_{c^\star} \leftarrow L_{v^\star,c^\star} \cdot b_{v^\star,c^\star}$
    $\sigma(j^\star) \leftarrow 2$
    $\mathcal{Q}_{\text{ready}} \leftarrow \mathcal{Q}_{\text{ready}} \setminus \{v^\star\}$
    $\mathcal{I} \leftarrow \mathcal{I} \setminus \{c^\star\}$
(3) 带宽共享下的事件驱动进展
  $\mathcal{C}_{\text{run}} \leftarrow \{c:\ v_c \neq$ 空$\}$
  $Z \leftarrow \sum_{c\in\mathcal{C}_{\text{run}}} b_{v_c c}$
  如果 $Z > \epsilon$，**则**
    对于所有 $c \in \mathcal{C}_{\text{run}}$ **do**
      $\alpha_c \leftarrow \frac{b_{v_c c}}{Z} \cdot B_{\max}$
  else
    对于所有 $c \in \mathcal{C}_{\text{run}}$ **do**
      $\alpha_c \leftarrow \frac{B_{\max}}{|\mathcal{C}_{\text{运行}}|}$
  $\delta \leftarrow \min_{c\in\mathcal{C}_{\text{运行}}}$ 分钟 $\omega_c/(\alpha_c + \epsilon)$
  $t_{\text{cur}} \leftarrow t_{\text{cur}} + \delta$
  对所有 $c \in \mathcal{C}_{\text{un}}$ **do**
    $\omega_c \leftarrow \omega_c - \alpha_c \cdot \delta$
(4) 完成作业并更新就绪集
  对所有 $c \in \mathcal{C}_{\text{run}}$ **执行**
    如果 $\omega_c \leq \epsilon$ **则**
      $v \leftarrow v_c$
      $f_v \leftarrow t_{\text{cur}}$
      $\sigma(v) \leftarrow 3$
      $v_c \leftarrow$null $\omega_c \leftarrow 0$
  对所有 $v \in \mathcal{V}$ **执行**
    如果 $\sigma(v) = 0$ **且** pred(v) $\subseteq \{k \in \mathcal{V}:\ \sigma(k) = 3\}$ **且** $\big(\text{op}(v) \neq \texttt{Embed}$ **或** $t_{\text{cur}} + \epsilon \geq g_{\text{slice}(v)}\big)$ **则**
      $\sigma(v) \leftarrow 1$
      $\mathcal{Q}_{\text{ready}} \leftarrow \mathcal{Q}_{\text{ready}} \cup \{v\}$
返回 $\{s_v\}_{v\in\mathcal{V}}, \quad \{f_v\}_{v\in\mathcal{V}}$

为了捕捉带宽变化下的执行速率

共享时，我们将执行作业 v 在核心 c上的工作量 定义为

$$W_{v,c} = L_{v,c} \cdot b_{v,c}, \tag{24}$$

其中 $L_{v,\ c}$ 是作业 v 在核心 c上的基础延迟。在执行过程中，核心c 上运行中的作业的剩余工作量以 $\alpha_c(t)$的速率递减；随着并发运行的作业集合随时间变化，这自然会产生分段进度。RTFS计算阶段总结于算法 2。

给定这些释放时间约束，通过对有向无环图执行极大加动态规划，预测作业 v 的最早完成时间，计算式如下：

$$\hat{F}_v(t) = \begin{cases} S_v(t) + \bar{L}_v, & \text{pred}(v) = \varnothing, \\ \bar{L}_v + \max\limits_{v'\in\text{pred}(v)} \hat{F}_{v'}(t), & \text{pred}(v) \neq \varnothing. \end{cases} \tag{25}$$

该递归自然能够处理任意数量的分支和同步点：任何多输入算子都只是一个满足 $|\text{pred}(v)| > 1$ 的作业，其开始时间由最晚的前驱完成时间决定。根据当前剩余有效载荷预测的总完成时间为

$$\hat{T}_{\text{makespan}}(t) = \hat{F}_o(t). \tag{26}$$

完成总完成时间估计后，在每个时隙内贪心地分配子载波：将每个子载波分配给能最大幅度降低预测输出完成时间的数据切片。对于尚未完成的数据切片候选项 k，其 $R_k(t;\mathbf{x}) > 0$，通过仅将切片 k 减少子载波 f可传输的数量，构造试探性剩余向量 $R^{(k)}$，其表达式为

$$R^{(k)}_{k'} \leftarrow \begin{cases} \max\{0,\ R_k(t) - \Delta\,\rho_{k,f,t}\}, & k' = k, \\ R_{k'}(t), & k' \neq k. \end{cases} \tag{27}$$

然后，相应的边际降幅为

$$G(k) = \hat{T}_{\text{makespan}}(t; R_k) - \hat{T}_{\text{makespan}}(t; R^{(k)}). \tag{28}$$

基于此，子载波 f 被分配给切片 $k^\star = \text{argmax}_k\, G(k)$，之后将剩余向量更新为$R_k \leftarrow R^{(k^\star)}$，并继续处理子载波 f+1。 ˆ通过直接最大化 T_{makespan} 的降幅，这种贪心分配会优先选择对即将到来的同步点以及通往输出的端到端关键路径最为关键的切片。PACS 的通信阶段总结于算法 3。

2) 计算阶段

计算调度器采用与 RTFS 相同的基于秩的列表调度策略，将优先级最高的就绪作业分派到空闲核心上。不同之处在于，就绪集会随通信过程共同变化：分片一到达，其门控时间 A_k 一到， соответств应的切片门控作业便可进入就绪集，从而使计算能够与正在进行的传输重叠。分片到达会触发输入门控：切片 k 的第一个作业只有在 $t \geq A_k$时才能就绪，而所有其他作业均遵循标准的有向无环图优先约束。系统以事件驱动方式推进，直至以下事件中最早发生者：(i) 下一个计算完成事件

算法3PACS 通信阶段

输入： $\mathcal{K}$, $\mathcal{F}$, Δ, $\{D_k\}_{k\in\mathcal{K}}$, $\{\rho_{k,f,t}\}$, $\{\hat{u}_k(t)\}$, $\mathcal{P}$, $\epsilon>0$
输出： $\{x_{k,f,t}\}$, $\{t_k^{\mathrm{arr}}\}$

$R_k \leftarrow D_k,\ t_k^{\mathrm{arr}} \leftarrow +\infty,\ \forall k\in\mathcal{K}$
$t \leftarrow 0$
当 $\exists k\in\mathcal{K}:\ R_k>0$ **时**
  $\tilde{R}_k \leftarrow R_k,\ \forall k\in\mathcal{K}$
  $x_{k,f,t} \leftarrow 0,\ \forall k\in\mathcal{K}, f\in\mathcal{F}$
  $\hat{T}_{\mathrm{base}} \leftarrow \mathcal{P}\big(t, \tilde{R}, \hat{u}(t)\big)$
  对于 $f\in\mathcal{F}$，**执行**
    $k^\star \leftarrow \perp,\quad G_{\mathrm{best}} \leftarrow -\infty$
    对于所有 $k\in\mathcal{K}$，**执行**
      if $\tilde{R}_k \le 0$，**则**
        继续
      $b \leftarrow \Delta\cdot\rho_{k,f,t}$
      $\tilde{R}'_j \leftarrow \tilde{R}_j,\ \forall j\in\mathcal{K}$
      $\tilde{R}'_k \leftarrow$最大值$\{0,\ \tilde{R}_k - b\}$
      $\hat{T}_{\mathrm{try}} \leftarrow \mathcal{P}\big(t, \tilde{R}', \hat{u}(t)\big)$
      $\hat{G} \leftarrow \hat{T}_{\mathrm{base}} - \hat{T}_{\mathrm{try}}$
      若 $G > G_{\mathrm{best}}$ **则**
        $G_{\mathrm{best}} \leftarrow G,\quad k^\star \leftarrow k$
    若 $k^\star \neq \perp$ **则**
      $x_{k^\star,f,t} \leftarrow 1$
      $\tilde{R}_{k^\star} \leftarrow$最大值$\{0,\ R_{k^\star} - \Delta\cdot\rho_{k^\star,f,t}\}$
      $\hat{T}_{\mathrm{base}} \leftarrow \mathcal{P}\big(t, \tilde{R}, \hat{u}(t)\big)$
  对于 $f\in\mathcal{F}$，**执行**
    查找满足 k 使得 $x_{k,f,t}=1$ 的值（如果不存在，则继续）
    $R_k \leftarrow$最大值$\{0,\ R_k - \Delta\cdot\rho_{k,f,t}\}$
  对所有 $k\in\mathcal{K}$ **执行**
    如果 $R_k=0$ **和** $t_k^{\mathrm{arr}}=+\infty$，**则**
      $t_k^{\mathrm{arr}} \leftarrow t$
  $t \leftarrow t+1$
返回 $\{x_{k,f,t}\}, \{t_k^{\mathrm{arr}}\}$

算法4 RTFS计算阶段

输入： $\mathcal{V}$, $L_{v,c}$ 和$b_{v,c}$ 对于 $c\in\mathcal{C}$, $B_{\max}$, $\epsilon>0$, t_{start}; op(v), 切片(v);$\{t_k^{\mathrm{arr}}\}_{k=1}^{K}$。
输出： $\{s_v\}_{v\in\mathcal{V}}$, $\{f_v\}_{v\in\mathcal{V}}$

1：算法2，第1–7行
(1) 初始化
2: $t_{\mathrm{cur}} \leftarrow t_{\mathrm{start}}$
3: $g_k \leftarrow t_k^{\mathrm{arr}},\quad \forall k$
4: 算法2，第9–16行
5: **当** $\exists v\in\mathcal{V}:\ \sigma(v)\neq 3$ **执行**
(2) 将就绪作业调度至空闲核心
6: 算法2，第18–28行
(3) 带宽共享下的事件驱动进展
7: $\mathcal{C}_{\mathrm{run}} \leftarrow \{c : v_c \neq$空值$\}$
8: **if** $\mathcal{C}_{\mathrm{run}}=\varnothing$ **then**
9: $t_{\mathrm{comp}} \leftarrow +\infty$
10: **else**
11: $Z \leftarrow \sum_{c\in\mathcal{C}_{\text{运行}}} b_{v_c,\,c}$
12: **如果** $Z>\epsilon$ **那么**
13: **对于所有** $c\in\mathcal{C}_{\mathrm{run}}$ **do**
14: $\alpha_c \leftarrow \frac{b_{v_c,c}}{Z}\cdot B_{\max}$
15: **else**
16: **对于所有** $c\in\mathcal{C}_{\mathrm{run}}$ **do**
17: $\alpha_c \leftarrow \frac{B_{\max}}{|\mathcal{C}_{\text{运行}}|}$
18: $\delta_{\mathrm{comp}} \leftarrow \min_{c\in\mathcal{C}_{\text{运行}}} \omega_c/(\alpha_c+\epsilon)$
19: $t_{\mathrm{comp}} \leftarrow t_{\mathrm{cur}} + \delta_{\mathrm{comp}}$
20: $t_{\mathrm{gate}} \leftarrow \min\{g_k :\ g_k > t_{\mathrm{cur}}+\epsilon\}$
21: **如果**不存在这样的 k **则**
22: $t_{\mathrm{gate}} \leftarrow +\infty$
23: $t_{\mathrm{next}} \leftarrow \min\{t_{\mathrm{comp}},\ t_{\mathrm{gate}}\}$
24: **如果** $t_{\mathrm{next}}=+\infty$ **则**
25: **跳出**
26: $\delta \leftarrow t_{\mathrm{next}} - t_{\mathrm{cur}}$
27: **如果** $\delta>0$ **且** $\mathcal{C}_{\mathrm{run}}\neq\varnothing$ **则**
28: **对于所有** $c\in\mathcal{C}_{\mathrm{run}}$ **do**
29: $\omega_c \leftarrow \omega_c - \alpha_c\cdot\delta$
30: $t_{\mathrm{cur}} \leftarrow t_{\mathrm{next}}$
(4) 完成作业并更新就绪集
31: 算法2，第41–50行
32: **返回** $\{s_v\}_{v\in\mathcal{V}},\ \{f_v\}_{v\in\mathcal{V}}$

和 (ii) 下一个通信时隙边界，然后相应地更新就绪集和调度决策。PACS 的计算阶段总结于算法4。

C. 复杂度分析

设 $V \triangleq |\mathcal{V}|$ 和 $E \triangleq |\mathcal{E}|$ 分别表示多模态有向无环图中的作业数和优先约束边数。对于 RTFS 和 PACS，向上秩 $\{\mathrm{rank}_u(v)\}$ 通过逆拓扑动态规划（DP）在 $\mathcal{O}(V+E)$ 中预先计算一次。

1) RTFS

在每个时隙中，RTFS 使用贪心搜索，将每个 F 子载波分配给 K 个切片之一；该搜索针对每个子载波评估至多 K 个候选项，并更新至多 K 个切片的预测剩余时间，因此每个时隙的开销为 $\mathcal{O}(FK^2)$，在 T个时隙内的总开销为 $\mathcal{O}(TFK^2)$。所有切片到达后，RTFS 会运行事件驱动列表调度器：每个作业只调度和完成一次；每次作业完成都会触发对至多 C 个活动核心的带宽重新分配，并更新就绪集。使用以秩 u 为键的优先队列时，调度开销为 $\mathcal{O}(V\log V)$，事件处理的开销为 $\mathcal{O}(VC)$，因此计算的开销为 $\mathcal{O}(V\log V+VC)$（不包括秩的预计算）。

2) PACS

PACS 使用轻量级预测器 $\mathcal{P}$，在有向无环图上通过极大加动态规划计算输出完成时间；给定释放时间后，$\mathcal{P}$ 的运行时间为 $\mathcal{O}(V+E)$。每个时隙中，PACS 通过枚举最多 K 个候选切片，并调用 $\mathcal{P}$ 评估边际收益，为每个子载波分配资源，因此每个时隙的复杂度为 $\mathcal{O}(FK(V+E))$，总体复杂度为 $\mathcal{O}(TFK(V+E))$。PACS 采用与 RTFS 相同的事件驱动列表调度器，并额外根据切片到达时间对就绪集进行门控，因此计算阶段的复杂度仍为 $\mathcal{O}(V\log V+VC)$。

V. 评估

本节开发了一种事件驱动模拟器，用于协同建模无线传输与具有优先约束的多核加速器执行，从而实现端到端评估-

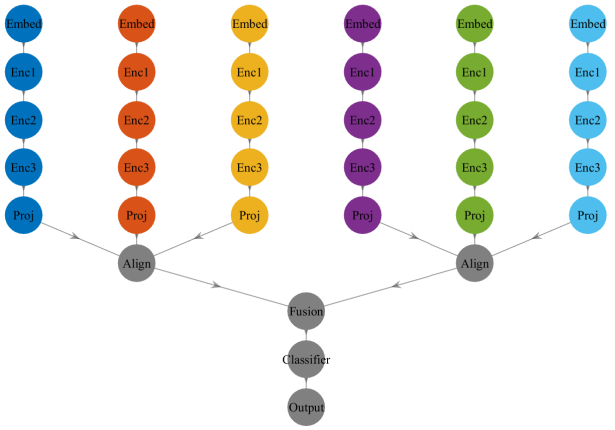


图3. 多模态DNN工作负载的网络架构。

WNP中的多模态深度神经网络工作负载[3]。

A. 仿真设置

对于多模态DNN工作负载，我们构建了一个两级有向无环图，以同时表示模态并行执行和跨模态同步，如图3所示。具体而言，输入被划分为 K 个模态。对于每个模态k，一个模态专属子图由五个依次执行的算子组成：Embed$_k$ → Enc1$_k$ → Enc2$_k$ → Enc3$_k$ →Proj$_k$。随后，跨模态子图对各模态的输出进行同步和聚合。具体来说，两个对齐节点分别等待来自互不重叠的模态子集的结果：Align$_1$,Align$_2$ → Fusion → Classifier → Output，其中 Align$_1$ 取决于 $\{\text{Proj}_k\}_{k=0}^{\lfloor K/2\rfloor-1}$，Align$_2$ 取决于 $\{\text{Proj}_k\}_{k=\lfloor K/2\rfloor}^{K-1}$。这种扇出—扇入结构引入了多输入同步点，并且对拖尾效应敏感，因此揭示了通信与计算流水线化的机会，同时使端到端完工时间由传输完成情况和计算调度共同决定。

对于无线传输，每个时隙、每个子载波的可达速率 $\rho_{k,f,t}$ 在一定范围内生成独立同分布（i.i.d.）样本，以模拟时变信道，并使用后缀均值预测器估计未来的有效速率。在计算方面，每个作业 $v \in \mathcal{V}$ 具有核心相关时延 $L_{v,c}$ 以及带宽需求 $b_{v,c}$二者均根据切片令牌大小和算子类型合成生成。多核加速器实施共享的片上网络带宽预算 $B_{\max}$；在运行时，带宽根据各运行中处理器核的需求按比例动态分配，并根据作业剩余的带宽工作量，以事件驱动方式推进每个作业。表 II 总结了实验中使用的关键参数。

令 N_k 表示每个切片的令牌预算，它可作为通信和计算的统一缩放因子。切片 k 的通信载荷为

$$D_k = N_k\, D_{\mathrm{f}}\, b\, \kappa_k\, \eta, \tag{29}$$

[3]这些合成输入可以无缝替换为实际测量数据。此外，仿真能够在缓解设备和环境特定的采样偏差的同时，对更广泛的参数空间进行可控探索，从而提升通用性和可复现性。

表 II 仿真参数

参数	符号	值 / 范围
节点/模态数量	K	6
切片令牌大小	—	$[512, 128, 256, 128, 192, 128]$
有效载荷模型	—	D=512，半精度浮点数（2 字节），开销 1.05
压缩因子	—	$[0.25, 0.25, 0.80, 0.60, 0.60, 0.60]$
切片大小	D_k	[134.4, 33.6, 215.0, 80.6, 121.0, 80.6] 千字节
加速器核心	C	4
片上网络带宽预算	$B_{\max}$	1.0（归一化）
通信步骤	Δ	1 ms
每时隙子载波数	F	16
速率范围（每个子载波）	$\rho_{k,f,t}$	1–10 兆比特每秒（1000–10000 千比特每秒）
通信轨迹长度	T	8000 个时隙
核心速度抖动	—	± 0.08
时延缩放	—	$\times 6$

其中， D_{f} 为特征维度， b为每个元素的字节数，κ_k 为特定切片的压缩因子， η 为协议开销。对于作业 v，令牌尺度为

$$S(v) = \begin{cases} N_k, & \text{if } v \text{ belongs to slice } k, \\ \sum_{k=1}^{K} N_k, & \text{if } v \text{ is cross–modality,} \end{cases} \tag{30}$$

每核作业时延由一个考虑异质性和抖动的算子专属模型生成：

$$\ell_{v,c} = g_{\mathrm{op}(v)}(S(v)) \cdot \frac{\xi_{v,c}}{v_c} \cdot \Delta. \tag{31}$$

此处，$g_{\mathrm{op}(v)}(\cdot)$表示依赖于算子的基线时延模型（例如，嵌入/投影采用线性模型，编码器块则包含二次项），$\xi_{v,c} \in [0.9, 1.1]$ 是用于刻画运行间变异性的乘性抖动因子，v_c 表示处理单元 c的相对速度，用于建模异构计算，其中 $v_c = \max(1 + u_c,\ 0.7)$，且 $u_c \sim \mathcal{U}(-0.08,\ 0.08)$。因此，增大 N_k 会同时增加切片有效载荷 D_k 以及所有工作负载依赖于切片 k的作业的执行时间。

B. 端到端推理时延

图 4 比较了 RTFS 和 PACS 的端到端执行时间线。在 RTFS 中，计算与通信通过全等待屏障分隔：在模态切片到达之前，加速器无法开始执行入口 `Embed` 作业。因此，无线尾时延会完全表现为计算端的空闲区间，并几乎直接增加到端到端完工时间中。相比之下，PACS 消除了屏障，并以切片门控方式激活模态专属子图。一旦某个切片可用，其编码器任务便会被释放，并可与其余模态的持续上行传输并行运行。这种通信与计算重叠显著提高了核心利用率、减少了空闲间隔，并使末端 输出 节点更早完成。此外，PACS 根据预测的输出 完成时间的边际缩减来分配资源块，从而优先处理位于（或门控）跨模态同步路径上的切片（例如，对齐和融合），进而缩短端到端完工时间。

图。5 进一步从多核加速器上的片上网络带宽共享角度对比了 RTFS 和 PACS。图中的柱状条表示各核心瞬时获得的份额

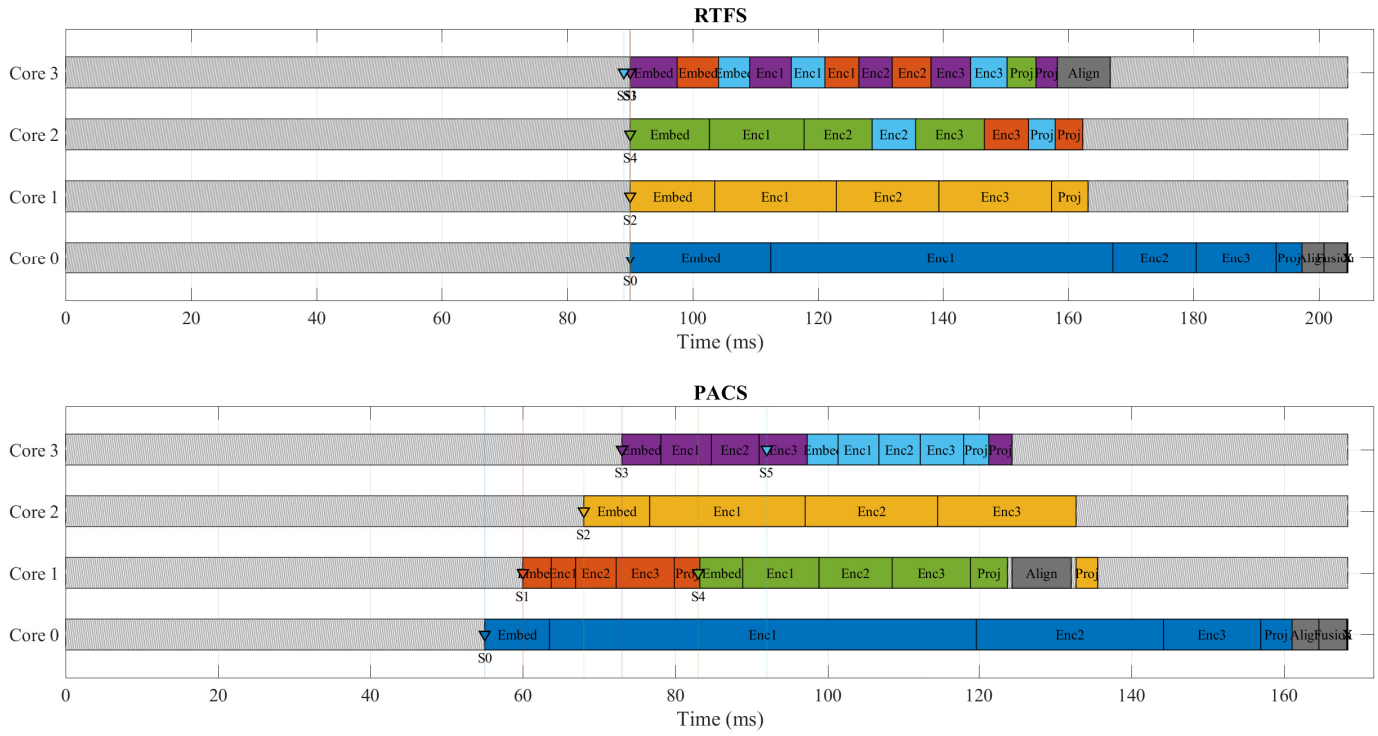


图 4. RTFS 和 PACS 的端到端执行时间线。

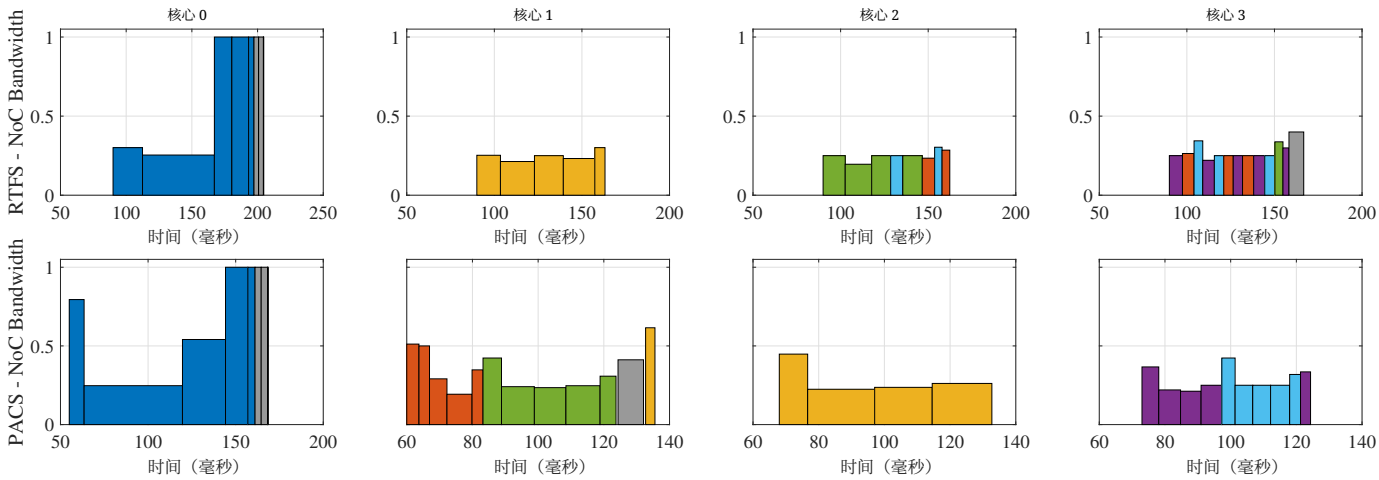


图 5. RTFS 和 PACS 多核加速器上的片上网络带宽共享。

占归一化片上网络预算的比例，颜色表示特定切片的作业。与 RTFS 相比，PACS 的带宽利用率更加持续，长时间低活跃度区间更少，这与图4中观察到的空闲间隔减少以及通信与计算重叠改善相一致。此外，由于不同模态的切片以交错方式到达，片上网络带宽争用得以缓解，从而进一步提高了计算吞吐量和整体执行效率。

图 6展示了 RTFS 和 PACS 在四个系统扩展维度下的端到端完工时间细分：(a) 变化加速器核心的数量，(b) 改变 OFDMA 子载波的数量，(c) 改变无线时延缩放因子，以及 (d) 改变片上网络带宽预算。对于 RTFS，完工时间通过堆叠条形图明确分解为通信和计算组件，这反映了其等待所有执行完成的机制，该机制在很大程度上使这两个阶段串行化。相比之下，PACS 支持切片门控执行，并使计算与正在进行的传输重叠。在这些子图中，当子载波数量（即通信资源）发生变化时，性能趋势最为显著；而计算相关参数的变化只会引起

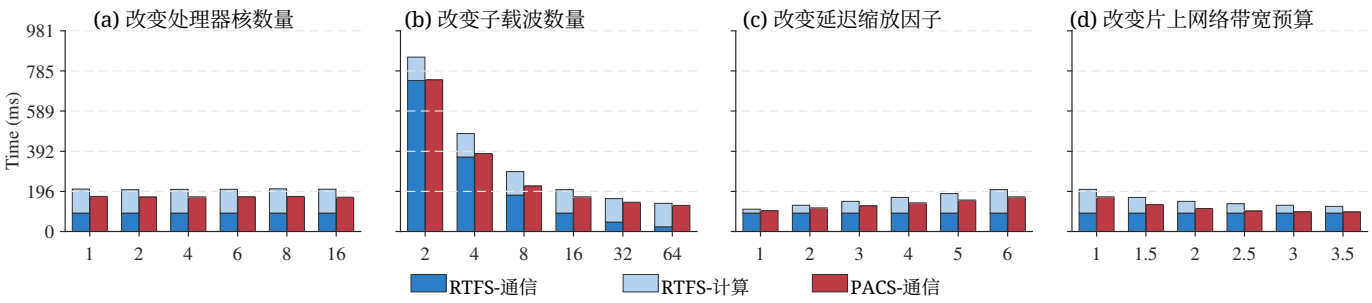


图 6. 四个系统扩展维度对 RTFS 和 PACS 端到端完工时间的影响——(a) 加速器核心数量，(b) OFDMA 子载波数量，(c) 无线时延缩放因子，以及 (d) 片上网络带宽预算。

表 III 不同令牌长度和压缩设置下的端到端完工时间 T_{makespan} (毫秒)。

切片令牌大小	压缩因子 [0.5, 0.5, 0.50, 0.50, 0.50, 0.5]		压缩因子 [0.25, 0.25, 0.50, 0.50,0.50, 0.5]		压缩因子 [0.25, 0.25, 0.80, 0.60,0.60, 0.6]		压缩因子 [0.25, 0.25, 0.50, 0.50,0.80, 0.8]		压缩因子 [0.5, 0.5, 0.50, 0.50, 0.80, 0.8]	
	RTFS PACS	**Gain**	**RTFS PACS**	**Gain**	**RTFS PACS**	**Gain**	**RTFS PACS**	**Gain**	**RTFS PACS**	**Gain**
[128, 128, 128, 128, 128, 128]	113.92 113.92	0.00%	105.92 104.92	+0.94%	114.92 116.92	-1.74%	126.92 126.94	-0.02%	134.92 132.43	+1.85%
[64, 64, 128, 128, 256, 256]	154.87 143.83	+7.13%	150.87 141.47	+6.23%	168.87 161.41	+4.42%	191.87 181.47	+5.42%	196.87 184.59	+6.24%
[64, 128, 256, 128, 192, 128]	139.15 126.34	+9.21%	132.15 132.15	0.00%	145.15 136.53	+5.94%	157.15 148.54	+5.48%	168.15 155.52	+7.51%
[512, 128, 256, 128, 192, 128]	205.60 166.39	+19.08%	189.60 153.39	+19.10%	204.60 168.39	+17.70%	217.60 180.39	+17.10%	231.60 191.39	+17.36%
[512, 128, 256, 384, 192, 64]	230.33 193.66	+15.92%	213.33 179.66	+15.78%	230.33 192.66	+16.36%	236.33 199.66	+15.52%	236.33 199.66	+15.52%

轻微变化。这种对比表明，整体完工时间主要受通信限制。

表III报告了不同令牌长度和压缩设置下的结果。数据表明，当不同模态的令牌长度和压缩比相对均衡，即通信和计算负载相当时，RTFS 相较于 PACS 的优势并不明显。在某些情况下，PACS 的表现甚至略逊于 RTFS。相比之下，当不同模态的令牌长度和压缩比差异显著，导致通信和计算需求高度不均衡时，PACS 相较于 RTFS 则表现出明显优势。总的来说，这些结果表明，当模态异构性较高时，PACS 能更有效地以计算过程掩蔽无线时延。

VI. 结论

本文研究了 WNP 中多模态深度神经网络工作负载的编排，旨在最小化统一通信—计算流水线的完工时间。为实现无线传输与深度神经网络执行的联合优化，我们提出了模块化、可扩展的框架 O-WiN，并设计了两种启发式算法：RTFS 和 PACS。仿真结果表明，利用通信与计算之间流水线并行的 PACS，与顺序执行的 RTFS 启发式算法相比，显著缩短了端到端完工时间。此外，在模态异构性较高时，PACS 带来的增益更大，因为尽早释放并执行模态特定分支能够更好地掩蔽无线尾时延。

参考文献

[1] P. Xu、X. Zhu和D. A. Clifton，“基于变换器的多模态学习：综述”，*IEEE*模式分析与机器智能汇刊，第45卷，第10期，第12113–12132页，2023年10月。[2] D. Wen等，“面向多设备边缘人工智能的面向任务感知、计算与通信集成”，*IEEE*无线通信汇刊，第23卷，第3期，第2486–2502页，2024年3月。[3] J. Yi、G. Lee、M. Jeong、S. Shin、D. Kim和Y. Lee，“在App-RAN相互感知下实现移动边缘计算实时视频分析的端到端时延保障”，见第*23*届移动系统、应用与服务国际年会论文集（*MobiSys'25*），美国加利福尼亚州阿纳海姆，2025年6月，第402–416页。[4] L. Kundu、X. Lin、R. Gadiyar、J.-F. Shi、L. Lacasse和S. Chowdhury，“AI-RAN：利用人工智能驱动的计算基础设施变革无线接入网”，*IEEE*通信杂志，第64卷，第1期，第168–174页，2026年1月。[5] Y. Shi、K. Yang、T. Jiang、J. Zhang和K. B. Letaief，“通信高效的边缘人工智能：算法与系统”，*IEEE*通信综述与教程，第22卷，第4期，第2167–2191页，2020年第4季度。[6] E. Li、L. Zeng、Z. Zhou和X. Chen，“边缘人工智能：通过边缘计算按需加速深度神经网络推理”，*IEEE*无线通信汇刊，第19卷，第1期，第447–457页，2020年1月。[7] K. Yang、Y. Shi、W. Yu和Z. Ding，“面向边缘推理的能效处理与鲁棒无线协作传输”，*IEEE*物联网期刊，第7卷，第10期，第9456–9470页，2020年10月。[8] J. Li、W. Liang、Y. Li、Z. Xu、X. Jia和S. Guo，“通过探索深度神经网络模型划分与推理并行，实现边缘计算中时延感知深度神经网络推理的吞吐量最大化”，*IEEE*移动计算汇刊，第22卷，第5期，第3017–3030页，2023年5月。[9] J. Shao、Y. Mao和J. Zhang，“面向边缘推理学习面向任务的通信：一种信息瓶颈方法”，*IEEE*选定领域通信期刊，第40卷，第1期，第197–211页，2022年1月。[10] J. Shao和J. Zhang，“Bottlenet++：设备-边缘协同推理系统中特征压缩的端到端方法”，见*IEEE*国际通信会议研讨会论文集（*ICC Workshops*），爱尔兰都柏林，2020年6月，第1–6页。[11] W. Xiao、Y. Hao、J. Liang、L. Hu、S. A. AlQahtani和M. Chen，“面向边缘视觉计算的自适应压缩卸载与资源分配”，*IEEE*认知通信与网络汇刊，第10卷，第6期，第2357–2369页，2024年12月。[12] A. Furutanpey、P. Raith和S. Dustdar，“FrankenSplit：采用浅层变分瓶颈注入的高效神经特征压缩，面向

移动边缘计算”，*IEEE*移动计算汇刊，第23卷，第12期，第10770–10786页，2024年12月。[13] J. Shao、Y. Mao和J. Zhang，“面向多设备协同边缘推理的面向任务通信”，*IEEE*无线通信汇刊，第22卷，第1期，第73–87页，2023年1月。[14] S. Xie、S. Ma、M. Ding、Y. Shi、M. Tang和Y. Wu，“面向采用数字调制的面向任务通信的鲁棒信息瓶颈”，*IEEE*选定领域通信期刊，第41卷，第8期，第2577–2591页，2023年8月。[15] H. Li、J. Shao、H. He、S. Song、J. Zhang和K. B. Letaief，“利用信息瓶颈应对面向任务通信中的分布偏移”，*IEEE*选定领域通信期刊，第43卷，第7期，第2667–2683页，2025年7月。[16] S. Xie、H. He、S. Song、J. Zhang和K. B. Letaief，“迈向实时边缘人工智能：结合视觉特征对齐的模型无关面向任务通信”，*IEEE*选定领域通信期刊，第43卷，第12期，第4262–4276页，2025年12月。[17] E. Baek、D. Kwon和J. Kim，“一种多神经网络加速架构”，见*ACM/IEEE*第*47*届计算机体系结构国际年会论文集（*ISCA*），2020年，第940–953页。[18] S.-C. Kao和T. Krishna，“Magma：在多个加速器核心上映射多个深度神经网络的优化框架”，见*IEEE*高性能计算机体系结构国际研讨会论文集（*HPCA*），2022年，第814–830页。[19] S. Zheng、S. Chen和Y. Liang，“将深度神经网络协同映射到复杂异构片上系统的内存与计算”，见*ACM/IEEE*第*60*届设计自动化会议论文集（*DAC*），2023年，第1–6页。[20] S. Kim、H. Genc、V. V. Nikiforov、K. Asanovi´c、B. Nikoli´c和Y. S. Shao，“MoCA：面向多租户深度神经网络的以内存为中心的自适应执行”，见*IEEE*高性能计算机体系结构国际研讨会论文集（*HPCA*），2023年，第828–841页。[21] H. Kwon、L. Lai、M. Pellauer、T. Krishna、Y.-H. Chen和V. Chandra，“面向多深度神经网络工作负载的异构数据流加速器”，见*IEEE*高性能计算机体系结构国际研讨会论文集（*HPCA*），2021年，第71–83页。[22] S. Kim、H. Kwon、J. Song、J. Jo、Y.-H. Chen、L. Lai和V. Chandra，“DREAM：面向动态实时多模型机器学习工作负载的动态调度器”，见第*28*届*ACM*编程语言与操作系统体系结构支持国际会议论文集（*ASPLOS*），第4卷，2023年，第73–86页。[23] H. Fan、S. I. Venieris、A. Kouris和N. Lane，“Sparse-DySta：面向稀疏多DNN工作负载的稀疏性感知动态与静态调度”，见第*56*届*IEEE/ACM*微体系结构国际年会论文集（*MICRO*），2023年，第353–366页。[24] X. Zhou等，“TaiChi：基于图调度的多深度神经网络高效执行”，见欧洲设计、自动化与测试会议暨展览会论文集（*DATE*），法国里昂，2025年，第1–7页。