arXiv:2302.01568v1 [cs.LG] 3 Feb 2023

DynaMIX：多任务系统中基于深度神经网络的实时应用资源优化

Minkyoung Cho 和 Kang G. Shin美国密歇根大学，美国密歇根州安娜堡{minkycho, kgshin}@umich.edu

摘要——**随着深度神经网络（DNNs）逐渐展现出其重要性和可行性，越来越多的基于深度神经网络的应用（例如物体检测和分类）已在自动驾驶汽车（AVs）上开发并部署。为了满足不断增长的期望和需求，自动驾驶汽车应在满足时间要求（尤其是安全要求）的同时，为多个并发车载应用“优化”有限车载计算资源的使用。也就是说，自动驾驶汽车实时应用应与其他并发应用共享有限的车载资源，同时不得错过由生成并向应用提供输入图像的摄像头帧率所规定的截止期限。然而，现有的深度神经网络解决方案大多（即使不是全部）由于计算成本高昂，专注于提升特定硬件的并发性，而不会根据运行中的应用数量，动态优化／修改深度神经网络应用的资源需求。为缓解这一局限，我们提出了DynaMIX（Dyna**mic **MIX**ed-precision model construction，动态混合精度模型构建），该方法可优化并发应用的资源需求，并旨在最大化执行准确率。为实现实时资源优化，我们利用应用性能配置文件构建了一个优化问题，同时考虑每个应用的准确率和最坏情况延迟。我们还提出了动态模型重配置，通过在运行时仅延迟加载选定的层来减少加载整个模型的开销。DynaMIX在多任务系统中根据约束满足情况和推理准确率进行了评估，并与最先进的解决方案进行了比较，结果表明，在各种环境／运行条件下，该方法具有有效性和可行性。

I. 引言

随着自动驾驶汽车（AV）中使用的深度神经网络（DNN）应用不断增多，确保这些应用及时、高质量地执行至关重要。特斯拉 [1]和丰田 [2],等主要汽车制造商正在扩大深度神经网络在基于摄像头的视觉应用中的使用范围。在自动驾驶汽车中，摄像头以每秒 10–40 帧（FPS）的速率持续监测环境状况 [3], [4]。由于当前图像帧的处理结果必须在下一帧到达时，所有使用摄像头拍摄的图像帧的车载应用都必须满足由摄像头帧率决定的相同截止期限。然而，并发视觉应用之间的干扰可能导致其中一些应用错过截止期限。图1展示了一个示例场景：两个应用处理来自同一摄像头的图像帧。在这种情况下，这两个应用有着相同 的截止期限，因为它们既不能产生过时数据，也不能干扰下一批图像帧的处理。如果分类应用的执行先于实时目标检测（RTOD）应用，

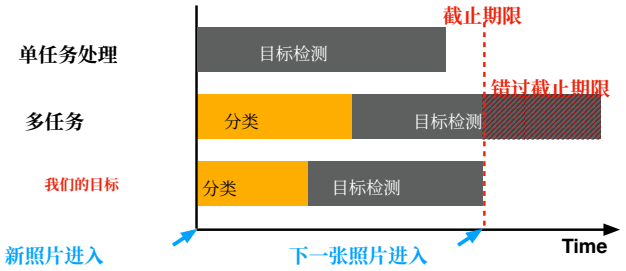


图1. DynaMIX 的示例用例及其目标

目标检测应用的端到端（即从摄像头到检测输出）延迟会超过其截止期限。

然而，如何确保自动驾驶汽车等嵌入式控制应用按时完成任务，仍是一个难题。当并发应用的数量动态变化时，为满足其截止期限，所需资源（如 CPU 和内存空间）也应相应调整。遗憾的是，大多数视觉应用都需要大量计算，因此，最先进的多任务系统 [3], [12]–[14] 专注于在特定处理器上并发执行多个深度神经网络应用，却不会动态调整其资源需求。减少每个应用所需资源最直接的方法是压缩深度神经网络模型。考虑到准确率与计算量之间的权衡，以单一精度统一压缩模型中的所有层并不高效。例如，以高精度量化模型可能导致错过截止期限。相反，低精度量化会降低准确率 [15],，进而可能造成致命事故。这种权衡要求设计一种方案，在考虑总准确率 （定义为并发应用准确率的加权和）及其截止期限的同时，将每个深度神经网络模型优化为混合精度表示。

本文提出 DynaMIX（动态混合精度模型构建），它会根据运行中应用程序的状态，动态调整每个深度神经网络模型的位宽设置，以优化其计算和内存资源。要实现 DynaMIX，必须解决两个难题，才能使混合精度模型适用于自动驾驶汽车等时间关键型系统。首先，我们必须在运行时确定每个深度神经网络模型的最佳压缩程度（即最佳位配置）。在运行时平衡延迟与准确率十分困难，因为当运行中

表 I 与先前工作的比较。“多任务支持”表示运行不受特定深度神经网络模型限制的各种应用程序

	多任务支持	实时保证	不绑定特定处理器
A. 资源管理 [5]–[7]	✓		✓
B. RTOD 执行 [3], [8]–[10]		✓	✓
C. 基于硬件的调度 [11]–[14]	✓	✓	
DynaMIX	✓	✓	✓

的应用程序状态发生变化时，两者都要求针对每个应用运行的压缩模型数量，执行相应数量的深度神经网络推理，因此会带来很高的在线开销。为解决这一难题，我们基于测量数据，通过回归建立应用性能配置文件，根据可用内存大小预测最坏情况延迟和最可能的推理准确率。随后利用这些配置文件，对多个应用进行在线资源优化。

其次，我们需要降低模型加载开销。当最优位设置发生变化时，除非所有层的位宽都发生变化，否则加载整个混合精度模型的效率并不高。然而，将由一个深度神经网络模型生成的所有可能的混合精度模型加载到内存中，既不高效，也不可行。为降低模型切换开销，我们提出一种基于延迟加载概念的运行时模型重配置方法；延迟加载已广泛应用于网络服务和操作系统等其他领域[16], [17]。所提出的逐层延迟加载有助于避免每次最优位设置改变时都加载整个模型，从而高效利用平台资源。

DynaMIX由离线阶段和在线阶段组成。在在线阶段，它将资源优化建模为一个约束非线性优化问题。该优化的约束不仅包括截止期限，还包括离线生成的配置文件，以确定满足截止期限和可用内存空间约束、使总准确率最大化的最优解（即为每个应用分配的内存空间）。当新的摄像头图像进入系统，且检测到并发应用的状态发生变化时，主进程会调整分配给并发应用的内存空间大小。为每个深度神经网络应用确定调整内存空间后的新位配置后，该应用会通过加载必要的高精度层来重新配置其执行路径，从而优化计算资源用量。这样，所有应用都能在不牺牲总准确率的情况下满足截止期限。

我们评估了 DynaMIX 在运行多个并发应用时的表现。我们的研究结果有助于一个或多个并发应用满足截止期限，同时优化资源分配。结果表明，即使并发应用数量增加或时间约束变得严格，每个应用所采用的模型准确率也只会损失 $< 2.2\%$。此外，我们从多个方面证明了 DynaMIX 资源分配的可行性，并展示了其处理多任务场景的有效性。我们还识别出 DynaMIX 无法支持的情况，以及总准确率过低、以致应用在实际中无法接受的情况。

总之，本文的贡献如下：

- 首个面向并发车载应用的资源优化方案，无需绑定特定处理器即可满足其截止期限；
- 利用应用性能配置文件构建优化问题，在实时兼顾多个应用的准确率和最坏情况延迟的同时，求得其最优压缩程度；
- 当应用的最优位设置发生变化时，仅延迟加载选定的层并进行运行时模型重配置，从而降低整个模型的加载开销；
- 支持快速部署现有深度神经网络应用，无需进行任何（重新）训练，以避免运行时自适应量化造成显著的准确率下降。

II. 相关工作

嵌入式平台上关于多个深度神经网络推理或实时深度神经网络推理的既有研究，主要分为三个方向：资源管理、实时目标检测优化以及基于硬件的调度（表 I）。

A。多种视觉应用的资源管理 *Apps*

资源管理在确保多个视觉应用持续运行方面发挥着关键作用。大多数资源管理方案通过将部分或全部计算卸载到云服务器或其他边缘设备，降低深度神经网络流水线执行的过高成本。MCDNN [5], 作为一种具有代表性的资源管理方案，提出了一种运行时调度器，用于在边缘设备和云服务器上处理深度神经网络模型（或其片段）。尽管卸载减轻了边缘系统的计算负载，但由于云端可访问性不可预测或可能发生隐私泄露，应用质量会因此变得不稳定。

为缓解或避免这些问题， [6], [7] 提出了设备端资源管理方案。DeepMon [6]旨在通过优化移动 GPU 上的卷积神经网络（CNN），确保视觉应用持续运行。它通过缓存复用中间结果，加速了卷积计算。NestDNN[7] 提出了基于滤波器剪枝的资源管理方案。然而，DeepMon 和 NestDNN 均仅支持非实时任务，这可能是因为最常用的实时任务（例如目标检测或跟踪）需要大量计算，而边缘系统无法及时完成这些计算（例如，DeepMon 仅达到 1∼2帧每秒）。

DynaMIX 的灵感来自 [6], [7]，因为两者都采用无服务器方法。DynaMIX 通过压缩深度神经网络降低视觉应用的复杂度，并利用应用性能方面的知识找出最优模型。

1) 为何不改造 *NestDNN [7]* 以保证满足截止期限？：即使修改 NestDNN 的调度器以满足实时约束条件，也无法同时保证准确率和及时性。NestDNN 为每个应用使用数量较少且固定（即五个）的剪枝模型，因此需要牺牲准确率来生成不同大小的模型。如果为了提高准确率而生成更多剪枝模型，由于为并发深度神经网络应用寻找最优模型集的开销很高，便难以保证及时执行。为解决这一问题，需要利用应用性能配置文件构建优化问题，并像我们在本文中提出的那样生成查找表。相比之下，DynaMIX 完全支持实时场景。

B. 嵌入式系统上的 *RTOD*执行

RTOD 在提升安全性和便利性方面潜力巨大，因此已有大量研究致力于在嵌入式系统（尤其是自动驾驶汽车）上运行 RTOD。尽管目标检测近期取得了显著进展 [18]–[20]，，但由于需要对摄像头图像流中捕获的多个物体进行定位和分类，计算开销巨大，RTOD 仍然是嵌入式实时系统中的瓶颈。

AdaVP [8]构建了一个并行的目标检测与跟踪流水线，以并行运行两个应用程序；当检测到新目标时，会在运行时调整深度神经网络设置（即输入帧大小），以提高跟踪准确率。Yang 等人 [3] 处理了使用私有摄像头视频流和相同深度神经网络模型的并发视觉应用；通过共享基例架构并使用多线程处理多个摄像头图像，提高了吞吐量。DNN-SAM [10] 支持使用一个实时目标检测模型进行多次推理，以处理不同摄像头生成的图像。它根据关键性级别将原始图像拆分为不同部分，并通过缩放调整每个部分的大小，从而确保满足截止期限。

[9] 的作者提出了一种新的实时目标检测系统，该系统可以根据动态截止期限 改变执行路径——实时目标检测任务的截止期限会随底层驾驶环境而变化，例如跳过某些层，或选择不同大小的子网络之一。此外，他们利用逐层延迟（针对所有层）和动态截止期限确定实时目标检测的执行路径，但在路径选择决策中没有考虑其他并发应用造成的中断。DynaMIX受到 [9] 的启发，其执行路径可以动态改变，以适应运行环境条件。当每个应用可用的资源量发生变化时，DynaMIX会重新配置深度神经网络执行流水线。

总之，先前的工作通过模型自适应或多线程加速实时目标检测任务，以满足其显式或隐式截止期限。然而，多线程/应用程序造成的资源争用可能导致性能下降，而实时模型自适应则可能造成显著的准确率下降。更重要的是，在实时多任务平台上应用时，实时目标检测任务对硬件资源的充分利用需求可能会带来问题。

1) 为什么不能将 *Heo* 等人 *[9]* 的方法修改为适用于多神经网络执行？： [9] 的作者展示了特定层的最坏执行时间模型能够用于多租户系统，但他们既没有说明如何在并发深度神经网络模型之间共享有限资源，也没有考虑应用程序之间上下文切换的开销。相比之下，DynaMIX 解决了多任务系统中的这些问题。

C. 多 *DNN* 推理的基于硬件的调度

AI-MT [11]为多任务平台提出了一种新的神经网络加速器架构和调度方案。 [12], [13] 通过使用专用深度神经网络加速器（如现场可编程门阵列、神经网络处理器），协调多个延迟关键型深度神经网络任务。在异构处理器之间调度作业，使其能够实时执行。 [14]的最新研究提出了一种 GPU 调度方法，用于运行多个实时应用。基本上，该方法在异构或多个处理器之间调度内存受限型和计算受限型作业，以提高并发性。然而，它不适用于图1所示的典型情况。它只关注使用未经修改的深度神经网络模型来调度多个实时作业，而这些模型需要固定的资源量，因此限制了它能够处理的并发作业数量。相比之下，DynaMIX 会动态调整每个应用的资源消耗。此外，DynaMIX 与这些已有解决方案相互正交，因此可以与它们协同运行。

III. 背景：混合精度量化

对激活值和模型参数进行量化，可以通过降低底层模型的计算复杂度来加速深度神经网络执行。这一方法基于以下事实：在大多数计算平台上，整数运算在向量化计算中的吞吐量远高于浮点运算。然而，在经过充分训练（即已收敛至最低损失）的模型中移除部分位，会导致全精度模型与量化模型之间出现输出扰动，从而显著降低准确率。

混合精度量化通过采用具有不同位宽的层 [21]–[25]，被视为解决这一问题的有效方案。对于对量化更敏感的层，采用更高的位宽有助于保留其原始数值，从而减少模型的输出扰动和准确率下降。典型的混合精度量化算法由层敏感度测量和层位宽决策组成。

A. 层敏感度测量

层敏感度表示量化某一层时模型输出变化的程度。对于最全面的混合精度方法，具有 L 层、采用 B 种位宽的模型会产生 B^L 个量化模型。由于深度神经网络模型近年来不断加深，混合精度将不再那么有吸引力。因此，识别层敏感度是缩小比特分配庞大设计空间的一种高效方法

。经过准确测量的层敏感度还可用于计算所得混合精度模型的总体扰动，并且是寻找最佳位设置的良好指标。

层敏感度会受到多个因素的影响，例如层位置、操作类型、与其他层的连接关系以及层参数规模。在大型模型中，考虑所有这些因素来定义敏感度十分困难；研究人员通过计算*L2*范数 或*KL*散度 [22], [25]，将输出扰动用作敏感度指标。

本文采用当前最先进的混合精度量化方法 ZeroQ [22],中定义的敏感度指标，该指标基于全精度模型与量化模型之间的*KL*散度，定义如下：

$$S_i(k) = \frac{1}{N}\sum_{j=1}^{N} \mathrm{KL}\left(\mathcal{N}(x_j),\ \widetilde{\mathcal{N}}_i^k(x_j)\right)$$

其中， $S_i(k)$表示量化模型 $\widetilde{\mathcal{N}}_i^k(\cdot)$的敏感度，该模型的第 i 层被量化为 k位； $\mathcal{N}(\cdot)$表示全精度模型； x 是用于测量敏感度的一小组输入图像，大小为 N 。

B. 硬件感知比特宽度决策

混合精度量化具有进行硬件感知模型压缩的灵活性。作为最基本的比特宽度决策方法，基于规则的方案依赖对深度神经网络模型和硬件架构的了解以及人工投入。然而，模型日益复杂，使得这类启发式方法难以应用。ZeroQ [22] 采用基于帕累托前沿的方法，寻找输出扰动最小的最优压缩模型。每个混合精度模型的总体敏感度通过对模型中所有层的敏感度求和计算得出。尽管作者在这一过程中没有体现层间依赖，但他们指出，这种敏感度计算的计算开销较低，并且在经验上取得了良好效果。此外，ZeroQ还将敏感度和模型大小作为准确率和延迟的间接指标。

然而，每个处理器都有各自不同的架构设计，这意味着最佳位设置会因硬件 [21] 而异。例如，为提高可复用性而设计的权重参数布局可以有效降低传统卷积层的延迟，因此其位宽可以高于全连接层或深度卷积层。由此，通过软硬件协同设计，这种先进的量化可以

降低计算和资源开销，同时相较于硬件无关的深度神经网络压缩方法（例如，均匀量化 [15]），不会导致推理准确率严重下降。

得益于其灵活性，混合精度能够加速网络模型，同时仅造成可忽略的准确率下降。因此，近期推出的处理引擎都支持可变位宽的混合精度运算 [26],[27]。这些最新进展提升了基于混合精度的方法在深度神经网络加速中的重要性。

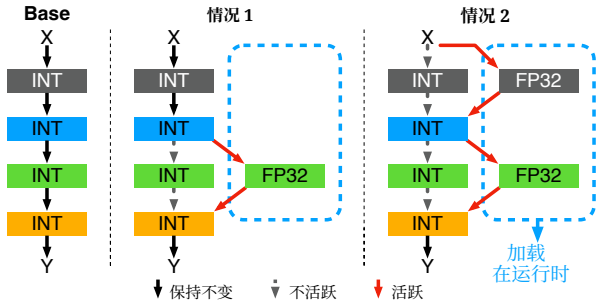


图 2. 运行时模型重配置与执行机制。

受基于强化学习的位宽设置方法 [21]，启发，我们以硬件感知的方式为每个应用确定最佳混合精度模型。我们不使用浮点运算次数、模型大小和敏感度等间接指标，而是采用在仿真环境中测得的最坏情况延迟（以满足截止期限）、峰值内存使用量（以考虑内存容量）和推理准确率。我们的目标是采用混合精度方法寻找最优方案，在充分利用有限资源的同时最大限度地提高总准确率。第IV节和第V节将详细介绍 DynaMIX 如何构建混合精度模型，以及如何为每个基于深度神经网络的应用选择最优位配置。

IV. 动态模型重配置与执行

DynaMIX 的核心是根据当前系统状态，为每个应用找到最合适的混合精度模型；否则，应用进程可能因深度神经网络模型准确率较低而不稳定，或严重浪费平台资源。因此，我们需要考虑具有不同位配置的各种压缩模型，以及保存和加载多个模型的问题。然而，每次都重新加载整个模型可能不适用于实时应用；当只需加载少数层时，这种做法也缺乏效率。因此，我们提出在运行时通过延迟加载浮点层来重新配置模型的比特宽度，以降低模型加载开销并确保满足截止期限。这种动态模型重配置方法受到 [28], [29] 的启发；在该方法中，比特宽度决策仅针对单个 深度神经网络模型作出，且运行时自适应量化需要从头训练一个专用深度神经网络。相比之下，我们提出了一种无需任何额外（重新）训练即可确定多个深度神经网络位配置的方案。

在 DynaMIX 中，当确定每个应用程序的最优位设置后，仅加载目标模型所需的附加层来组成执行路径（表 II 中的 7 ）。在运行时执行该操作之前，我们将一个模型拆分为多个层，并存储所有层的全精度（FP）版本和量化（INT）版本（1 ）。也就是说，对于由 L 个层组成的模型，将保存 $2L$ 个层版本。具体而言，对于浮点层，会保存操作类型（例如卷积）和全精度权重；对于整数层，则会将量化全精度激活值所需的附加参数与操作类型和整数权重一同保存。

表 II 系统组件导航。PIC 指相应组件的负责流程。

阶段	组件	章节	PIC
离线	① 按精度保存层	IV	Sub
	② 混合精度模型构建	V-A	Sub
	③ 配置文件生成	V-B	Main
	④ 查找表生成	V-D	Main
Online	⑤ 内存资源分配	V-C	Main
	⑥ Lookup Table Querying	V-D	Main
	⑦ Model Reconfig. and Execution	IV	Sub

机制。 模型重配置包括：1）加载必要的层；2）更改模型执行路径。这里我们处理的是一种多任务系统，其中主进程负责进程监控和内存分配，子进程负责模型执行。为了仅加载运行时所需的层，DynaMIX 首先应通过加载所有可能应用程序的整数层，并在不更改原始架构的情况下构建这些层，将所有可能应用程序的全量化模型作为其基础模型。当新的摄像头图像进入系统，且检测到并发应用程序的状态发生任何变化时，主进程会调整分配给并发应用程序的内存资源大小（5）。根据调整后的每个应用程序内存空间确定新的位设置后，加载新模型所需的全精度层（6），然后用这些层替代现有的整数层执行。执行完成后，删除全精度层，各模型恢复到其原始的基础模型状态。图 2 展示了上述机制的两种不同情况。接下来将详细说明如何在考虑给定内存空间的情况下构建混合精度模型。

V. 系统架构

我们的主要目标是在满足截止期限的同时，为多个同时运行的应用优化系统资源（如CPU时间和内存空间）的分配。由于实时CNN模型的计算复杂度很高，这一问题至今尚未得到充分解决。

使用高低精度（2）表示模型，是降低深度神经网络计算成本的关键，从而使多个深度神经网络应用能够共享有限的平台资源。DynaMIX 的核心是一种内存资源分配算法，它会确定每个应用应采用的最佳位设置，以最大化总准确率（5）。该算法使用离线创建的应用程序配置文件，将一组混合精度模型（3）的信息保存为约束项。因此，计算资源优化

由内存需求动态变化的应用程序触发。完成一系列资源优化流程后，通过查找表（6）选择要加载/执行的附加层；这些查找表是离线生成的（4）。

为减轻计算密集型进程的在线负担，DynaMIX 被划分为离线和在线阶段（见图3）。1、7 将在第IV节讨论，其余组件将在后续章节中详细说明。

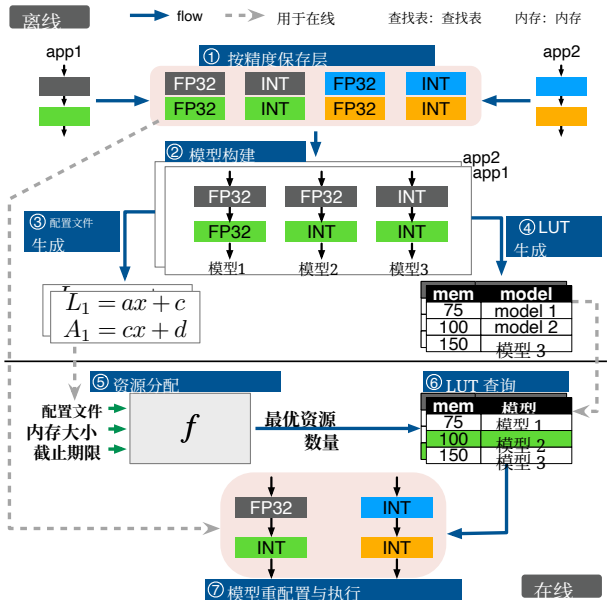


图3。系统架构。

A. 混合精度模型构建

利用所有层的浮点和整数版本，我们可以构建不同的混合精度模型（基于相同的架构），以展示混合精度量化对准确率、速度、模型大小等方面的影响。DynaMIX 的主要目标是在给定资源和截止期限的条件下，找到准确率最高的模型。我们生成具有不同位配置的多种压缩模型（性能各异），并根据可用内存资源量，利用这些模型预测性能。

我们提出一种基于排序的方法来构建模型，使其在不同压缩程度（定义为模型中量化层的数量）下均能获得最高准确率。在这种方法中，按照第III-A节讨论的层敏感度降序排列模型的浮点层。ZeroQ [22] 表明，量化模型（其中多个层以低位宽量化）的总体敏感度，可以通过累加该模型中所有量化层的敏感度来计算。基于这一计算，从全量化模型开始，按照层敏感度降序逐一将整数层替换为对应的浮点层。这意味着，在所有量化程度下，生成的模型相较于全精度模型都会产生最小的输出扰动，从而使准确率下降幅度最小。该模型构建过程如算法1所示。虽然这一过程可以像深度优先搜索一样以递归或迭代方式实现，为便于阅读，我们采用递归方式描述；该过程的初始调用为 BUILD（全量化模型，$\emptyset$）。

我们基于排序的方法为由 L 层组成的模型生成了 L 个压缩模型，从而有效去除了穷举式模型生成方法所产生的 2^L 个模型中不必要且重复的模型。

算法1混合精度　　模型构建

L ：按层敏感度降序排列的一组层
敏感度
$\mathcal{S}$:一组压缩　　模型（$\mathcal{S} \leftarrow \emptyset$)

过程 构建(模型 m，层 l)
　n = 将 l 替换为 m 浮点层　▷ 如果 l 是　$\emptyset, n \leftarrow m$
　$\mathcal{S} \leftarrow \mathcal{S} \cup n$
　v: 该 L中最敏感的层　▷ 如果 L是 $\emptyset$**返回** $\mathcal{S}$
　从 L中移除 v
　构建(n, v)
　返回 $\mathcal{S}$
结束过程

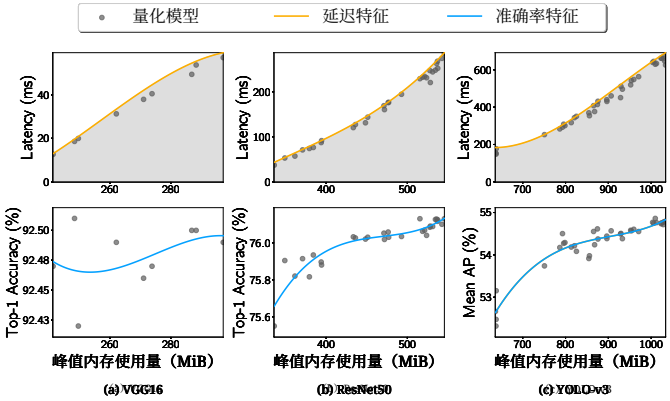


图 4. VGG16、ResNet50 和 YOLO-v3 的模型压缩与性能分析结果。每个灰点表示具有特定位设置的某个压缩模型。黄色线表示最坏情况延迟的配置文件，因此涵盖了灰色区域中所示的所有压缩模型。蓝色线表示最可能准确率的配置文件，展示了准确率的变化趋势。

图4展示了VGG16 [30], 、ResNet50 [31], 和YOLO-v3 [20]的压缩模型的推理结果。请注意，延迟包括模型重配置、执行和恢复所需的时间。本文始终使用2种位宽，即FP32和INT8，以展示使用混合精度模型对应用性能的有效性，即所有层均采用其中一种精度表示。上排的三幅图展示了延迟随压缩程度的变化，下排的三幅图展示了准确率随压缩程度的变化。考虑到模型的内存使用量和延迟可能在每次运行时发生变化，我们对每个模型测量了100次，并将最高峰值内存使用量和延迟作为最坏情况。结果表明，随着压缩程度的提高，延迟会降低，同时准确率不会大幅下降。为了在在线阶段利用压缩模型的性能结果，我们为每个模型创建了两个配置文件和一个查找表。

B. 配置文件生成

上述过程为每个应用生成了多个混合精度模型，因此，我们进行内存资源分配的目标是为所有并发应用找到一组最优的压缩模型。虽然大量的组合能够灵活适应各种实际情况，但寻找最优组合所需的高昂计算成本，使其不适用于实时资源分配。我们的模型压缩会生成 $O(L)$ 个模型；对于 10 FPS 摄像头，每帧的端到端延迟应小于 100 毫秒 [3]。如果平台上同时运行 I 个应用，则寻找最优组合需要耗费过多时间（最长可达$O(100 \cdot L^I)$ ms）。

为实现实时优化，NestDNN 的作者 [7]仅考虑了五个候选模型，但这会牺牲资源利用率和准确率。然而，在 DynaMIX 中，系统通过对压缩模型应用多项式回归，离线为每个应用生成延迟和准确率的多项式配置文件。这种应用性能分析无需额外的

在线阶段的推理处理。我们介绍了延迟特征和准确率特征，它们用于我们的在线内存资源分配算法。

表 III 延迟与准确率特征。X 表示峰值内存使用量。

神经网络模型	Type	配置文件
VGG16	延迟	$-0.0001472\ x^3 + 0.1135x\ x^2 - 28.16\ x + 2267$
	准确率	$-7.781e\text{-}07\ x^3 + 0.0006417\ x^2 - 0.1753\ x + 108.4$
ResNet50	延迟	$1.021e\text{-}05\ x^3 - 0.01083\ x^2 + 4.648\ x - 683$
	准确率	$1.285e\text{-}07\ x^3 - 0.000181\ x^2 + 0.08553\ x + 62.48$
YOLO-v3	延迟	$-6.706e\text{-}06\ x^3 + 0.01861\ x^2 - 15.5\ x + 4241$
	准确率	$6.764e\text{-}08\ x^3 - 0.0001834\ x^2 + 0.1676\ x + 2.821$

1）延迟特征：通过使用目标深度神经网络的延迟特征，我们可以估算模型压缩到特定大小时的最坏情况延迟。如果某些模型未被延迟特征涵盖，那么运行时新选定的模型可能会超过根据延迟特征得出的预期值，从而无法保证截止期限。因此，延迟特征应涵盖所有生成的压缩模型。第 VI-B 节讨论了多任务平台上的任务间干扰对执行延迟的影响。

为了创建涵盖所有压缩模型的延迟特征，当压缩模型的延迟高于更小模型的延迟时，就将其存储起来，随后对存储的模型进行多项式回归。我们采用三次多项式，因为延迟与峰值内存使用量并非完全成正比，但相关性较强，而且高阶回归会增加优化算法的负担（见图 4）。为每个模型获得多项式特征后，为保证给定内存空间下的最坏情况延迟，我们从所有压缩模型集合的延迟特征中，删除那些延迟高于估算值的模型。VGG16、ResNet50 和 YOLO-v3 的最终延迟特征见表 III。

2）准确率特征：准确率特征的作用与延迟特征不同，后者关注准确率随压缩程度的变化。为此，在创建延迟特征后，我们对所有剩余模型进行回归，不再进一步删除模型。准确率与峰值内存使用量也呈弱正线性相关，因此采用三次多项式进行回归（见图 4）。VGG16、Res Net50 和 YOLO-v3 的准确率特征见表 III。

C. 最优资源分配

当新的摄像头图像进入系统，且检测到应用进程状态发生任何变化时，DynaMIX 会为并发应用分配内存资源，同时在给定截止期限内最大化总准确率。该过程中的优化问题通过将离线阶段生成的延迟和准确率特征纳入考虑来求解，从而确定为每个应用分配的最优内存资源。接下来，我们将介绍资源分配算法的设计，该问题被表述为一个约束非线性优化问题。

DynaMIX 假设内存容量有限，实时应用的时间要求由摄像头帧率决定。因此，并发应用的资源分配问题表述如下：

$$\begin{aligned} \arg\max_{m_i} \quad & \sum_{i \in I} \lambda_i A_i\left(m_i\right) \\ \text{s.t.} \quad & \sum_{i \in I} L_i\left(m_i\right) \leq D - \varepsilon \\ & \forall i \in I : m_i \leq M_{\max} - \mu_i \\ & \forall i \in I : m_i^L \leq m_i \leq m_i^U \end{aligned} \tag{1}$$

其中，I 是并发应用的集合。我们希望为每个应用 $i \in I$ 确定最佳内存大小 m_i，使总准确率最大化。总准确率按所有应用准确率的加权和计算；当各应用的重要程度不同，或量化过程中准确率的变化程度因应用而异时，λ_i 用于赋予不同权重。本文将 λ_i 设为 1。对于约束项，我们考虑应用可使用的内存容量 $M_{\max}$以及给定的截止期限 D。其中，ε 是主进程中的平均处理时间；μ_i 是并发应用基础模型的内存使用量，计算方式为 $\mu_{total} - \mu_{app_i}$，其中 μ_{total} 为 $\mu_{vgg} + \mu_{resnet} + \mu_{yolo}$，而 μ_{app_i} 是 μ_{vgg}、μ_{resnet}和 μ_{yolo}之一。ε 和 μ_i 将在第 VI-B 节详细说明，该节将评估求解此问题的可行性。我们还使用在曲线生成阶段获得的延迟特征 L_i 和准确率特征 A_i。延迟相关约束项确保估算的延迟L_i（m_i）不超过截止期限。最后，我们为 m_i 设置下限和上限（m_i^L、m_i^U），以缩小求解此优化问题时的搜索空间。m_i^L 设为应用 i的最小峰值内存使用量，m_i^U设为应用 i的最大峰值内存使用量。优化结果决定分配给每个应用的适当内存空间大小。

算法 2应用 i 的查找表生成

$\mathcal{T}$：查找表 ($\mathcal{T} \leftarrow \emptyset$)
$\mathcal{M}$：一组混合精度模型
z：查找表条目的数量
$max(i)$：最大的峰值内存使用量
$min(i)$：最小的峰值内存使用量
m_{pm}：模型 m的峰值内存使用量
m_{acc}：模型 m的准确率

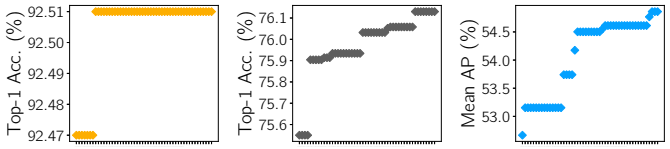
按峰值内存使用量升序排列 $\mathcal{M}$
$val \leftarrow 0, oldkey \leftarrow 0$
当 $\mathcal{M} \neq \emptyset$ **时**
  $key \leftarrow min(i) + \left\lceil \frac{(z-1)(m_{pm}-min(i))00}{max\ i - min\ i} \right\rceil \frac{max(i)-min(i)}{(z-1)}$
  如果 $key == oldkey$
    如果 m_{acc} 大于 val_{acc} **则**
      $val \leftarrow m$
    结束 if
  else
    $\mathcal{T} \leftarrow \mathcal{T} \cup (oldkey,\ val)$
    $val \leftarrow m,\quad oldkey \leftarrow key$
  结束 if
  在 $\mathcal{M}$ 中移除 m
结束 while

D. 最终模型选择的查找表

资源分配后，DynaMIX 应确定要加载到系统中的压缩模型。对于每个深度神经网络模型，压缩模型数量众多，若依次搜索最合适的模型，很可能会耗费过多时间，无法满足实时应用的需求。因此，我们通过查找表缩短搜索时间。每个模型的查找表在离线阶段生成，并在在线阶段使用。

1）查找表生成：我们提出了一种简单而高效的查找表生成方法，该方法基于等差数列原理。首先，我们将每个应用的峰值内存使用量范围等分。其次，以键值对的形式向表中添加新表项，其中键是某个划分区间的端点，值是在分配与该端点对应的内存大小时，准确率最高的模型的位设置。从最小的区间开始，将各区间对应的表项逐一插入表中。因此，无论深度神经网络模型的复杂程度如何，生成的查找表大小始终固定（即预设的表项数量）。图5 展示了位设置存储在查找表中的模型的准确率。每张图显示了 DynaMIX 需要为每个应用存储的不同设置数量（例如，VGG16 为 2）。查找表生成的详细过程见算法 2。

2) 查询查找表：生成的查找表使 DynaMIX 能够在在线资源分配后快速选择最终模型。



(a) VGG16 (b) ResNet50 (c) YOLO-v3

图5. 查找表中存储的模型精度。每个 x 轴上的刻度表示将峰值内存使用量范围按固定数量（即查找表条目数）划分后得到的区段。每个生成的查找表都包含 50 个表项。

$$key = min(i) + \left\lfloor \frac{(z-1)(m_i^\star - min(i))}{max(i) - min(i)} \right\rfloor \frac{max(i) - min(i)}{(z-1)}$$

其中， $m_i^\star$ 表示分配给应用 i的内存大小， z为表项数量。 $max(i)$ 和 $min(i)$ 分别表示应用 i的最大和最小峰值内存使用量。

将生成的键值输入表格后，即可确定要执行的最终模型。随后，应用的每个子进程都会为新选定的模型加载所需的额外浮点层，并按照第IV节所述的重新配置执行路径运行。

VI. 评估

现在，我们在仿真环境中实现表 II 所列的所有组件，对 DynaMIX 进行评估。我们使用 PyTorch 1.10.2[32]，（最具代表性的深度学习框架之一）实现了层量化和模型执行。根据 PyTorch 的量化支持，在 x86-64 CPU 上采用 INT8 作为低精度格式，以加速深度神经网络执行。我们没有使用间接指标来衡量应用性能，而是运行所有压缩模型来测量实际延迟，并使用内存分析器 [33]测量峰值内存使用量。生成配置文件时，我们使用 NumPy API 中的多项式回归工具 polyfit。资源分配算法使用 SciPy API 中的序列最小二乘规划工具编写。所有实验均在配备 AMD Ryzen 7 5700G 处理器和 16GB 内存、运行 Ubuntu Linux 20.04 LTS 操作系统的设备上进行。鉴于 DynaMIX 的目标，我们希望满足并发深度神经网络应用的相同截止期限。

随后，目标截止时间是通过反映在我们的环境中测得的原始（即全精度）模型的性能来相对确定的。尽管我们是在仿真环境中进行实验，但我们的目标是展示当运行中的应用状态发生变化时，DynaMIX 的资源优化能力。

多任务实现。我们的多任务环境由主进程和子进程构成。主进程检测系统中的变化，并向并发应用分配平台资源。每个子进程负责执行其深度神经网络模型。请注意，子进程的数量等于可能的应用类型数量。对于多任务，我们采用非抢占式最短作业优先（SJF）调度，以避免在基于深度神经网络的应用之间进行上下文切换的高开销，考虑应用之间的同等优先级（归因于共同截止时间），并最小化并发应用的平均等待时间。例如，本文中介绍的三个应用程序按照 VGG16、ResNet50 和 YOLO-v3 的顺序进行调度。

如第IV节所述，每个子进程首先为其应用构建全量化模型，作为基例。处理来自摄像头的图像时，Dyna MIX 会根据是否检测到并发应用的状态发生变化而采取不同的行为。如果发生变化，主进程会为每个应用确定最优内存空间，然后通过发送消息将相应的位配置通知其中一个子进程（情况 ***1***）。否则，主进程不会进行资源分配，而是将上一次的位配置通知子进程（情况 ***2***）。子进程收到信息后，会加载必要的层、运行模型，然后通过共享内存向主进程报告任务已完成。随后，主进程重复此过程，直到所有应用程序都执行完毕。在第六节C部分的实验中，我们以 1 秒为间隔调整应用状态。如果情况 ***1*** 满足截止期限，那么情况 ***2***也能满足。因此，我们展示情况 ***1***的截止期限满足情况。

接下来，我们首先介绍评估中使用的深度神经网络模型和数据集，并讨论 DynaMIX 的可行性、有效性和鲁棒性。

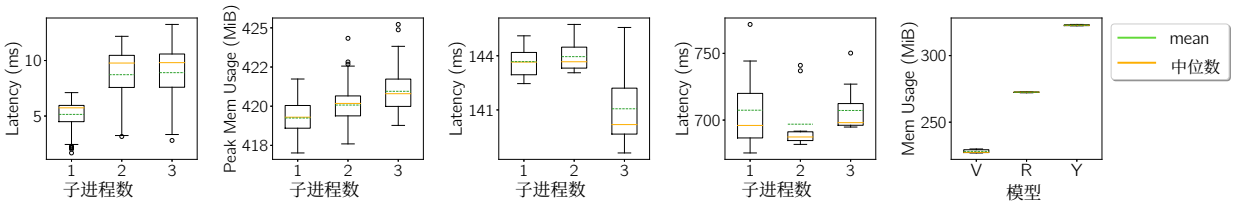
A. 神经网络和数据集

图像分类和目标检测应用经常在自动驾驶汽车上同时运行。此外，即使是相同的任务，模型的复杂度也会因功能而大不相同。例如，对汽车前方的物体进行分类，比识别交通信号灯的颜色更困难。考虑到多样化的真实场景，我们为两类应用选取了以下三种深度神经网络模型。尽管最先进的模型不断快速演进，但其底层架构仍以我们选用的模型为基础 [34], [35]。

1) CIFAR-10 上的 *VGG16*：CIFAR-10 [36]由 6 万张图像组成，涵盖 10个类别，图像为低分辨率的动物或车辆彩色图像。由于类别数量较少，且不同类别物体之间的视觉差异明显，CIFAR-10 分类被认为是一项简单易行的任务。基于 CIFAR-10，我们使用了在该数据集上完成充分训练的 VGG 16,[1] 。VGG 是最流行且最具代表性的架构之一 [7], [24], [25], ，其所有卷积层都使用小型（例如， 3×3）卷积核。特别是，VGG16（由 16 层组成）在 CIFAR-10 数据集上展现出近乎最先进的推理准确率。由于任务简单，如图 4 和图 5 所示，VGG16 的准确率高于其他模型。

2）*ImageNet*上的*ResNet50*： ImageNet [37] 由140万张图像组成，分为1000个类别，每张都是动物、植物、车辆或电子设备的高分辨率彩色图像。由于类别间特征相似度高且类别数量众多，ImageNet上的分类比CIFAR-10更困难。本研究使用了在ImageNet数据集上[2] 完整训练的ResNet50模型

[1]来源：https://github.com/chengyangfu/pytorch-vgg-cifar10 2来源：https://github.com/pytorch/vision/blob/main/torchvision/



(a) 资源分配 (b) 资源分配 (c) YOLO：全量化 (d) YOLO：全精度 (e) 基础模型

图6. 主进程和子进程的延迟及峰值内存使用量分布。子进程数为1时，加载并执行YOLO-v3；为2时，加载并执行ResNet50和YOLO-v3；为3时，加载并执行全部三个应用程序。最后一张图中，“V”“R”和“Y”分别代表“VGG16”“ResNet50”和“YOLO-v3”。每个箱线图均包含100次测量结果。

。ResNet在VGG之后发布，其神经网络架构深度远超VGG（最多152层，而VGG最多19层），并通过在网络中添加跳跃连接，解决了设计深度网络时出现的梯度消失问题。由于ImageNet的复杂性，ResNet-50在CIFAR-10上的准确率低于VGG16（见图4和图5），但目前广泛使用的基于ImageNet的模型准确率可达 75 ∼ 80%，而ResNet-50在ImageNet上展现出接近最先进水平的推理准确率。更重要的是，ResNet架构在构建当前高性能深度模型方面具有巨大潜力 [38]。

*3) MS COCO*上的*YOLO-v3*：对于目标检测任务，我们使用了在MS COCO数据集上训练的YOLO-v3模型[3] [39], 。该数据集包含91个类别的5K张图像，内容与日常生活相关，例如包含一个或多个车辆及日用品的图像。YOLO是首个能够在一次前向传播中对输入图像或视频中的多个目标进行定位和分类的目标检测架构。得益于其速度，实时目标检测已应用于自动驾驶领域 [3]，并且其在提升安全性和便利性方面的作用日益增强。在图4和图5中，YOLO-v3的准确率（以“平均精度（AP）”衡量）低于分类模型的准确率（以“Top-1”衡量）。这是因为AP不仅表示分类准确率，还表示对各目标位置的预测准确率，而Top-1仅表示分类准确率。然而，YOLO-v3在准确率和延迟之间实现了良好平衡，是学术界和工业界最常用的实时目标检测模型之一 [3], [4], [40]。

B. DynaMIX 中资源分配算法的可行性

确保资源分配的可行性取决于以下因素：1）求解式（1）所产生的开销和额外内存使用量；2）在式（1）的第一个约束项中使用延迟配置文件是否合理；3）式（1）的第二个约束项中每个应用的基例模型的内存使用量。我们通过逐一考察这些因素，展示所提出算法的可行性。

1) 求解式（*1*）时的运行时开销和内存使用量：式（1）包含一个三次目标函数，以及三次和线性约束。尽管使用下界/上界缩小了搜索空间，但我们仍需评估其运行时开销，以确保满足截止期限。图6 (a) 显示了并发应用数量与求解式（1）的延迟之间的关系。单任务处理情况下的延迟约为多任务情况下的 1.7×smallerthan。基于这一观察，我们将式（1）中的 ε 设为 15 ms，同时考虑资源分配的最大延迟。与此同时，图6 (b) 显示，无论运行中的应用数量如何，峰值内存使用量几乎保持不变（最小均值与最大均值之比约为 1）。因此，鉴于 M_{max} 是应用子进程的内存容量，而非主进程的内存容量，主进程不需要更多空间，因此对 M_{max}影响很小。

针对真实场景中的各种情况，在改变截止期限和内存容量的同时，测量图 6（a）和（b）中每个框对应的延迟和峰值内存使用量。

2）使用延迟配置文件的合理性：鉴于延迟配置文件基于独立运行时的测量结果，即使与其他应用程序多路复用，延迟也应保持不变。从图 6（c）和（d）可以看出，在基于非抢占式最短作业优先调度的多任务环境中，每个压缩模型的延迟都保持不变。受篇幅限制，我们仅展示了两个模型：图（c）为 YOLO-v3 的全量化模型，图（d）为其全精度模型。由此可得，延迟配置文件可用于第一个约束项。

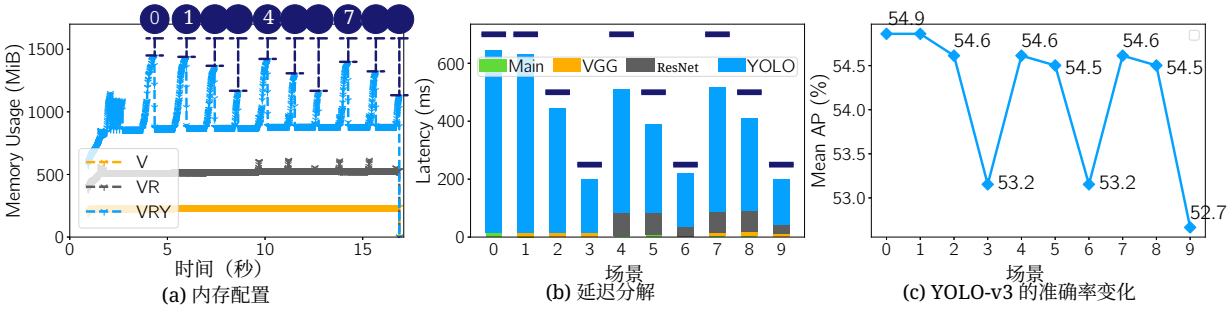
3) 各应用基础模型的内存使用量：加载基础模型后，并发应用越多，占用的内存空间就越大。图6 (e) 展示了各基础模型占用的内存。根据其均值，我们将 μ_{vgg}、μ_{resnet}、μ_{yolo}、μ_{total} 分别设为 228、273、323、823，用于第二个约束项。

C. DynaMIX 的有效性

为展示 DynaMIX 在多任务环境中的有效性，我们设计了若干场景，在两种时间约束下改变并发应用的数量：充足的截止期限和收紧后的截止期限，以运行多个深度神经网络应用（见表 IV）。受篇幅限制，我们选取了一些确定性但具有代表性的案例（而非随机案例），通过调整截止期限或并发应用的数量来展示各应用性能的变化。

通常，模型执行速度在很大程度上取决于处理引擎（例如 CPU、GPU）。考虑到 CPU 的速度远低于 GPU，我们根据在仿真环境中测得的延迟设置截止期限。在

[3]来源：https://github.com/eriklindernoren/PyTorch-YOLOv3



(a) 内存配置　(b) 延迟分解　(c) YOLO-v3 的准确率变化

图7. 资源优化结果。(a) 中，我们用蓝线标出了 3 个应用的峰值内存使用量与内存容量之间的余量；“V”表示 VGG16，“VR”表示 VGG16 和 ResNet50 的内存使用量之和，“VRY”表示所有 3 个应用使用的总内存空间。(b) 显示了总延迟（包括模型重配置、应用程序执行等）以及各个截止期限（用蓝线标出）；(c) 显示了 YOLO-v3 的准确率变化。

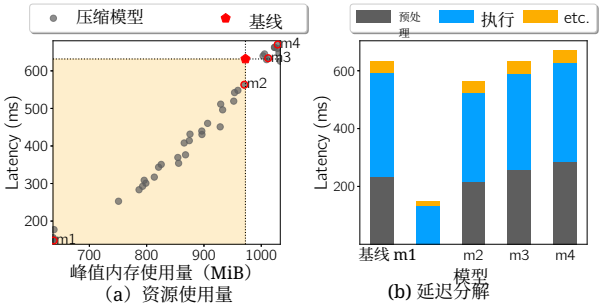
表 IV 多任务系统上的评估场景

场景	描述
初始	基于 YOLO-v3 的应用程序独立运行 。
1	基于 VGG16 的应用程序于 开始运
2	截止期限在 从 700 ms 缩短至 500 ms。
3	截止期限在 时从 500 ms 缩短至 250 ms。
4	基于 ResNet50 的应用程序于 开始运
5	截止期限从 700 毫秒降至 500 毫秒时。
6	截止期限从 500 毫秒降至 250 毫秒，变化范围为 。
7	基于 VGG16 的应用程序于 启动。
8	截止期限在 时从 700 ms 缩短至 500 ms。
9	截止期限从 500 毫秒缩短至 250 毫秒，见 。

具体而言，我们采用了 YOLO-v3 混合精度模型产生的最小（148 ms）、中位数（470 ms）和最大（702 ms）最坏情况延迟。考虑到这些模型的最大峰值内存使用量，我们将内存容量设为 1600 兆字节。图 7 (a) 和 (b) 显示，Dyna MIX 的资源优化能够满足截止期限。

我们假设初始环境（ ）处于这样一种状态：YOLO-v3 单独运行，并且有足够的时间和内存空间运行全精度模型。在这种情况下，DynaMIX 为应用分配 1026 兆字节，并预期在使用准确率特征时获得最高准确率。尽管目标截止时间（700 毫秒）是在考虑全精度模型的情况下确定的，DynaMIX 仍选择了另一种模型（整数层占总层数的 21%），该模型运行得更快更准确，同时占用更少的资源。在 中，尽管与 VGG16 共享资源，YOLO-v3 仍使用与 中相同的位配置。这一现象归因于 VGG16 较低的内存使用量和时间复杂度。此外，鉴于无论压缩程度如何，VGG16 的准确率几乎保持不变（见图 4），DynaMIX 将 VGG16 最大程度地压缩，并将大部分资源分配给 YOLO-v3。然而，在 中，运行 ResNet50 需要同时压缩 ResNet50 和 YOLO-v3。具体而言，分配给 YOLO-v3 的内存空间从 1017 兆字节减少到 952 兆字节，因此 YOLO-v3 使用了比 更多的低精度层（占总层数的 60%）。

。在 中，由于 VGG16 的计算和内存开销较低，ResNet50 和 YOLO-v3 的压缩程度与 中相同。 、 和 展示了 DynaMIX 在截止期限收紧时（从



（a）资源使用量　(b) 延迟分解

图 8. 将 DynaMIX 和基线应用于 YOLO-v3 时的结果比较。(a) 中的红色五边形表示使用基线时的最坏情况延迟和峰值内存使用量，这两项指标均基于 100 次测量计算得出。(b) 展示了 (a) 中主模型的延迟分解。对于基线，“预处理”表示加载整个模型所需的时间；否则，它表示模型重配置所需的时间。“等等”包括模型恢复和进程间通信所花费的时间。

700 毫秒缩短至 500 毫秒）对并发应用进行资源优化的结果。 、 和 展示了截止期限进一步收紧（从 500 毫秒缩短至 250 毫秒）时的优化结果。在这些场景中，通过构建更紧凑的混合精度模型来执行并发深度神经网络模型。因此，图 7(a)中峰值内存使用量与内存容量之间留出了更大的余量。测量到的延迟/峰值内存使用量与截止期限/内存容量之间的整体余量看起来可能较大，这是因为我们在性能分析时采用了最坏情况延迟和最大峰值内存使用量（通过迭代测量得出），因此余量会随执行情况而变化。

我们的资源分配旨在不违反任何时间约束的前提下最大化总准确率，因此准确率没有显著损失（见图7 (c)）。由于篇幅有限，我们仅展示了 YOLO-v3 的结果；它是对实时执行和总准确率下降影响最大的模型。在所考虑的所有场景中，三个模型的准确率下降均低于 2.2%。最大准确率下降出现在 中的 YOLO-v3。

D. 性能比较

1) 与基线的比较：如果为新应用分配了足够的资源，使其能够在情况 ***1***下运行全精度模型，子进程便会通过加载其所有浮点层来重新配置深度神经网络流水线，这种做法对资源的利用效率可能低于基线方案（即一次性加载整个全精度模型

无需重新配置）。尽管存在这种效率低下的情况，我们仍展示了 DynaMIX 在大多数情况下能够高效利用资源。

图 8 (a) 展示了 YOLO-v3 混合精度模型的峰值内存使用量和最坏情况延迟（与图 4 中的 YOLO-v3 延迟图相同），但这次用红色五边形标出了基线方法产生的峰值内存使用量和最坏情况延迟，图 8 (b) 则展示了主要情况（以红色空心圆点标出）的延迟细分。此分析揭示了 DynaMIX 有效性的几个方面。首先，*m4* （执行路径中的所有层都替换为浮点层）相比基线，在延迟和内存使用方面效率较低。然而，我们的压缩模型中有 73.7% 的内存使用量和处理时间有所减少（图 8 中的黄色区域），其中尤以 78.9% 的压缩模型延迟降低最为明显。对于 *m1*（所有层均为低精度），延迟和内存使用量相较基线分别减少了 4.3× 和 1.5×。此外，当约 22.7% 的层被量化时（*m3*），其延迟与基线相同；当约 30.7% 的层被量化时（*m2*），其内存使用量与基线相同。根据这些结果，我们得出结论：DynaMIX 能够通过优化运行中应用程序所需的计算和内存资源，对给定的截止期限作出良好响应。

2）与 NestDNN 的比较：NestDNN [7]也采用深度神经网络剪枝方法来运行并发视觉应用（非实时）。它们根据剪枝程度创建固定数量的小型模型，并从内存使用量、准确率和能耗方面展示了其可行性。能耗不在我们的研究范围内，而应用的内存使用量则通过其模型大小这一间接指标进行讨论。因此，我们仅比较 DynaMIX 与 NestDNN 的准确率。

表V展示了将两种方法应用于 VGG16 时所生成模型之间的差异。这里展示了 DynaMIX 的两类模型：1）所有生成的混合精度模型（ALL）；2）保存在查找表（LUT）中的模型。NestDNN 在线进行资源分配时仅使用五个压缩模型，而我们可以通过将应用程序配置文件纳入优化问题，并借助查找表缩短模型搜索时间，来处理更多模型。此外，DynaMIX 生成的所有压缩模型的准确率都高于 NestDNN，这是因为 NestDNN 的剪枝方案会改变模型参数的分布，导致准确率显著下降。尽管作者通过重新训练在一定程度上恢复了准确率，但由于模型容量减小，准确率并未完全恢复。DynaMIX 较高的准确率提升了其可行性，但压缩比有所降低。不过，可以通过采用更低的精度（例如 INT4）来提高压缩程度；如果同时采用不同的低精度，则可以兼顾更高的压缩程度和准确率。

E. DynaMIX 的鲁棒性

由于自动驾驶汽车中的系统错误可能导致紧急情况，我们必须确保系统在系统可用性方面的鲁棒性

表V NestDNN [7] 与 DynaMIX 的比较。最大压缩比是指未压缩模型大小与最小压缩模型大小的比值。

	[7]	DynaMIX（全部）	DynaMIX（查找表）
# 压缩模型	5 17 2		
最低准确率	83	92.47	92.47
最高准确率	89	92.51	92.51
最高压缩比 7.8		4.0	4.0

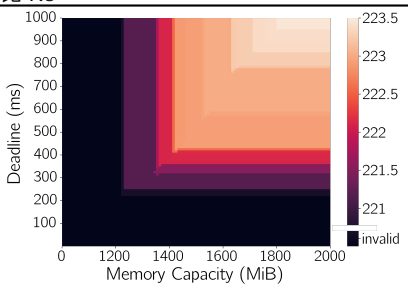


图 9. 改变截止期限和内存容量时的总准确率。颜色越浅，总准确率越高。黑色单元格表示无效结果，这是由于优化无法完成或准确率显著下降所致。

和质量——这是系统失效的两个关键因素。当截止期限设定低于某个水平时，DynaMIX 中的资源优化机制无法得出结果。此外，虽然可以使用高度压缩的模型来满足较短的截止期限，但其准确率较低，可能导致致命事故。考虑到这些影响系统可用性和质量的因素，我们可以评估系统鲁棒性。

下面我们使用查找表得出的总准确率来展示 DynaMIX 的鲁棒性。由于不同内存容量的平台上的鲁棒性可能有所不同，我们在图 9 中展示了截止期限、内存容量与总准确率之间的关系。黑色单元格表示系统在以下三种情况下会产生无效结果：可用位宽无法满足截止期限、内存容量不足以支持 DynaMIX，或总准确率下降超过预设阈值（本文中为 3%）。在其余情况下，DynaMIX 能够为用户保证可接受的质量。

由于我们的环境支持 INT8 和 FP32 运算，总准确率的下降不会超过 3%；但若将黑色区域考虑在内，截止期限和内存容量的下限似乎较高。INT4 和 INT2 等超低位宽整数能够降低这些下限，从而使 DynaMIX 在更紧迫的截止期限和更少的内存容量下仍能无误运行。因此，我们可以利用系统支持的多种位宽来提升系统鲁棒性。

VII. 结论与未来工作

我们提出了一种名为 DynaMIX 的资源优化框架，使多个基于深度神经网络的实时应用能够满足相同的应用截止期限（由摄像头的帧率决定），这对 CPS 应用至关重要。DynaMIX 离线生成应用性能配置文件，并在运行时重新配置混合精度深度神经网络模型，根据运行中应用的状态反映其动态变化的资源需求，从而共享

平台资源。我们在仿真环境中的实现证明了该框架的可行性和有效性。

我们假设有足够的计算资源供这些应用使用，使其模型能够装入内存。如何放宽这一假设是我们未来工作的内容之一。我们还希望将 DynaMIX 应用于混合关键性系统，在该系统中可以运行多个关键性级别不同的应用。尽管当前版本专注于基于 C NN 的应用，DynaMIX 也可以扩展以支持各种基于深度神经网络的应用，包括语言模型，从而促进在自动驾驶汽车上执行许多实时应用。

参考文献

[1] 特斯拉，“https://www.tesla.com/autopilot。”[2] 丰田，“https://global.toyota/en/newsroom/corporate/35063150.html。”[3] M. Yang、S. Wang、J. Bakita、T. Vu、F. D. Smith、J. H. Anderson 和 J.- M. Frahm，“重新思考面向时间敏感型自动驾驶应用的 CNN 框架：应对一项工业挑战”，载于 *2019* 年 *IEEE* 实时与嵌入式技术及应用研讨会。电气电子工程师学会，2019 年，第 305–317 页。[4] S.-C. Lin、Y. Zhang、C.-H. Hsu、M. Skach、M. E. Haque、L. Tang 和 J. Mars，“自动驾驶的架构影响：约束与加速”，载于 第二十三届编程语言与操作系统架构支持国际会议论文集，2018 年，第 751–766 页。[5] S. Han、H. Shen、M. Philipose、S. Agarwal、A. Wolman 和 A. Krishnamurthy，“MCDNN：资源约束下深度流处理的基于近似的执行框架”，载于 第十四届移动系统、应用与服务年度国际会议论文集，2016 年，第 123–136 页。[6] L. N. Huynh 、 Y. Lee 和 R. K. Balan，“DeepMon：面向连续视觉应用的移动 GPU 深度学习框架”，载于第十五届移动系统、应用与服务年度国际会议论文集，2017 年，第 82–95 页。[7] B. Fang、X. Zeng 和 M. Zhang，“NestDNN：面向资源的设备端多租户连续移动视觉深度学习”，载于 第二十四届移动计算与网络年度国际会议论文集，2018 年，第 115–127 页。[8] M. Liu、X. Ding 和 W. Du，“无需卸载的移动设备连续实时目标检测”，载于 *2020* 年 *IEEE* 第四十届分布式计算系统国际会议。电气电子工程师学会，2020 年，第 976–986 页。[9] S. Heo、S. Cho、Y. Kim 和 H. Kim，“采用多路径神经网络的实时目标检测系统”，载于 *2020* 年 *IEEE* 实时与嵌入式技术及应用研讨会。电气电子工程师学会，2020 年，第 174–187 页。[10] W. Kang、S. C hung、J. Y. Kim、Y. Lee、K. Lee、J. Lee、K. G. Shin 和 H. S. Chwa，“DNN-SAM：面向实时目标检测的 DNN 拆分与合并执行”，载于 *2022* 年 *IEEE* 第二十八届实时与嵌入式技术及应用研讨会。电气电子工程师学会，2022 年，第 160–172 页。[11] E. Baek、D. Kwon 和 J. Kim，“多神经网络加速架构”，载于 *2020* 年 *ACM/IEEE* 第四十七届计算机体系结构年度国际研讨会。电气电子工程师学会，2020 年，第 940–953 页。[12] W. Jiang、Z. Song、J. Zhan、Z. He、X. Wen 和 K. Jiang，“面向实时多任务应用的混合精度神经网络加速器优化协同调度”，《系统架构杂志》，第 110 卷，文章编号 101775，2020 年。[13] J. S. Jeong、J. Lee、D. Kim、C. Jeon、C. Jeong、Y. Lee 和 B.-G. Chun，“Band：异构移动处理器上的协调式多 DNN 推理”，载于 第二十届移动系统、应用与服务年度国际会议论文集，2022 年，第 235–247 页。[14] M. Han、H. Zhang、R. Chen 和 H. Chen，“并发 GPU 加速 DNN 推理的微秒级抢占”，载于 第十六届 *USENIX* 操作系统设计与实现研讨会，2022 年，第 539–558 页。[15] R. Krishnamoorthi，“高效推理的深度卷积网络量化：白皮书”，*arXiv* 预印本 *arXiv:1806.08342*，2018 年。[16] H. Park 和 C. Kim，“面向 Web 应用开发的自适应网页与延迟加载组件设计”，《韩国信息与通信工程学会期刊》，第 23 卷，第 5 期，第 516–522 页，2019 年。

[17] J. W. Arendt、P. P. Giangarra、R. K. Manikundalam、D. R. Padgett 和 J. M. Phelan，“共享库延迟加载的系统与方法”，1998年1月13日，美国专利5,708,811。[18] W. Liu、D. Anguelov、D. Erhan、C. Szegedy、S. Reed、C.-Y. Fu 和 A. C. Berg，“SSD：单次多框检测器”，载于《欧洲计算机视觉会议》。施普林格，2016 年，第 21–37 页。[19] S. Ren、K. He、R. Girshick 和 J. Sun，“Faster R-CNN：利用区域提议网络实现实时目标检测”，《IEEE模式分析与机器智能汇刊》，第 39 卷，2016 年，第 1149、1137 页。[20] J. Redmon、S. Divvala、R. Girshick 和 A. Farhadi，“You Only Look Once：统一的实时目标检测”，载于《IEEE计算机视觉与模式识别会议论文集》，2016 年，第 779–788 页。[21] K. Wang、Z. Liu、Y. Lin、J. Lin 和 S. Han，“HAQ：采用混合精度的硬件感知自动量化”，载于《IEEE/CVF计算机视觉与模式识别会议论文集》，2019 年，第 8612–8620 页。[22] Y. Cai、Z. Yao、Z. Dong、A. Gholami、M. W. Mahoney 和 K. Keutzer，“ZeroQ：一种新型零样本量化框架”，载于《IEEE/CVF计算机视觉与模式识别会议论文集》，2020 年，第 13 169–13 178 页。[23] Z. Dong、Z. Yao、A. Gholami、M. W. Mahoney 和 K. Keutzer，“HawQ：具有混合精度的神经网络 Hessian 感知量化”，载于《IEEE/CVF国际计算机视觉会议论文集》，2019 年，第 293–302 页。[24] D. Lee、M. Cho、S. Lee、J. Song 和 C. Choi，“利用合成数据生成的混合精度量化新型敏感度指标”，载于 2021 年 IEEE 国际图像处理会议。电气电子工程师学会，2021 年，第 1294–1298 页。[25] W. Zhe、J. Lin、V. Chandrasekhar 和 B. Girod，“优化深度神经网络权重与激活值压缩的比特分配”，载于 2019 年 IEEE 国际图像处理会议。电气电子工程师学会，2019 年，第 3826–3830 页。[26] NVIDIA。 [在线]。可访问：https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/ [27] P. K. Raha、T. Knopp、S. Ahmad、A. Ansari、F.-H. Ho、T. To、V. Nalluri、M. Sarmah 和 R. Patwari，“8.2 一种多功能的 7纳米自适应计算加速平台处理器”，载于 2020 年 IEEE 国际固态电路会议-(ISSCC)。电气电子工程师学会，2020 年，第 146–148 页。[28] C. Tang 、 H. Zhai 、 K. Ouyang 、 Z. Wang 、 Y. Zhu 和 W. Zhu，“任意位宽网络：逐层量化与自适应推理相结合的方法”，arXiv 预印本 arXiv:2204.09992，2022 年。[29] A. Bulat 和 G. Tzimiropoulos，“Bit-Mixer：具有运行时位宽选择的混合精度网络”，载于《IEEE/CVF国际计算机视觉会议论文集》，2021 年，第 5188–5197 页。[30] K. Simonyan 和 A. Zisserman，“ 用于大规模图像识别的超深卷积网络 ”，arXiv 预印本 arXiv:1409.1556，2014 年。[31] K. He、X. Zhang、S. Ren 和 J. 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》，2016 年，第 770–778 页。[32] T. Contributors，“https://pytorch.org/docs/stable/quantization.html”，2019 年。[33] F. Pedregos，“https://github.com/pythonprofilers/memory profile。”[34] S. Yun、D. Han、S. J. Oh、S. Chun、J. Choe 和 Y. Yoo，“CutMix：使用可定位特征训练强分类器的正则化策略”，载于《IEEE/CVF国际计算机视觉会议论文集》，2019 年，第 6023–6032 页。[35] X. Ding、X. Zhang、N. Ma、J. Han、G. Ding 和 J. Sun，“RepVGG：让 VGG风格卷积网络再次强大”，载于《IEEE/CVF计算机视觉与模式识别会议论文集》，2021 年，第 13 733–13 742 页。[36] A. Krizhevsky，“ 从微型图像中学习多层特征 ”，2009 年。[37] O. Russakovsky、J. Deng、H. Su、J. Krause、S. Satheesh、S. Ma、Z. Huang、A. Karpathy、A. Khosla、M. Bernstein、A. C. Berg 和 L. Fei-Fei，“ImageNet大规模视觉识别挑战赛”，《国际计算机视觉杂志》，第 115 卷，第 3 期，第 211–252 页，2015 年。[38] H. Touvron、A. Vedaldi、M. Douze 和 H. J´egou，“修正训练集与测试集分辨率差异”，《神经信息处理系统进展》，第 32 卷，2019 年。[39] T.-Y. Lin、M. Maire、S. Belongie、J. Hays、P. Perona、D. Ramanan、P. Doll´ar 和 C. L. Zitnick，“Microsoft COCO：常见物体”

上下文”，载于欧洲计算机视觉会议。施普林格，2014，第 740–755 页。[40] W. Kang、S. Chung、J. Y. Kim、Y. Lee、K. Lee、J. Lee、K. G. Shin 和 H. S. Chwa，“DNN-SAM：用于实时目标检测的 DNN 拆分与合并执行。”