Latency-Aware Unified Dynamic Networks for Efficient Image Recognition

Yizeng Han, Zeyu Liu, Zhihang Yuan, Yifan Pu, Chaofei Wang, Shiji Song, *Senior Member, IEEE*, and Gao Huang, *Member, IEEE*

***Abstract*— Dynamic networks have become a pivotal area of study in deep learning due to their ability to selectively activate computing units (such as layers or channels) or dynamically allocate computation to information-rich regions. This capability significantly curtails unnecessary computations, adapting to varying inputs. Despite these advantages, the practical efficiency of dynamic models often falls short of theoretical computation. This discrepancy arises from three primary challenges: 1) a lack of a *unified framework* across different dynamic inference paradigms due to the fragmented research landscape; 2) an excessive focus on algorithm design at the expense of *scheduling strategies*, which are essential for optimizing resource utilization on hardware; and 3) the complexity of latency evaluation, since most current libraries cater to static operators. To tackle these issues, we introduce *Latency-Aware Unified Dynamic Networks (LAUDNet)*, a general framework that integrates three fundamental dynamic paradigms–spatially-adaptive computation, layer skipping, and channel skipping–into a single unified formulation. LAUDNet not only refines algorithmic design but also enhances scheduling optimization with the aid of a latency predictor. This predictor efficiently and accurately predicts the inference latency of dynamic operators on specific hardware setups. Our empirical assessments across multiple vision tasks–image classification, object detection, and instance segmentation–confirm that LAUDNet significantly bridges the gap between theoretical and practical efficiency. For instance, LAUDNet cuts down the practical latency of its static counterpart, ResNet-101, by over 50% on hardware platforms like V100, RTX 3090, and TX2 GPUs. Additionally, LAUDNet excels in the accuracy-efficiency trade-off compared to other methods.**

***Index Terms*—Dynamic networks, efficient inference, convolutional neural networks, vision transformers.**

Manuscript received 30 August 2023; revised 20 February 2024; accepted 21 April 2024. Date of publication 25 April 2024; date of current version 5 November 2024. This work was supported in part by the National Key R&D Program of China under Grant 2021ZD0140407, in part by the National Natural Science Foundation of China under Grant 42327901 and Grant 62321005, and in part by the Guoqiang Institute of Tsinghua University. We also appreciate the generous donation of computing resources by High-Flyer AI. Recommended for acceptance by J. Wang. *(Yizeng Han, Zeyu Liu, and Zhihang Yuan are equal contribution to this work). (Corresponding author: Gao Huang.)*

Yizeng Han, Yifan Pu, Chaofei Wang, and Shiji Song are with the Department of Automation, Tsinghua University, Beijing 100084, China (e-mail: yizeng38@gmail.com; pyf20@mails.tsinghua.edu.cn; wcf-119@163.com; shijis@tsinghua.edu.cn).

Zeyu Liu is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: liuzeyu20@mails.tsinghua.edu.cn).

Zhihang Yuan is with Houmo.AI., Beijing 100088, China (e-mail: hahnyuan@gmail.com).

Gao Huang is with the Beijing Academy of Artificial Intelligence, Department of Automation, Tsinghua University, Beijing 100084, China (e-mail: gaohuang@tsinghua.edu.cn).

Code is available at: https://www.github.com/LeapLabTHU/LAUDNet.

Digital Object Identifier 10.1109/TPAMI.2024.3393530

I. Introduction

Deep neural networks have demonstrated exceptional capabilities in various domains such as computer vision [1], [2], [3], [4], [5], natural language processing [6], [7], [8], [9], and multi-modal understanding/generation [10]. Despite their stellar performance, the intensive computational requirements of these deep networks often limit their deployment on resource-constrained platforms, like mobile phones and IoT devices, highlighting the need for more efficient deep learning models.

Unlike traditional static networks [2], [3], [4] which process all inputs uniformly, dynamic models [11] adaptively allocate computation in a data-dependent fashion. This adaptivity involves bypassing certain network layers [12], [13], [14], [15] or convolution channels [16], [17] conditionally, and executing spatially adaptive inference that concentrates computational effort on the most informative regions of an image [18], [19], [20], [21], [22], [23]. As the field evolves and various dynamic models show promise, it begs the question: *How can we design a dynamic network for practical use?*

Addressing this question is challenging due to difficulties in fairly comparing different dynamic-computation paradigms. These challenges fall into three categories: 1) The lack of a unified framework to encompass different paradigms, as research in this area is often fragmented; 2) The focus on algorithm design, which often results in the mismatch between practical efficiency and their theoretical computational potential, due to the significant impact of scheduling strategies[1] and hardware properties on real-world latency; 3) The laborious task of evaluating a dynamic model's latency on different hardware platforms, as common libraries (e.g., cuDNN) are not built to accelerate many dynamic operators.

In response, we introduce a *Latency-Aware Unified Dynamic Network (LAUDNet)*, a framework that unifies three representative dynamic-inference paradigms. Specifically, we examine the algorithmic design of layer skipping, channel skipping, and spatially dynamic convolution, integrating them through a "mask-and-compute" scheme (Fig. 1(a)).

Next, we delve into the challenges of translating theoretical efficiency into tangible speedup, especially on multi-core processors such as GPUs. Traditional literature commonly adopts hardware-agnostic FLOPs (floating-point operations) as a crude efficiency measure, failing to provide latency-aware guidance

[1]Scheduling strategies are essential for practical efficiency because they optimize the use of GPU threads and memory with CUDA codes.

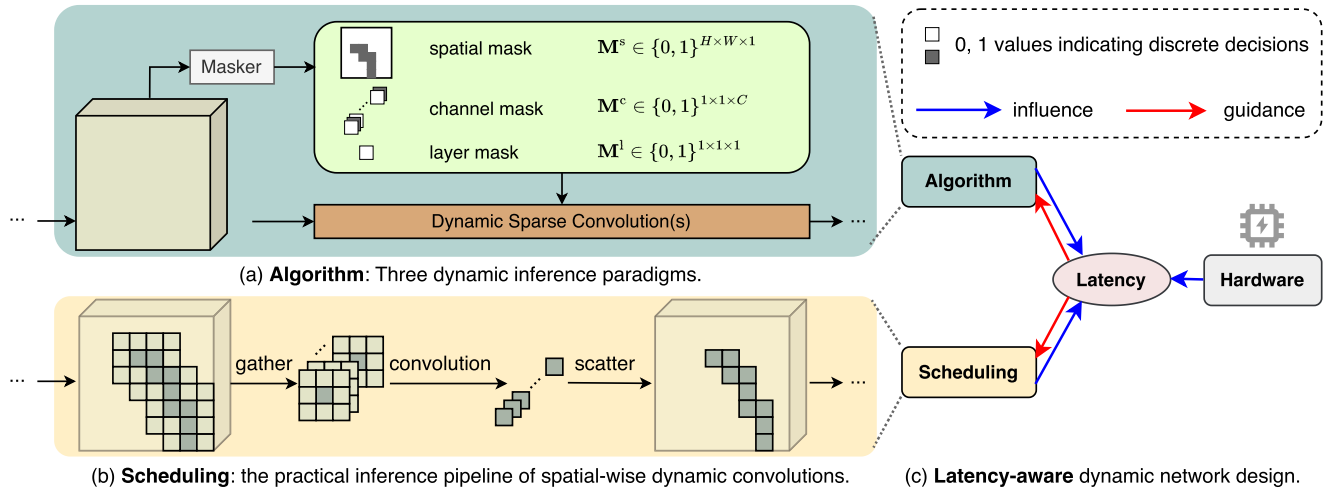


Fig. 1. An overview of our method. (a) illustrates three representative adaptive inference *algorithms* (i.e., spatial-wise dynamic convolution, channel skipping, and layer skipping); (b) is an example of the *scheduling* strategy for spatial-wise dynamic convolution; and (c) presents our key idea of using the latency to *guide* both algorithm design and scheduling optimization.

for algorithm design. In dynamic networks, adaptive computation coupled with sub-optimal scheduling strategies intensifies the gap between FLOPs and latency. Moreover, most existing methods execute adaptive inference at the finest granularity. For instance, in spatial-wise dynamic inference, the decision to compute each feature pixel is made independently [19], [20], [21]. This fine-grained flexibility results in non-contiguous memory access [21], necessitating specialized scheduling strategies (Fig. 1(b)).

Given that dynamic operators exhibit unique memory access patterns and scheduling strategies, libraries designed for static models, like cuDNN, fail to optimize dynamic models effectively. The lack of library support implies that each dynamic operator requires individualized scheduling optimization, code refinement, compilation, and deployment, making network latency evaluation across hardware platforms labor-intensive. To address this, we propose a novel latency prediction model that efficiently estimates network latency by taking into account algorithm design, scheduling strategies, and hardware properties. Compared to hardware-agnostic FLOPs, our predicted latency offers a more realistic representation of dynamic model efficiency.

Guided by the latency prediction model, we tackle the aforementioned challenges within our latency-aware unified dynamic network (LAUDNet) framework. For a given hardware device, we use the predicted latency as the guiding metric for algorithm design and scheduling optimization, as opposed to the conventionally used FLOPs (Fig. 1(c)). In this context, we propose coarse-grained dynamic networks where "whether-to-compute" decisions are made at the patch/group level rather than individual pixels/channels. Though less flexible than pixel/channel-level adaptability in prior works [16], [17], [19], [20], [21], this approach encourages contiguous memory access, enhancing real-world speedup on hardware. Our improved scheduling strategies further permit batching inference. We investigate dynamic inference paradigms, focusing on the accuracy-latency trade-off. Notably, previous research has established a correlation between latency and FLOPs on CPUs [21], [23], hence in this paper, we primarily target the GPU platform, a more challenging but less explored environment.

The LAUDNet is designed as a general framework in two ways: 1) Multiple adaptive inference paradigms can be easily implemented in various vision backbones, like ResNets [2], RegNets [24] and vision Transformers [25], [26]; and 2) The latency predictor functions as an off-the-shelf tool that can be readily applied to diverse computing platforms, such as server-end GPUs (Tesla V100, RTX3090), desktop-level GPU (RTX3060) and edge devices (Jetson TX2, Nvidia Nano).

We evaluate LAUDNet's performance across multiple backbones for image classification, object detection, and instance segmentation. Our results show that LAUDNet significantly improves the efficiency of deep CNNs, both in theory and practice. For instance, the inference latency of ResNet-101 on ImageNet [1] is reduced by >50% on different types of GPUs (e.g., V100, RTX3090 and TX2), without compromising accuracy. Moreover, our method outperforms various lightweight networks in low-FLOPs scenarios.

Although parts of this work were initially published in a conference version [27], this paper significantly expands our previous efforts in several key areas:

- A unified dynamic-inference framework is proposed. While the preliminary paper [27] predominantly focused on spatially adaptive computation, this paper delves deeper into two additional and important dynamic paradigms, specifically, dynamic layer skipping and channel skipping (Fig. 1 and Section III-A). Furthermore, we integrate these paradigms into a unified framework, and provide more thorough study on architecture design and complexity analysis (Section III-B).
- The latency predictor has been enhanced to support an expanded set of dynamic operators, including layer skipping and channel skipping (Section III-C). Moreover, we adopt Nvidia Cutlass [28] to optimize the scheduling strategies. Hardware evaluations demonstrate that our

TABLE I
ABLATION STUDIES ON OPERATOR FUSION

Masker-Conv	Gather-Conv	Scatter-Add	Latency (μs) V100	Latency (μs) TX2
✗	✗	✗	162.4	1084.5
✓	✗	✗	135.1	1072.3
✓	✓	✗	131.7	1024.7
✓	✓	✓	**118.3**	**859.7**

The bold values refer to the best result of one column wihin the same detection/segmentation framework.

latency predictor can accurately predict the latency on real hardware (Fig. 5).

- The LAUDNet framework has been extended to accommodate Transformer architectures, as detailed in Section III-B. This extension notably enhances latency optimization through the implementation of dynamic *token skipping* (spatially adaptive computation), *head (channel) skipping*, and *layer skipping*. Such advancements significantly broaden the applicability of LAUDNet. The empirical evaluation, illustrated in Fig. 10(c) and discussed in Section IV-C2, yields valuable insights into the design of efficient Transformers, underpinning the framework's versatility and efficacy.
- For the first time, we incorporate batching inference for our dynamic operators (Section III-D). This innovation leads to more consistent prediction outcomes and an enhanced speedup ratio on GPU platforms (Figs. 8 and 12).
- We undertake an exhaustive analysis of various dynamic granularities (Fig. 9) and paradigms (Figs. 10, 11, 13, Tables II and III), spanning different vision tasks and platforms, with added evaluations on contemporary GPUs like RTX3060 and RTX3090. We are confident that our results will offer valuable insights to both researchers and practitioners.

II. Related Works

Efficient deep learning has garnered substantial interest. Traditional solutions involve lightweight model design [29], [30], [31], [32], network pruning [33], [34], [35], [36], weight quantization [37], [38], [39], [40], and knowledge distillation [41], [42]. However, these *static* methods have sub-optimal inference strategy, leading to intrinsic redundancy since they process all inputs with equal computation.

Dynamic networks [11], [12], [18], [43] propose an appealing alternative to static models by enabling input-conditional *dynamic inference*. This adaptive approach has yielded superior results across various domains. In visual recognition, prevalent dynamic paradigms include early exiting [12], [13], [44], layer skipping [14], [15], [43], channel skipping [16], [17], [45], and spatial-wise dynamic computation [19], [20], [21]. This paper primarily targets the latter three paradigms, as they can be readily applied to arbitrary visual backbones, thereby offering a generality advantage. Layer skipping and channel skipping explore *structural* redundancy within deep networks by selectively activating computation units, such as layers or convolution channels when processing different inputs. Spatial-wise dynamic models alleviate spatial redundancy in image features and selectively assign computation to the regions most pertinent to the task at hand.

Despite their effectiveness, previous studies often fail to recognize the shared underlying formulation across different dynamic paradigms. In contrast, we introduce a unified framework that encompasses all three paradigms, facilitating a thorough exploration of dynamic networks. Additionally, existing methods primarily concentrate on algorithm design, which often results in a significant disparity between theoretical and practical efficiency. In our latency-aware co-design framework, we bridge this gap by utilizing latency directly from our latency predictor to guide both algorithm design and scheduling optimization. This approach results in improved latency performance across diverse platforms.

Hardware-aware network design. Researchers have acknowledged the necessity to bridge the gap between theoretical and practical efficiency of deep models by considering actual latency during network design. Two primary approaches have emerged: the first entails conducting speed tests on hardware and deriving guidelines to facilitate *hand-designing* lightweight models [32], and the second involves performing speed tests for various types of *static* operators and modeling the latency predictor as a small trainable model [46], [47], [48], [49]. Neural architecture search (NAS) techniques [50], [51] are then used to *search* for hardware-friendly models.

Our work distinguishes itself from these approaches in two significant ways: 1) while existing works predominantly focus on constructing *static* models that inherently exhibit computational redundancy by treating all inputs uniformly, our goal is to design latency-aware *dynamic* models that adjust their computation based on inputs; 2) conducting speed tests for dynamic operators across various hardware devices can be laborious and impractical. To circumvent this, we propose a latency prediction model that efficiently estimates the inference latency of dynamic operators on any given computing platform. This model accounts for algorithm design, scheduling strategies, and hardware properties simultaneously, providing valuable insights without the need for extensive speed testing.

III. Method

This section begins by providing an introduction to the foundational concepts underlying three dynamic inference paradigms (Section III-A). We then present the architecture design of our LAUDNet framework, which unifies these paradigms under a cohesive *mask-and-compute* formulation (Section III-B). Next, we explain the latency prediction model (Section III-C), which guides the determination of granularity settings and scheduling optimization (Section III-D). Finally, we describe the training strategies for our LAUDNet (Section III-E).

A. Preliminaries

Spatially adaptive computation. Existing spatial-wise dynamic networks typically incorporate a masker $\mathcal{M}^s$ within each convolutional block of a CNN backbone. Given an input $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ to a block, where H and W represent the feature height and width, and C denotes the channel number.

TABLE II
OBJECT DETECTION RESULTS ON THE COCO DATASET

Detection Framework	Backbone	Backbone FLOPs (G)	Backbone Latency (ms) V100	3090	3060	TX2	Nano	mAP (%)
Faster R-CNN	ResNet-101 (Baseline)	141.2	33.9	29.8	44.8	586.4	1600.4	39.4
	LAUD^{s}-ResNet-101 (S^{net}=4-4-2-1, t=0.6)	90.7	32.4	36.8	40.4	402.2	1082.4	**40.3**
	LAUD^{s}-ResNet-101 (S^{net}=4-4-7-1, t=0.5)	79.5	30.4	29.4	38.2	390.4	1050.7	40.0
	LAUD^{s}-ResNet-101 (S^{net}=4-4-7-1, t=0.4)	**67.9**	27.4	26.2	34.5	340.0	911.4	39.5
	LAUD^{c}-ResNet-101 (C^{net}=2-2-2-2, t=0.8)	112.37	30.6	30.0	42.0	471.6	1264.3	40.2
	LAUD^{c}-ResNet-101 (C^{net}=2-2-2-2, t=0.7)	96.42	27.9	27.3	37.8	400.4	1065.4	40.0
	LAUD^{c}-ResNet-101 (C^{net}=2-2-2-2, t=0.6)	80.73	23.9	24.6	33.9	335.7	**884.0**	39.7
	LAUD^{l}-ResNet-101 (t=0.5)	97.97	24.2	22.1	32.2	409.2	1114.1	40.2
	LAUD^{l}-ResNet-101 (t=0.4)	86.71	**19.8**	**18.2**	**26.5**	**331.2**	899.9	39.5
RetinaNet	ResNet-101 (Baseline)	141.2	33.9	29.8	44.8	586.4	1600.4	38.5
	LAUD^{s}-ResNet-101 (S^{net}=4-4-2-1, t=0.5)	77.8	29.0	32.7	36.7	350.1	937.1	39.3
	LAUD^{s}-ResNet-101 (S^{net}=4-4-7-1, t=0.4)	**66.4**	28.1	26.0	35.2	335.0	897.1	38.9
	LAUD^{c}-ResNet-101 (C^{net}=2-2-2-2, t=0.6)	79.6	23.7	24.4	33.7	331.2	871.4	39.3
	LAUD^{c}-ResNet-101 (C^{net}=2-2-2-2, t=0.5)	65.5	20.9	22.1	30.4	**278.7**	**724.6**	38.5
	LAUD^{l}-ResNet-101 (t=0.5)	95.1	23.6	21.5	31.4	397.7	1082.5	**39.4**
	LAUD^{l}-ResNet-101 (t=0.3)	74.4	**18.7**	**17.3**	**25.0**	311.4	846.3	38.6
Deformable-DETR DINO-DETR Rank-DETR Stable-DINO	ResNet-50 (Baseline)	73.3	18.4	16.2	25.0	313.6	851.4	46.9 49.0 50.2 50.4
DDQ-DETR	ResNet-50 (Baseline)	73.3	18.4	16.2	25.0	313.6	851.4	51.1
	LAUD^{s}-ResNet-50 (S^{net}=8-4-7-1, t=0.6)	61.9	21.3	22.7	27.6	287.9	776.4	**51.3**
	LAUD^{s}-ResNet-50 (S^{net}=8-4-7-1, t=0.5)	56.8	19.9	20.9	25.8	264.5	711.6	51.1
	LAUD^{c}-ResNet-50 (C^{net}=2-2-2-2, t=0.7)	50.3	15.0	15.2	21.7	222.0	581.1	50.7
	LAUD^{c}-ResNet-50 (C^{net}=2-2-2-2, t=0.6)	**42.5**	**13.3**	13.8	19.6	**186.6**	**488.1**	50.5
	LAUD^{l}-ResNet-50 (t=0.5)	62.2	15.9	14.5	21.5	265.8	720.8	51.1
	LAUD^{l}-ResNet-50 (t=0.3)	54.4	13.5	**12.6**	**18.0**	226.1	614.7	50.9
	ResNet-101 (Baseline)	141.2	33.9	29.8	44.8	586.4	1600.4	51.8
	LAUD^{s}-ResNet-101 (S^{net}=4-4-2-1, t=0.5)	93.3	33.7	38.3	41.8	417.9	1125.7	**52.4**
	LAUD^{s}-ResNet-101 (S^{net}=4-4-2-1, t=0.4)	85.7	31.5	35.7	39.2	385.5	1036.7	51.9
	LAUD^{c}-ResNet-101 (C^{net}=2-2-2-2, t=0.8)	111.8	30.8	30.1	42.1	474.1	1271.8	52.3
	LAUD^{c}-ResNet-101 (C^{net}=2-2-2-2, t=0.6)	80.8	24.1	24.7	34.1	338.5	891.9	52.0
	LAUD^{c}-ResNet-101 (C^{net}=1-1-1-1, t=0.5)	**62.4**	**20.5**	**21.7**	**29.9**	**270.2**	**701.0**	51.8
	LAUD^{l}-ResNet-101 (t=0.4)	104.5	26.0	23.6	34.4	439.8	1197.8	52.2
	LAUD^{l}-ResNet-101 (t=0.3)	101.9	25.4	23.1	33.6	429.0	1168.3	51.9

The bold values refer to the best result of one column wihin the same detection/segmentation framework.

Assuming a convolution stride of 1, the masker $\mathcal{M}^{s}$ takes $\mathbf{x}$ as input and generates a binary-valued spatial mask $\mathbf{M}^{s} = \mathcal{M}^{s}(\mathbf{x}) \in \{0,1\}^{H\times W}$. Each element in $\mathbf{M}^{s}$ determines whether to perform convolution operations at the corresponding output location. Unselected regions are populated with values from skip connection [19], [20].

During inference, the current scheduling strategy for spatial-wise dynamic convolutions generally involve three steps [52] (Fig. 1(b)): 1) *gathering*, which re-organizes the selected pixels (if the convolution kernel size is greater than 1×1, the neighbors are also required) along the *batch* dimension; 2) *computation*, which performs convolution on the gathered input; and 3) *scattering*, which fills the computed pixels on their corresponding locations of the output feature. Compared to performing convolutions on the entire feature map, this scheduling strategy reduces computation at the cost of overhead from mask generation and non-contiguous *memory access*. As a result, the overall latency could even be increased, particularly when the *granularity* of dynamic convolution is at the pixel level (Fig. 6).

Dynamic layer skipping [14], [15], [53] adaptively determines whether to execute each layer or block, leveraging the structural redundancy of deep models to achieve data-dependent network *depth*. The implementation of dynamic layer skipping is similar to spatially adaptive inference, but with a scalar $0, 1$ decision variable $\mathbf{M}^{l}$ instead of a spatial $H \times W$ mask. Compared to spatially adaptive inference, layer skipping provides less flexibility but more regular computation patterns. Moreover, it generally does not require special scheduling strategies, as the original convolution operators remain unmodified.

Dynamic channel skipping [16], [17], [54] takes a more conservative approach to dynamic architecture versus full layer skipping. It uses a C-dimensional vector $\mathbf{M}^{c} \in \{0,1\}^{C}$ to adaptively determine the runtime *width* of a convolution layer with C output channels. For instance, the ith $(1 \le i \le C)$ channel is computed only if $\mathbf{M}^{c}_{i} = 1$. The scheduling of dynamic channel skipping usually requires gathering convolution kernels instead of feature pixels as in spatially dynamic computation (compare Fig. 2(b) and (c)).

TABLE III
INSTANCE SEGMENTATION RESULTS ON THE COCO DATASET

Segmentation Framework	Backbone	Backbone FLOPs (G)	Backbone Latency (ms) V100	3090	3060	TX2	Nano	AP^{mask} (%)	AP^{box} (%)
Mask R-CNN	ResNet-101 (Baseline)	141.2	33.9	29.8	44.8	586.4	1600.4	36.1	40.0
	LAUD^{s}-ResNet-101 (S^{net}=4-4-2-1, t=0.5)	80.5	29.7	33.5	37.5	361.9	969.9	**37.0**	**41.0**
	LAUD^{s}-ResNet-101 (S^{net}=4-4-2-1, t=0.4)	**69.2**	26.4	29.6	33.7	**314.3**	**838.8**	36.1	40.0
	LAUD^{c}-ResNet-101 (C^{net}=2-2-2-2, t=0.8)	112.7	30.7	30.0	42.1	473.1	1269.3	36.9	40.9
	LAUD^{c}-ResNet-101 (C^{net}=2-2-2-2, t=0.7)	95.9	27.1	27.2	37.7	397.8	1057.5	36.6	40.6
	LAUD^{c}-ResNet-101 (C^{net}=2-2-2-2, t=0.6)	80.8	23.9	24.6	33.9	335.7	883.9	36.4	40.3
	LAUD^{l}-ResNet-101 (t=0.5)	101.7	25.1	22.9	33.3	424.4	1155.7	36.7	40.7
	LAUD^{l}-ResNet-101 (t=0.4)	91.8	22.8	20.9	30.4	384.3	1045.4	36.4	40.4
	LAUD^{l}-ResNet-101 (t=0.3)	82.2	**20.6**	**18.9**	**27.5**	345.3	938.5	36.2	40.0
Mask2Former	ResNet-101 (Baseline)	141.2	33.9	29.8	44.8	586.4	1600.4	44.0	46.7
	LAUD^{s}-ResNet-101 (S^{net}=4-4-2-1, t=0.5)	109.7	37.2	43.8	47.1	481.8	1301.9	**44.0**	**47.1**
	LAUD^{s}-ResNet-101 (S^{net}=4-4-2-1, t=0.4)	98.8	34.0	39.9	43.2	436.0	1176.1	43.8	46.7
	LAUD^{c}-ResNet-101 (C^{net}=2-2-2-2, t=0.8)	111.9	30.5	29.9	41.8	469.0	1257.6	44.0	46.9
	LAUD^{c}-ResNet-101 (C^{net}=2-2-2-2, t=0.7)	**94.9**	**26.8**	27.0	37.4	**393.1**	**1044.5**	43.9	46.8
	LAUD^{l}-ResNet-101 (t=0.5)	112.8	27.7	25.1	36.6	469.4	1279.2	44.0	47.0
	LAUD^{l}-ResNet-101 (t=0.4)	109.9	27.0	**24.5**	**35.7**	457.5	1246.6	43.9	46.8

The bold values refer to the best result of one column wihin the same detection/segmentation framework.

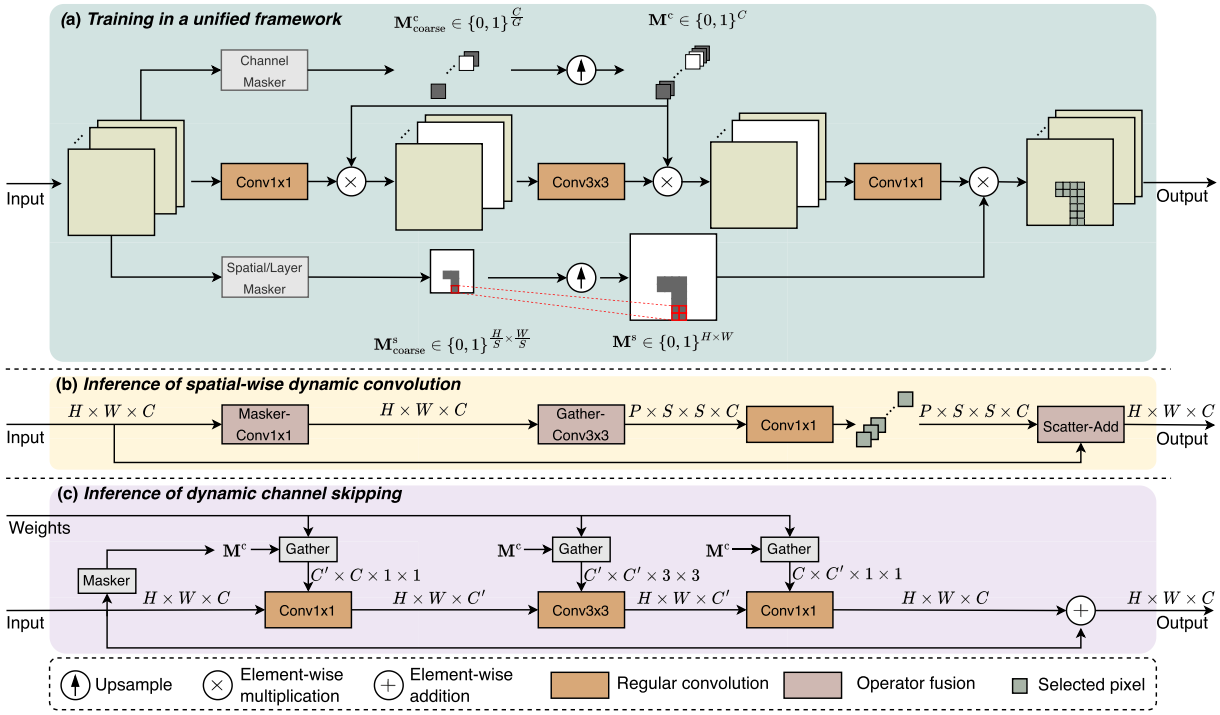


Fig. 2. Our proposed LAUDNet block. (a) we first use a lightweight module to generate the channel mask $\mathbf{M}^{c}$ or the spatial/layer mask $\mathbf{M}^{s}/\mathbf{M}^{l}$. The granularity of dynamic inference is controlled by G (for channel skipping) and S (for spatially adaptive computation). During training, the channel mask is multiplied with the input and output of the 3×3 convolution, and the spatial mask is applied on the final output of the block. Layer skipping could be easily implemented by setting S equal to the feature resolution. The scheduling strategies in inference ((b) for spatial-wise dynamic convolution and (c) for channel skipping) is performed to decrease memory access and facilitate parallel computation (Section III-D). Note that we omit layer skipping here due to its simplicity: the whole block will be executed if the layer masker produces a value of 1.

B. LAUDNet architecture

Overview. Our analysis in Section III-A reveals that the three dynamic inference paradigms share a common "*mask-and-compute*" scheme, with the key difference being the *mask shapes*. Leveraging this insight, we propose a unified framework (Fig. 2) where lightweight modules generate the channel mask $\mathbf{M}^{c}$ and the spatial/layer mask $\mathbf{M}^{s/l}$, respectively. Notably, layer skipping can be treated as a special case of spatially adaptive inference by introducing the concept of *granularity* in dynamic computation as follows.

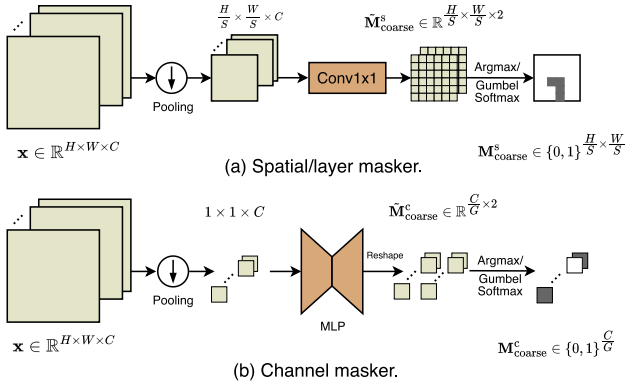


Fig. 3. The architecture design of two types of maskers. The spatial/layer masker (a) is composed of a an adaptive pooling layer and a 1×1 convolution. The channel makser (b) consists of a global average pooling and a 2-layer MLP. The argmax operation is directly applied to obtain the discrete decisions during inference, while Gumbel Softmax [55], [56] is utilized for end-to-end training (Section III-E).

Dynamic granularity. As mentioned in Section III-A, using *pixel*-level dynamic convolutions [19], [20], [21] poses substantial challenges for realistic speedup on multi-core processors due to non-contiguous memory access. To address this, we propose to optimize the *granularity* of dynamic computation. For spatially adaptive inference, instead of producing an $H \times W$ mask directly, we first generate a low-resolution mask $\mathbf{M}^{\mathrm{s}}_{\mathrm{coarse}} \in \{0,1\}^{\frac{H}{S} \times \frac{W}{S}}$, where S is the *spatial granularity*. Each element in $\mathbf{M}^{\mathrm{s}}_{\mathrm{coarse}}$ determines computation for a corresponding $S \times S$ feature patch. For instance, the first ResNet stage[2] deal with 56×56 features. Then the valid choices for S are $\{1, 2, 4, 7, 8, 14, 28, 56\}$. The mask $\mathbf{M}^{\mathrm{s}}_{\mathrm{coarse}}$ is then upsampled to the size of $H \times W$. Notably, $S = 1$ corresponds to pixel-level granularity [19], [20], [21], while $S = 56$ naturally implements layer skipping.

Similarly, we introduce channel granularity G for channel skipping. Each element in $\mathbf{M}^{\mathrm{c}}_{\mathrm{coarse}} \in \{0,1\}^{\frac{C}{G}}$ determines computation for G feature channels. The choice of the spatial granularity S and the channel granularity G for each block will be guided by our latency predictor (Section III-C) for balancing flexibility and efficiency. Note that we apply the channel mask at the first two convolution layers within a block. This design is compatible with various backbone architectures, including those with arbitrary bottleneck ratios or group convolutions [24].

Masker design. We design different structures for spatial (layer) and channel-wise dynamic computation. As shown in Fig. 3(a), the spatial masker uses an adaptive pooling layer to downsample the input $\mathbf{x}$ to the size of $\frac{H}{S} \times \frac{W}{S} \times C$, followed by a 1×1 convolution layer producing the soft logits $\tilde{\mathbf{M}}^{\mathrm{s}}_{\mathrm{coarse}} \in \mathbb{R}^{\frac{H}{S} \times \frac{W}{S} \times 2}$. For the channel masker, we use a 2-layer MLP (Fig. 3(b)) to produce channel-skipping decisions. Given input channels C the target mask dimension $D = C/G$, we set the hidden units in the MLP as $\max\{\lfloor D/16 \rfloor, 16\}$, where $\lfloor \cdot \rfloor$ denotes a round-down operation. Appendix C.1, available online, shows this design effectively reduces the latency of channel maskers, especially in late stages with more channels.

Computational complexity. We first point out that the masker FLOPs are negligible compared to the backbone convolutions. Therefore, we mainly analyse the complexity of standard convolution blocks here.

For spatially adaptive computation, We define the *activation ratio* $r^{\mathrm{s}} = \frac{\sum_{i,j} \mathbf{M}^{\mathrm{s}}_{i,j}}{H \times W} \in [0, 1]$ to denote the fraction of computed pixels. Following [20], we further compute $r^{\mathrm{s}}_{\mathrm{dil}}$ of a dilated spatial mask to represent the activation ratio of the first convolution in a block. It is observed in our experiments that $r^{\mathrm{s}}_{\mathrm{dil}}$ is generally close to r^{s}. With FLOPs F_1, F_2, F_3 for the three convolution layers, the *theoretial* speedup is $\frac{r^{\mathrm{s}}_{\mathrm{dil}} F_1 + r^{\mathrm{s}} F_2 + r^{\mathrm{s}} F_3}{F_1 + F_2 + F_3} \approx r^{\mathrm{s}}$.

For channel skipping, the *activation ratio* is $r^{\mathrm{c}} = \frac{\sum_i \mathbf{M}^{\mathrm{c}}_i}{C} \in [0, 1]$. Apply the mask before and after the 3×3 convolution makes its complexity quadratic with respect to r^{c}. The overall speedup is $\frac{r^{\mathrm{c}} F_1 + (r^{\mathrm{c}})^2 F_2 + r^{\mathrm{c}} F_3}{F_1 + F_2 + F_3} \leq r^{\mathrm{c}}$.

Generalization in Transformer architectures. It is essential to highlight that the implementation of the three dynamic paradigms–namely spatial-wise adaptive computation, dynamic channel selection, and layer skipping–is inherently more straightforward in vision Transformers compared to CNNs. These paradigms are not only more amenable to hardware considerations, requiring minimal scheduling optimization, but also benefit from the inherent structure of vision Transformers. For instance, spatial-wise dynamic computation can be efficiently executed through *token indexing and selection*, thanks to the image tokenization process in vision Transformers, thereby avoiding the complex pixel gathering required in convolution layers (Fig. 2(b)). From an algorithmic design perspective, the recent AdaViT framework [57] introduces a method for adaptively skipping tokens, heads/channels in multi-head attention, and layers, thus enabling dynamic computation across spatial, width, and depth dimensions simultaneously. However, despite theoretical comparisons presented in [57], the practical efficacy of these paradigms on hardware remains uncertain. This paper leverages the architectural design principles of AdaViT to circumvent the need for foundational redesigns and utilizes our proposed latency predictor to conduct a thorough examination of the practical performance of these dynamic paradigms in vision Transformers.

C. Latency Predictor

As stated before, it is laborious to evaluate the latency of dynamic operators on different hardware platforms. To efficiently seek preferable dynamic paradigms and granularity settings on any target device, we propose a latency prediction model $\mathcal{G}$. Given hardware properties $\mathbf{H}$, layer parameters $\mathbf{P}$, dynamic paradigm $\mathbf{D}$, spatial/channel granularity S/C, and activation rates $r^{\mathrm{s}}/r^{\mathrm{c}}$, $\mathcal{G}$ directly *predicts* block execution latency $\ell = \mathcal{G}(\mathbf{H}, \mathbf{P}, \mathbf{D}, S, C, r^{\mathrm{s}}, r^{\mathrm{c}})$.

Hardware modeling. We model a device with multiple processing engines (PEs) for parallel computation (Fig. 4). The memory system has three levels [58]: 1) off-chip memory,

[2]Here we refer to a stage as the cascading of multiple blocks which process features with the same resolution.

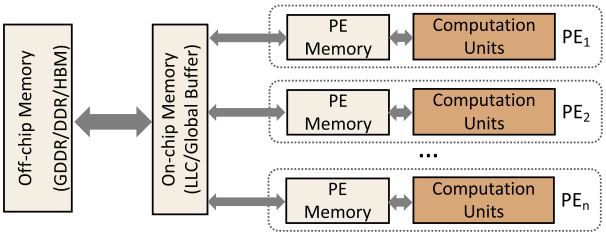


Fig. 4. Our hardware model, which allows us to model the latency of both data moving and computation.

2) on-chip global memory, and 3) memory in PE. In practice, the latency mainly comes from two processes: *data movement* and *parallel computation*

$$\ell = \ell_{\text{data}} + \ell_{\text{computation}} + \ell_{\text{Const}}, \tag{1}$$

where ℓ_{Const} is a hardware-specific constant. This model accurately predicts both ℓ_{data} and $\ell_{\text{computation}}$, enabling more practical efficiency measurement than FLOPs.

Latency prediction. Given hardware properties and model parameters, adopting a proper *scheduling strategy* is key to maximizing resource utilization through increased parallelism and reduced memory access. We use Nvidia Cutlass [28] to *search* for the optimal scheduling (tiling and in-PE parallelism configurations) of dynamic operations. The data movement latency can then be easily obtained from data shapes and target device bandwidth. Furthermore, the computation latency is derived from hardware properties. Please refer to Appendix A for more details, available online.

Empirical validation. We evaluate the performance of our latency predictor with a ResNet-101 block on an RTX3090 GPU, varying the activation rate r. The blue curves represent the predictions, and the scattered dots are obtained via *searching* for a proper scheduling strategy (implemented with custom CUDA code) using Nvidia Cutlass [28]. All the three dynamic paradigms are tested. Fig. 5 compares predictions to real GPU testing latency, showing accurate estimates across a wide range of activation rates.

D. Scheduling Optimization

We use general optimization methods like fusing activation functions and batch normalization (BN) layers into convolution layers. We also optimize our dynamic convolution blocks as follows.

Operator fusion for spatial maskers. As mentioned in Section III-B, spatial maskers have negligible computation but take the full feature map as input, making them *memory-bounded* (latency is dominated by memory access). Since the masker shares its input with the first 1×1 conv (Masker-Conv1×1 in Fig. 2(b)), fusing them avoids repeated input reads. However, this makes the convolution spatially static, potentially increasing computation. For simplicity, we adopt such operator fusion in all tested models. In practice, we find that operator fusion improves efficiency in most scenarios.

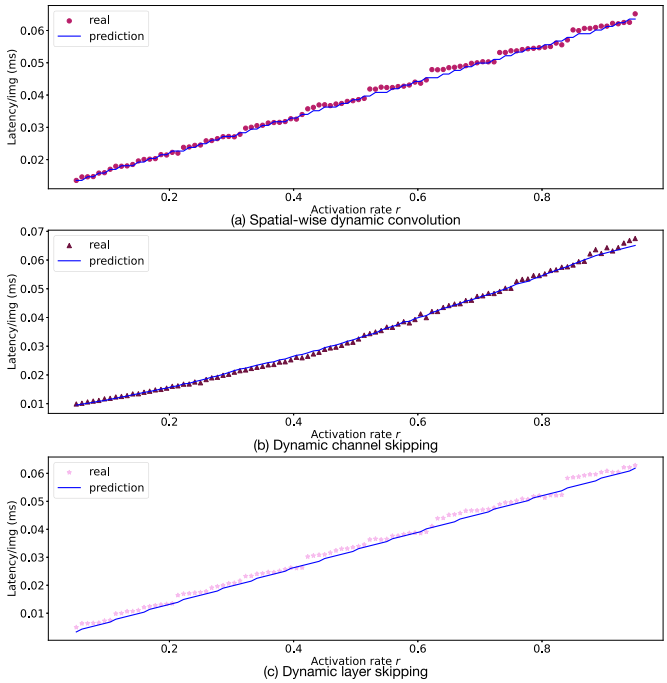


Fig. 5. Comparison between the real and predicted latency of a dynamic block in LAUD-ResNet-101.

Fusing gather and dynamic convolution. Traditional approaches first gather the input pixels of the first dynamic convolution in a block. The gather operation is also a *memory-bounded* operation. Furthermore, when the kernel size exceeds 1×1, input patches overlap, leading to repeated loads/stores. We fuse gathering into dynamic convolution to reduce the memory access (Gather-Conv3x3 in Fig. 2(b)).

Note that for dynamic channel skipping (Fig. 2(c)), gathering is conducted on convolution kernels rather than features. The weight gather operations is also fused with convolution by our scheduling optimization.

Fusing scatter and add. Conventional methods scatter the final convolution outputs before the element-wise addition. We fuse these two operators (Scatter-Add in Fig. 2(b)) to reduce memory access costs. The ablation study in Section IV-B validates the effectiveness of the proposed fusing methods.

Batching inference is enabled by recording patch, location, and sample correspondences during gathering and scattering (Fig. 2(b) and (c)). Inference with a larger batch size facilitates parallel computation, making latency more dependent on computation versus kernel launching or memory access. See Appendix C.1 for empirical analysis, available online.

E. Training

Optimization of non-differentiable maskers. The masker modules produce binary variables for discrete decisions, and cannot be directly optimized with back-propagation. Following [20], [21], [23], we adopt straight-through Gumbel Softmax [55], [56] for end-to-end training. Take spatial-wise dynamic inference as an example, let $\tilde{\mathbf{M}}^{\text{s}} \in \mathbb{R}^{H \times W \times 2}$ denote the output of the spatial mask generator $\mathcal{M}^{\text{s}}$. The decisions are obtained with the

argmax function during inference. Training uses a differentiable Softmax approximation

$$\hat{\mathbf{M}}^{\mathrm{s}} = \frac{\exp\left\{\left(\log\left(\tilde{\mathbf{M}}^{\mathrm{s}}_{:,:,0}\right) + \mathbf{G}_{:,:,0}\right)/\tau\right\}}{\sum_{k=0}^{1}\exp\left\{\left(\log\left(\tilde{\mathbf{M}}^{\mathrm{s}}_{:,:,k}\right) + \mathbf{G}_{:,:,k}\right)/\tau\right\}} \in [0,1]^{H\times W}, \tag{2}$$

where τ is the Softmax temperature. Similarly, a channel masker $\mathcal{M}^{\mathrm{c}}$ produces a $2\,C$-dimensional vector $\tilde{\mathbf{M}}^{\mathrm{c}} \in \mathbb{R}^{2\,C}$, where C is the channel number of the 3×3 convolution in a block. We first reshape $\tilde{\mathbf{M}}^{\mathrm{c}}$ into the size of $C\times 2$, and apply Gumbel Softmax along the second dimension to produce $\hat{\mathbf{M}}^{\mathrm{c}} \in [0,1]^{C}$. Following [20], [23], we let τ decay exponentially from 5.0 to 0.1 in training to facilitate the optimization of maskers.

Training objective. As analyzed in Section III-B, the FLOPs of each dynamic convolution block can be calculated based on our defined activation rate r^{s} (or r^{c}). Let F_{dyn} and F_{stat} denote the overall dynamic and static network FLOPs. We optimize their ratio to approximate a target $0<t<1$: $L_{\mathrm{FLOPs}} = (\frac{F_{\mathrm{dyn}}}{F_{\mathrm{stat}}} - t)^2$. In addition, we define L_{bounds} as in [20] to constrain the upper/lower bounds in early training epochs.

We further propose to leverage the static counterparts of our dynamic networks as "teachers" to guide the optimization procedure. Let $\mathbf{y}$ and $\mathbf{y}'$ denote the output logits of a dynamic "student" model and its static "teacher", respectively. Our final loss can be written as

$$\begin{aligned} L = {} & L_{\mathrm{task}} + \alpha(L_{\mathrm{FLOPs}} + L_{\mathrm{bounds}}) \\ & + \beta T^2 \cdot \mathrm{KL}(\sigma(\mathbf{y}/T)||\sigma(\mathbf{y}'/T)), \end{aligned} \tag{3}$$

where L_{task} represents the task-related loss, e.g., cross-entropy loss in classification. $\mathrm{KL}(\cdot||\cdot)$ denotes the Kullback–Leibler divergence, and α, β are the coefficients balancing these items. We use σ to denote the log-Softmax function, and T is the temperature for computing KL-divergence.

IV. Experiments

In this section, we first introduce the experiment settings in Section IV-A. Then the latency of different granularity settings are analyzed in Section IV-B. The performance of our LAUDNet on ImageNet is further evaluated in Section IV-C, followed by the visualization results in Section IV-D. We finally validate our method on the object detection and instance segmentation tasks (Section IV-E). For simplicity, we add "LAUD$^{\mathrm{s/c/l}}$-" as a prefix before model names to denote our LAUDNet with different dynamic paradigms (s for spatial, c for channel and l for layer), e.g., LAUD$^{\mathrm{s}}$-ResNet-50.

A. Experiment Setup

Image classification experiments are conducted on the ImageNet [1] dataset. We implement our LAUDNet on five representative architectures extending up to a broad spectrum of computational costs: four CNNs (ResNet-50, ResNet-101 [2], RegNetY-400 M, RegNetY-800M [24]) and a vision Transformer, T2T-ViT [26]. Different training settings are used for CNNs and Transformers. For CNNs, As per the established methodology in [20], we initialize the backbone parameter from a torchvision pre-trained checkpoint (https://pytorch.org/vision/stable/models.html), and finetune the whole network for 100 epochs employing the loss function in (3). We fix $\alpha=10$, $\beta=0.5$ and $T=4.0$ for all dynamic models. Note that we adopt the pretrain-fintune paradigm mainly to reduce the training cost, as Gumbel Softmax usually requires longer training for convergence. For our study on T2T-ViT, we use the same setup as described in AdaViT [57] and evaluate the efficiency of its various dynamic inference methods through our latency predictor.

Latency prediction. We evaluate our LAUDNet on various types of hardware platforms, including two server GPUs (Tesla V100 and RTX3090), a desktop GPU (RTX3060) and two edge devices (e.g., Jetson TX2 and Nvidia Nano). The major properties considered by our latency prediction model include the number of processing engines (#PE), the floating-point computation in a processing engine (#FP32), the frequency and the bandwidth. It can be observed from Table IV that server GPUs generally have a larger #PE than IoT devices. If not stated otherwise, the batch size is set as 128 for V100, RTX3090 and RTX3060 GPUs. On edge devices TX2 and Nano, tesing batch size is fixed as 1.

More details are provided in Appendix B, available online.

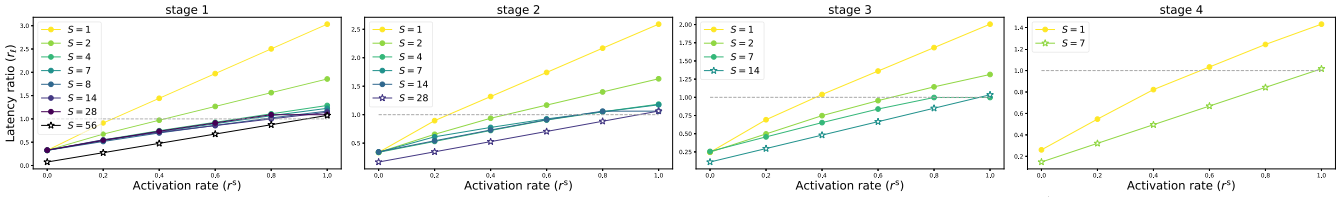
B. Latency Prediction Results

This subsection presents the latency prediction results of dynamic convolutional blocks using two distinct backbones: ResNet-50 [2] (on V100) and RegNetY-800MF [24] (on TX2). Each block features a bottleneck structure with varying channel numbers and convolution groups, and the RegNetY employs Squeeze-and-Excitation (SE) [59] modules. We define ℓ_{dyn} as the latency of a dynamic convolutional block and ℓ_{stat} as the latency of a static block. The ratio of the two is denoted as $r_\ell = \frac{\ell_{\mathrm{dyn}}}{\ell_{\mathrm{stat}}}$, with a realistic speedup being achieved when $r_\ell < 1$.

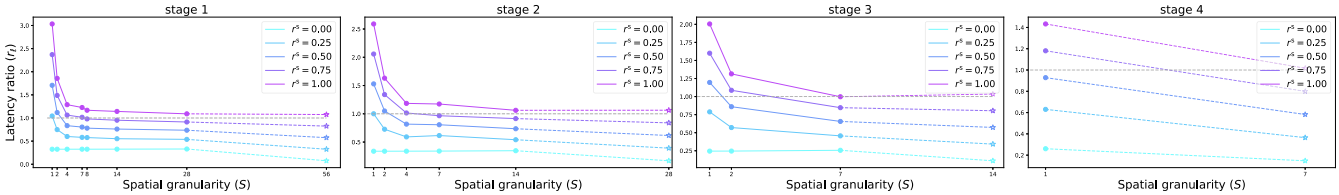
Effect of spatial granularity. The primary objective here is to investigate how the *granularity* of dynamic computation impacts the latency ratio r_ℓ. We explore the correlation between r_ℓ and the activation rate r^{s} (refer to Section III-B) for varying *granularity* settings. The results in Fig. 6(a) (ResNet on V100) and Fig. 6(c) (RegNetY-800M on TX2) demonstrate that:

- Despite the implementation of our optimized scheduling strategies, pixel-level dynamic convolution ($S=1$) does not consistently enhance practical efficiency. This approach to fine-grained adaptive inference has been adopted in previous works [20], [21], [60]. Our findings help elucidate why these studies only managed to achieve realistic speedup on less potent CPUs [21] or specialized devices [60];
- By contrast, a coarse granularity setting ($S > 1$) significantly mitigates this issue across both devices. Realistic speedup ($r_\ell < 1$) is attainable with larger activation values (r^{s}) when $S > 1$.

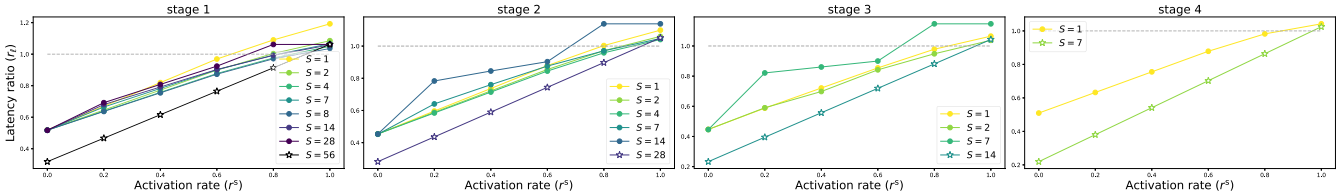
The latency prediction results are further used to determine preferable spatial granularity settings for the first 3 stages. Note that for the final stage where the feature resolution is 7×7, $S=1$ and $S=7$ correspond to two distinct dynamic paradigms (spatially adaptive inference and layer skipping). The relationship



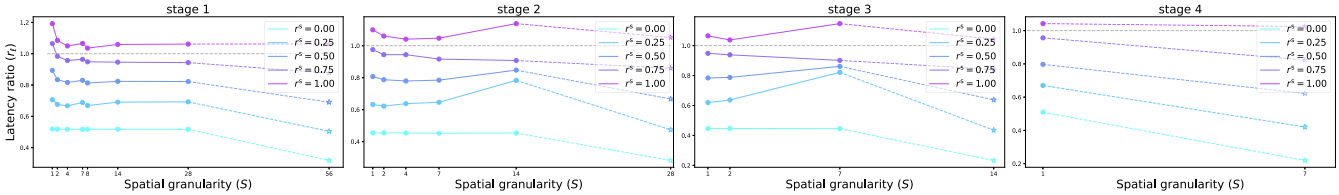
(a) Relationship between the latency ratio r_ℓ and the activation rate r^s for LAUD$^{s/1}$-ResNet.



(b) Relationship between the latency ratio r_ℓ and the spatial granularity S for LAUD$^{s/1}$-ResNet.



(c) Relationship between the latency ratio r_ℓ and the activation rate r^s for LAUD$^{s/1}$-RegNeY-800M.



(d) Relationship between the latency ratio r_ℓ and the spatial granularity S for LAUD$^{s/1}$-RegNeY-800M.

Fig. 6. Latency prediction results for LAUD$^{s/1}$-ResNet blocks on the Nvidia Tesla V100 GPU (a, b) and LAUD$^{s/1}$-RegNetY-800MF blocks on the Nvidia Jetson TX2 GPU (c, d). The circle markers (•) represent spatial-wise dynamic computation, and the star markers (⋆) denote layer skipping, which is implemented via the largest granularity S in each stage.

curves between r_ℓ and S depicted in Fig. 6(b) (ResNet on V100) and Fig. 6(d) (RegNetY-800M on TX2) reveal the following:

- The latency ratio r_ℓ generally decreases as S increases for a given r on V100;
- An excessively large S (indicating less flexible adaptive inference) provides negligible improvement on both devices. In particular, increasing S from 7 to 14 in the second stage of LAUD-RegNetY-800MF on TX2 detrimentally impacts efficiency. This is hypothesized to be due to the oversized patch size causing additional memory access costs on this device, which has fewer processing engines (PEs);
- Layer skipping (marked by ⋆) consistently outperforms spatial-wise dynamic computation (marked by •). We will analyze their performance across various vision tasks in Sections IV-C and IV-E.

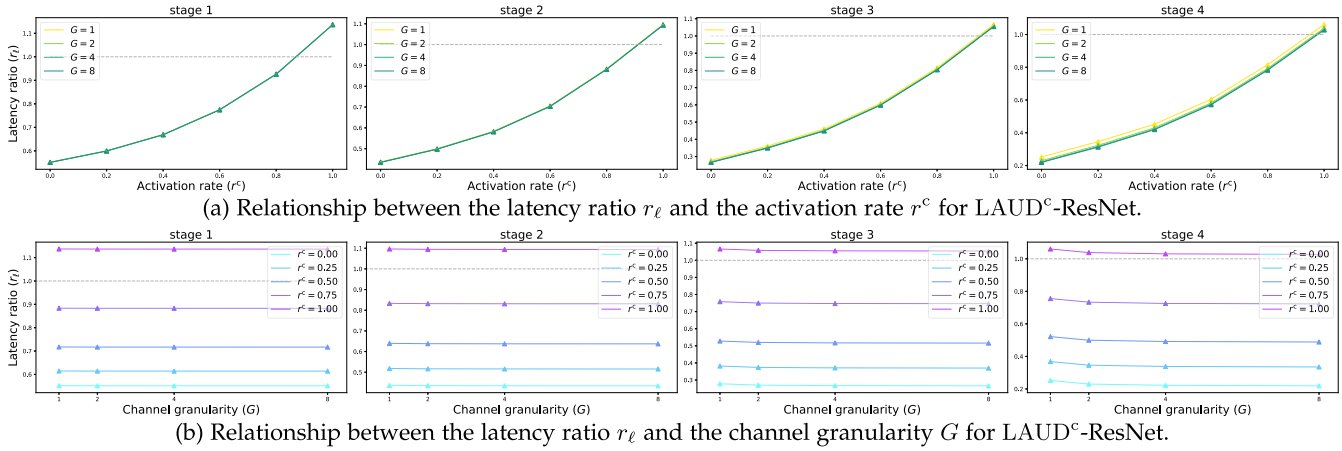
Based on these results, we can strike a balance between flexibility and efficiency by choosing suitable S for different models and devices. For instance, we can simply set $S^{\text{net}} =$ 4-4-2-1[3] in a LAUD^s-ResNet-50 to achieve realistic speedup.

Effect of channel granularity. We further investigate how the channel granularity G influences the realistic latency of channel-skipping dynamic models. Using LAUD^c-ResNet as an example, results presented in Fig. 7 show that the performance of channel skipping is less sensitive to the channel granularity G. Setting $G=2$ improves efficiency only in deeper stages, while extending G beyond 2 offers diminishing benefits. This aligns with our understanding that channel skipping requires more regular operations compared to spatially sparse convolution, implying that $G=1$ can already employ impactful speedup. Moreover, the curves in Fig. 7(a) are generally convex, since the computation of the 3×3 convolution is quadratic in relation to r^c (Section III-B).

Ablation study of operator fusion. In exploring the impact of our operator fusion, as detailed in Section III-D, we focus on a convolutional block from the initial stage of LAUD^s-ResNet-50 ($S = 4$, $r^s = 0.6$) for our case study. The findings, presented in Table I, reveal that operator fusion consistently aids in lowering the practical latency across different computing environments by minimizing memory access overhead. Notably, the fusion between the masker and the first convolution emerges as a significant factor in reducing latency on the server-end V100. In contrast, combining scattering and addition operations plays a pivotal role in latency reduction on the edge device TX2.

Ablation study of batch size. To establish a suitable testing batch size, we graph the relationship between latency per image and batch size for LAUD-ResNet-50 in Fig. 8. Two server-end

[3]We use this form to represent the S settings for the 4 network stages.



(a) Relationship between the latency ratio r_ℓ and the activation rate r^c for LAUD^c-ResNet.

(b) Relationship between the latency ratio r_ℓ and the channel granularity G for LAUD^c-ResNet.

Fig. 7. Latency prediction results for LAUD^c-ResNet blocks on the Nvidia Tesla V100 GPU.

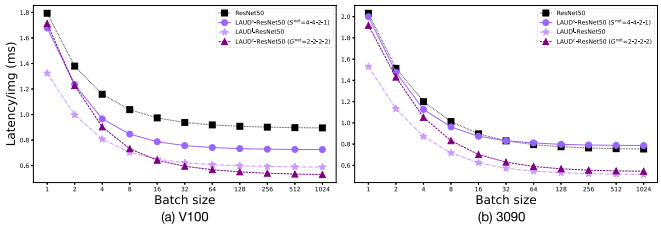


Fig. 8. Relationship between the latency per image and batch size of LAUD-ResNet-50 on V100 (a) and 3090 (b) GPUs.

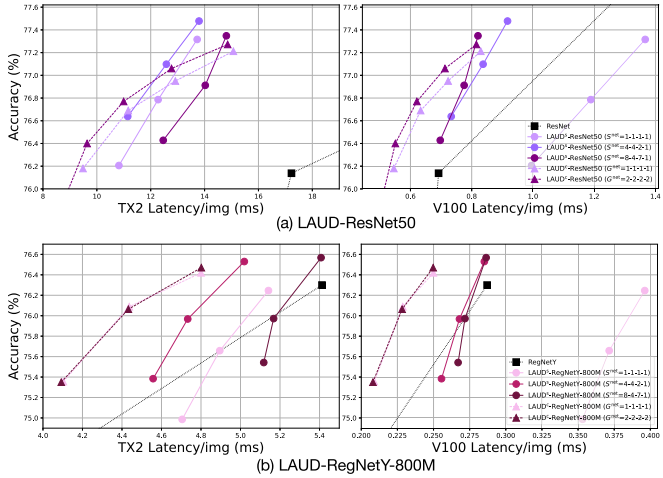


Fig. 9. Comparison of different granularities (S and G) in LAUD-ResNet-50 (a) and LAUD-RegNetY-800M (b). The latency on TX2 (left) and V100 (right) are presented.

GPUs (V100 and RTX3090) are tested. The results highlight that latency diminishes with an increase in batch size, eventually reaching a stable plateau when the batch size exceeds 128 on both platforms. This is comprehensible since a larger batch size favors enhanced computation parallelism, resulting in latency becoming more dependent on theoretical computation. The results on the desktop-level GPU, RTX3060 (Fig. 12 in Appendix C.1), available online, show a similar phenomenon. Based on these observations, we report the latency on server-end and desktop-level GPUs with a batch size of 128 henceforth.

C. ImageNet Classification

1) Comparison of Spatial/Channel Granularities: We begin by comparing different granularities for spatial and channel-wise dynamic computation. Based on the analysis in Section IV-B, the candidates for spatial and channel granularities are $S^{\text{net}} \in$ {1-1-1-1, 4-4-2-1, 8-4-7-1} and $G^{\text{net}} \in$ {1-1-1-1, 2-2-2-2} respectively. We select ResNet-50 and RegNetY-800M as backbones, and compare various settings on TX2 and V100. The results in Fig. 9 reveal that:

- Regarding spatially dynamic computation, the optimal granularity S^{net} is contingent on both network structures and hardware devices. For instance, S^{net} = 8-4-7-1 achieves a preferable performance on V100 for both models, yet incurs substantial inefficiency on TX2. This corresponds to our results in Fig. 6.
- Elevating the channel granularity G from 1 to 2 does yield sort of speedup for ResNet-50, but renders comparable performance in the case of RegNetY-800M. We hypothesize that a larger G is only beneficial for models with more extensive channel numbers, which also aligns with observations from Fig. 7.

2) Comparison of Dynamic Paradigms: Having decided on the optimal granularities, we submit different dynamic paradigms to a more detailed comparison. Additionally, our LAUDNet is compared to various competitive baselines. The findings are illustrated in Fig. 10.

Standard baseline comparison: ResNets. The compared baselines include various types of dynamic inference approaches: 1) layer skipping (SkipNet [14] and Conv-AIG [15]); 2) channel skipping (BAS [17]); and 3) pixel-level spatial-wise dynamic network (DynConv [20]). For our LAUDNet, we select the best granularity settings for spatial-wise and channel-wise dynamic inference. Layer skipping implemented in our framework is also included. We set training targets (cf. Section III-E) $t \in \{0, 4, \ldots, 0.8\}$ for our dynamic models to evaluate their performance across different sparsity regimes. We apply scheduling

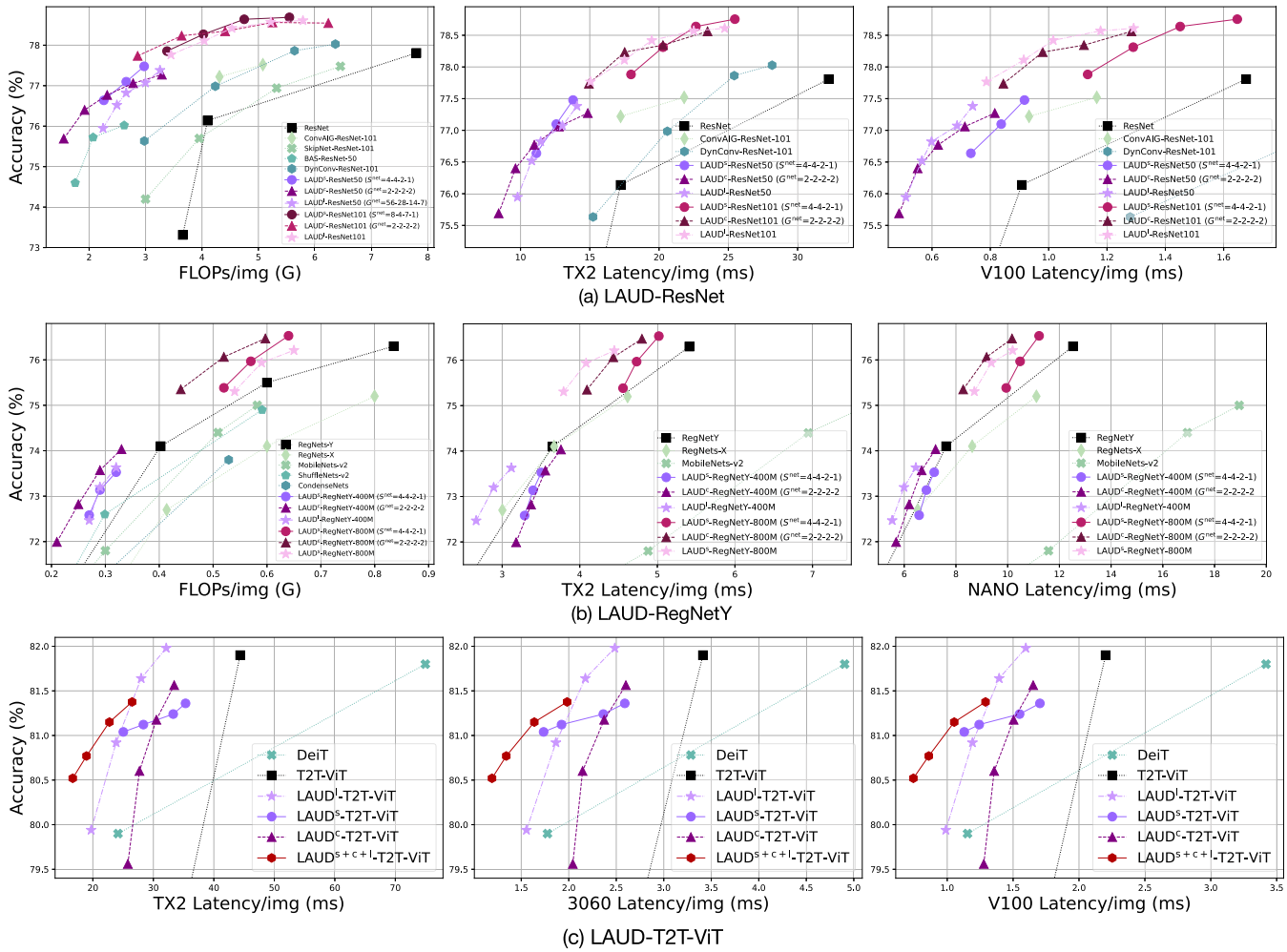


Fig. 10. Main results of LAUDNet implemented on ResNet (a), RegNetY (b) and T2T-ViT (c).

optimization (Section III-D) uniformly across all models [15], [20] for a fair comparison.

The results are exhibited in Fig. 10(a). On the left we plot the relationship between accuracy and FLOPs. It becomes obvious that our LAUD-ResNets, with various granularity settings, considerably outperform competing dynamic networks. Moreover, on ResNet-101, the three paradigms seem fairly comparable, whereas, on ResNet-50, layer skipping falls behind, especially when the training target is small. This is understandable because layer skipping might be overly aggressive for more shallow models.

Interestingly, the scenario alters as we explore real latency (middle on TX2 and right on V100). On the less potent TX2, latency generally exhibits a stronger correlation with theoretical FLOPs, given that it is *computation-bounded* (that means, the latency is primarily focused around computation) on such IoT devices. However, different dynamic paradigms yield varying acceleration impacts on server-end GPU, V100, as latency could be impacted by the memory access cost. For instance, layer skipping takes precedence over the other two paradigms on the deeper ResNet-101. With the target activation rate $t = 0.4$, our LAUD^{l}-ResNet-101 reduces the inference latency of its static counterpart by $\sim$53%. On the shallower ResNet-50, channel skipping keeps pace with layer skipping on some low-FLOPs models. Although our proposed course-grained spatially adaptive inference trails behind the other two schemes, it significantly outclasses the previous work using pixel-level dynamic computation [20]. The additional results in Appendix C.2, available online, also demonstrate the preferable efficiency of layer skipping on RTX3060 and RTX3090. Channel skipping outperforms the other two paradigms only on the edge device, Nvidia Nano.

Lightweight baseline comparison: RegNets. We further evaluate our LAUDNet in lightweight CNN architectures, i.e., RegNets-Y [24]. Two different sized models are tested: RegNetY-400MF and RegNetY-800MF. Compared baselines include other types of efficient models, e.g., MobileNets-v2 [30], ShuffletNets-v2 [32] and CondenseNets [35].

The results are presented in Fig. 10(b). We observe that while channel skipping surpasses the other two paradigms substantially in the accuracy-FLOPs trade-off, it is less efficient than layer skipping on most models except RegNet-Y-800M. Remarkably, layer skipping emerges as the most dominant

paradigm. We theorize that this is due to the model width (number of channels) of RegNet-Y being limited, and the inference latency still being bounded by memory access. Moreover, layer skipping enables skipping the memory-bounded SE operation [59]. The results on desktop-level and server-end GPUs (Appendix C.2), available online, further showcase the superiority of layer skipping.

Experiments on vision Transformers. Building on the foundation laid out in Section III-B, our LAUDNet seamlessly integrates with vision Transformers using the AdaViT [57] framework. Despite the absence of direct comparisons among the three dynamic paradigms in existing studies, with [57] employing all three simultaneously, it leaves open the question of which paradigm offers the best balance between accuracy and efficiency. We address this by showcasing the accuracy-latency trade-off curves for LAUD-T2T-ViT across various platforms–TX2, RTX3060, and V100 (the performance on RTX3090 is similar to that on V100)–in Fig. 10(c). The findings highlight several key insights:

- Layer skipping and head (channel) skipping are more advantageous for maintaining high accuracy at high activation rates, though both experience a significant accuracy decline at reduced activation rates.
- When evaluating the balance between practical latency and accuracy, layer skipping *consistently* outperforms head (channel) skipping on all platforms.
- Despite its lower theoretical upper-bound of accuracy, spatial-wise adaptive computation (token skipping) might excel over the other paradigms at lower activation rates, attributing its practical latency benefits to the straightforward implementation of indexing and selection operations on GPUs, without necessitating specialized operators as in CNNs.
- Simultaneously applying all three paradigms further enhances the accuracy-efficiency trade-off, showing the complementary strengths of each approach.

D. Visualization and Interpretability

We present visualization results of LAUDNet to delve into its interpretability from the perspectives of networks' structural redundancy and images' spatial redundancy.

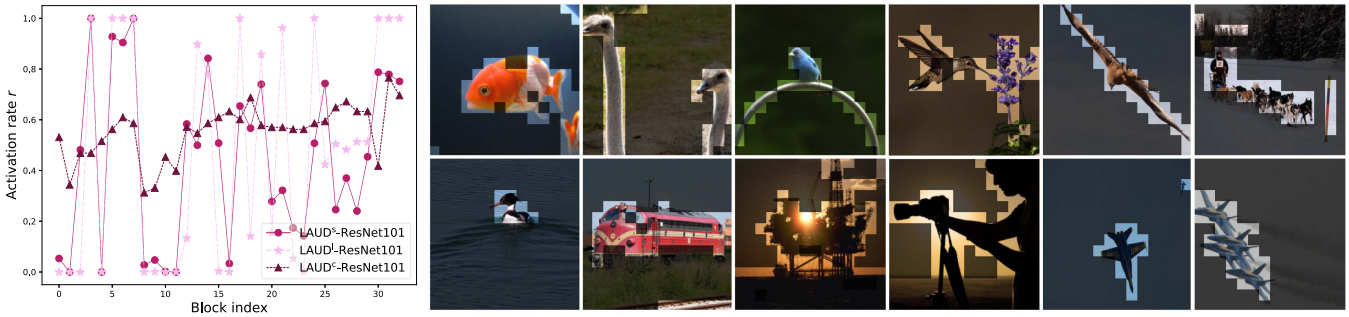
Activation rate. Fig. 11(a) illustrates the average activation rates $r^{\mathrm{s/c/l}}$ of each block in LAUD$^{\mathrm{s/c/l}}$-ResNet-101 ($t = 0.5$) on the ImageNet validation set. The results uncover that

- The activation rate patterns for spatially dynamic convolution and layer skipping are similar. The activation rates r^{s} and r^{l} seem more binarized (close to 0 or 1) in stages 1, 2, and 4. The dynamic region/layer selection predominantly occurs in stage 3;
- These two paradigms tend to maintain the entire feature map ($r^{\mathrm{s/l}} = 1.0$) at the first block of stages 2, 3, and 4, where the convolutional stride is 1. This aligns with the settings in [15], [54], where the training targets for these blocks are manually set to 1. Notably, we train our LAUDNet to meet an overall computational target, rather than confining the targets for different blocks as done in [15], [54].
- Channel skipping results in activation rates that are more centered around 0.5 throughout the network.

Dynamic patch selection. We visualize the spatial masks generated by our third block of a LAUD$^{\mathrm{s}}$-ResNet-101 ($S^{\mathrm{net}} =$ 4-4-2-1) in Fig. 11(b). The highlighted areas denote the locations of 1 elements in a mask, while computations in the dimmed regions are skipped by our dynamic model. It is evident that the masker is adept at pinpointing the most task-related areas, even minutiae such as the tiny aircraft at the corner, thereby trimming unnecessary computations in background zones. Such findings imply that, a granularity of $S = 4$ is aptly flexible for identifying crucial regions, paving the way for a harmonious balance between accuracy and efficiency. Intriguingly, the masker is able to pick out objects which are *not labeled* for that particular sample - for instance, the flower next to the hummingbird or the person clutching the camera. This signals that our spatially dynamic networks inherently discern regions imbued with semantic significance, and their prowess isn't shackled by mere classification labels. Such a trait is invaluable for a slew of downstream tasks, like object detection and instance segmentation (Section IV-E), tasks which necessitate the identification of various classes and objects within an image. For a broader range of visualization results, readers can refer to Appendix C.3, available online.

E. Dense Prediction Tasks

Our LAUDNet is further put to test on downstream tasks, i.e., COCO [61] object detection (as seen in Table II) and instance segmentation (presented in Table III). For object detection, the mean average precision (mAP) stands as the barometer for network efficacy. For instance segmentation, the AP$^{\mathrm{mask}}$ dives deeper to gauge the nuance of dense prediction. The average backbone FLOPs, and the average backbone latency on the validation set are used to measure the network efficiency. Due to LAUDNet's versatile nature, we can seamlessly replace the backbones in various detection and segmentation frameworks with our pre-trained models on ImageNet, then fine-tune them on the COCO dataset under the standard protocol for 12 epochs–except for models based on Mask2Former [62], which are trained for 50 epochs in line with the baseline configurations (detailed settings are elaborated in Appendix B.3), available online. In the domain of object detection, our experimentation covers three frameworks: the two-stage Faster R-CNN [63] with Feature Pyramid Network [64], the one-stage RetinaNet [65], and a DETR [66]-based model, namely Dense Distinct Query (DDQ)-DETR [67]. We compare our results against a range of recent advancements, such as Deformable DETR [68], DINO-DETR [69], Rank-DETR [70], and Stable-DINO [71]. For instance segmentation, we utilize the well-established Mask R-CNN [72] and the query-based Mask2Former [62]. The results are presented in Table II (for object detection) and Table III (for instance segmentation), unequivocally demonstrating that LAUDNet consistently boosts both mAP and efficiency across classic and state-of-the-art (SOTA) frameworks. Notably, while channel and layer skipping generally surpass spatial-wise dynamic computation in efficiency, the ideal dynamic paradigm



(a) Activation rates r of LAUD-ResNet101 (b) Patches selected by LAUD$^{\mathrm{s}}$-ResNet101

Fig. 11. Visualization results of activation rates $r^{\mathrm{s/c/l}}$ and selected patches by LAUD$^{\mathrm{s}}$-ResNet-101.

may vary depending on the specific detection framework, backbone architecture, and hardware platforms.

V. Conclusion

In this paper, we propose to build *latency-aware* unified dynamic networks (LAUDNet) under the guidance of a *latency prediction model*. By collectively considering the algorithm, scheduling strategy, and hardware properties, we can accurately estimate the practical latency of different dynamic operators on any computing platforms. Based on an empirical analysis of the correlation between latency and the *granularity* of spatial-wise and channel-wise adaptive inference, the algorithm and scheduling strategies are optimized to attain realistic speedup on a range of multi-core processors, such as Tesla V100 and Jetson TX2. Our experiments on image classification, object detection, and instance segmentation tasks affirm that the proposed method markedly boosts the practical efficiency of deep CNNs and surpasses numerous competing approaches. We believe our research brings useful insights into the design of dynamic networks. Future works include explorations on more types of model architectures (e.g., Transformers, large language models) and tasks (e.g., low-level vision tasks and vision-language tasks).

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[3] G. Huang, Z. Liu, G. Pleiss, K. Weinberger, and L. Van Der Maaten, "Convolutional Networks with Dense Connectivity," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 12, pp. 8704–8716, Dec. 1, 2022, doi: 10.1109/TPAMI.2019.2918284.
[4] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Representations*, 2021.
[5] A. Kirillov et al., "Segment anything," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2023, pp. 4015–4026.
[6] A. Vaswani et al., "Attention is all you need," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.
[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Hum. Lang. Technol.*, 2019, pp. 4171–4186.
[8] A. Radford et al., "Improving language understanding by generative pre-training," 2018.
[9] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.
[10] J. Achiam et al., "Gpt-4 technical report," 2023, *arXiv:2303.08774*.
[11] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 11, pp. 7436–7456, Nov. 2022.
[12] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *Proc. Int. Conf. Learn. Representations*, 2018.
[13] Y. Han et al., "Learning to weight samples for dynamic early-exiting networks," in *Proc. Eur. Conf. Comput. Vis.*, 2022, pp. 362–378.
[14] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "SkipNet: Learning dynamic routing in convolutional networks," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 420–436.
[15] A. Veit and S. Belongie, "Convolutional networks with adaptive inference graphs," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 3–18.
[16] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 2178–2188.
[17] B. E. Bejnordi, T. Blankevoort, and M. Welling, "Batch-shaping for learning conditional channel gated networks," in *Proc. Int. Conf. Learn. Representations*, 2020.
[18] M. Figurnov et al., "Spatially adaptive computation time for residual networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 1790–1799.
[19] X. Dong, J. Huang, Y. Yang, and S. Yan, "More is less: A more complicated network with less inference complexity," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 1895–1903.
[20] T. Verelst and T. Tuytelaars, "Dynamic convolutions: Exploiting spatial sparsity for faster inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 2317–2326.
[21] Z. Xie, Z. Zhang, X. Zhu, G. Huang, and S. Lin, "Spatially adaptive inference with stochastic feature sampling and interpolation," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 531–548.
[22] Y. Wang, K. Lv, R. Huang, S. Song, L. Yang, and G. Huang, "Glance and focus: A dynamic approach to reducing spatial redundancy in image classification," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.*, 2020, Art. no. 205.
[23] Y. Han, G. Huang, S. Song, L. Yang, Y. Zhang, and H. Jiang, "Spatially adaptive feature refinement for efficient inference," *IEEE Trans. Image Process.*, vol. 30, pp. 9345–9358, 2021.
[24] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 10425–10433.
[25] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *Proc. 38th Int. Conf. Mach. Learn.*, 2021, pp. 10347–10357.
[26] L. Yuan et al., "Tokens-to-token ViT: Training vision transformers from scratch on ImageNet," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 538–547.
[27] Y. Han et al., "Latency-aware spatial-wise dynamic networks," in *Proc. 36th Int. Conf. Neural Inf. Process. Syst.*, 2022, Art. no. 2670.
[28] Nvidia, "Cutlass," 2018. [Online]. Available: https://www.github.com/NVIDIA/cutlass
[29] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv: 1704.04861*.

[30] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
[31] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6848–6856.
[32] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 122–138.
[33] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proc. 4th Int. Conf. Learn. Representations*, May 2016. [Online]. Available: http://arxiv.org/abs/1510.00149
[34] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4335–4344.
[35] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger, "CondenseNet: An efficient densenet using learned group convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2752–2761.
[36] L. Yang et al., "CondenseNet V2: Sparse feature reactivation for deep networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 3568–3577.
[37] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4114–4122.
[38] J. Choi et al., "PACT: Parameterized clipping activation for quantized neural networks," 2018, *arXiv: 1805.06085*.
[39] S. Jung et al., "Learning to quantize deep networks by optimizing quantization intervals with task loss," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4345–4354.
[40] Z. Yuan, C. Xue, Y. Chen, Q. Wu, and G. Sun, "PTQ4ViT: Post-training quantization for vision transformers with twin uniform quantization," in *Proc. Eur. Conf. Comput. Vis.*, 2022, pp. 191–207.
[41] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proc. Int. Conf. Neural Inf. Process. Syst. Workshop*, 2014.
[42] C. Wang, Q. Yang, R. Huang, S. Song, and G. Huang, "Efficient knowledge distillation from model checkpoints," in *Proc. 36th Int. Conf. Neural Inf. Process. Syst.*, 2022, Art. no. 44.
[43] A. Graves, "Adaptive computation time for recurrent neural networks," 2016, *arXiv:1603.08983*.
[44] L. Yang, Y. Han, X. Chen, S. Song, J. Dai, and G. Huang, "Resolution adaptive networks for efficient inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 2366–2375.
[45] X. Gao, Y. Zhao, Ł. Dudziak, R. Mullins, and C. Z. Xu, "Dynamic channel pruning: Feature boosting and suppression," in *Proc. Int. Conf. Learn. Representations*, 2019.
[46] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *Proc. Int. Conf. Learn. Representations*, 2019.
[47] M. Tan et al., "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2820–2828.
[48] B. Wu et al., "FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 10726–10734.
[49] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *Proc. Int. Conf. Learn. Representations*, 2019.
[50] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2017.
[51] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Representations*, 2018.
[52] M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun, "SBNet: Sparse blocks network for fast inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8711–8720.
[53] Z. Wu et al., "BlockDrop: Dynamic inference paths in residual networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8817–8826.
[54] C. Herrmann, R. S. Bowen, and R. Zabih, "Channel selection using gumbel softmax," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 241–257.
[55] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *Proc. Int. Conf. Learn. Representations*, 2017.
[56] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *Proc. Int. Conf. Learn. Representations*, 2017.
[57] L. Meng et al., "AdaViT: Adaptive vision transformers for efficient image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 12299–12308.
[58] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Amsterdam, Netherlands: Elsevier, 2011.
[59] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 7132–7141.
[60] S. Colleman, T. Verelst, L. Mei, T. Tuytelaars, and M. Verhelst, "Processor architecture optimization for spatially dynamic neural networks," in *Proc. IFIP/IEEE 29th Int. Conf. Very Large Scale Integr.*, 2021, pp. 1–6.
[61] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 740–755.
[62] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, and R. Girdhar, "Masked-attention mask transformer for universal image segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 1280–1289.
[63] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 91–99.
[64] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017.
[65] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2999–3007.
[66] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 213–229.
[67] S. Zhang et al., "Dense distinct query for end-to-end object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 7329–7338.
[68] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: Deformable transformers for end-to-end object detection," in *Proc. Int. Conf. Learn. Representations*, 2020.
[69] H. Zhang et al., "DINO: DETR with improved denoising anchor boxes for end-to-end object detection," in *Proc. Int. Conf. Learn. Representations*, 2022.
[70] Y. Pu et al., "Rank-DETR for high quality object detection," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2023, pp. 16100–16113.
[71] S. Liu et al., "Detection transformer with stable matching," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2023, pp. 6491–6500.
[72] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2017, pp. 2961–2969.

Yizeng Han received the BS degree from the Department of Automation, Tsinghua University, Beijing, China, in 2018. He is currently working toward the PhD degree in control science and engineering with the Department of Automation, Institute of System Integration, Tsinghua University. His current research interests include computer vision and deep learning, especially in dynamic neural networks.

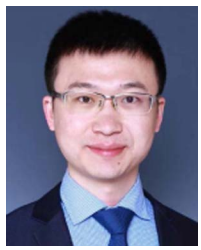
Zeyu Liu is currently working toward the undergraduate degree with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His current research interests include computer vision, deep learning and general AI.

Zhihang Yuan received the bachelor's degree from Peking University, in 2017, and the PhD degree in computer science from Peking University, in 2022. He currently focuses on AI research, with a specific interest in the compression of neural networks, and software-hardware co-optimization. In 2021, he joined Houmo AI and has since contributed to the design of AI accelerators.

Yifan Pu received the BS degree in automation from Beihang University, Beijing, China, in 2020. He is currently working toward the MS degree with the Department of Automation, Tsinghhua University, Beijing, China. His research interests include computer vision, machine learning and deep learning.

Chaofei Wang received the Ph.D. degree from the Department of Automation, Tsinghua University, Beijing, China. He is currently a Artificial Intelligence Development Engineer with Tsinghua University. His research interests include few shot learning, neural network compression, and vision generation.

Shiji Song (Senior Member, IEEE) received the PhD degree in mathematics from the Department of Mathematics, Harbin Institute of Technology, Harbin, China, in 1996. He is currently a professor with the Department of Automation, Tsinghua University, Beijing, China. He has authored more than 180 research papers. His current research interests include pattern recognition, system modeling, optimization and control.



Gao Huang (Member, IEEE) received the BS degree from the School of Automation Science and Electrical Engineering, Beihang University, in 2009, and the PhD degree from the Department of Automation, Tsinghua University, in 2015. He was was a postdoctoral researcher with the Department of Computer Science, Cornell University, Ithaca, from 2015 to 2018. He is currently an Associate Professor with the Department of Automation, Tsinghua University. His research interests include machine learning and computer vision.