SpeedDETR: Speed-aware Transformers for End-to-end Object Detection

Peiyan Dong [*1,2] **Zhenglun Kong** [*1] **Xin Meng** [*3] **Peng Zhang** [4] **Hao Tang** [†5] **Yanzhi Wang** [1]
Chih-Hsien Chou [2]

Abstract

Vision Transformers (ViTs) have continuously achieved new milestones in object detection. However, the considerable computation and memory burden compromise their efficiency and generalization of deployment on resource-constraint devices. Besides, efficient transformer-based detectors designed by existing works can hardly achieve a realistic speedup, especially on multi-core processors (e.g., GPUs). The main issue is that the current literature solely concentrates on building algorithms with minimal computation, oblivious that the practical latency can also be affected by the *memory access cost* and the *degree of parallelism*. Therefore, we propose SpeedDETR, a novel speed-aware transformer for end-to-end object detectors, achieving high-speed inference on multiple devices. Specifically, we design a latency prediction model which can directly and accurately estimate the network latency by analyzing *network properties*, *hardware memory access pattern*, and *degree of parallelism*. Following the effective local-to-global visual modeling process and the guidance of the latency prediction model, we build our hardware-oriented architecture design and develop a new family of SpeedDETR. Experiments on the MS COCO dataset show SpeedDETR outperforms current DETR-based methods by 1.5%$\sim$9.2% AP with 1.09$\times\sim$3.6$\times$ speedup on Tesla V100. Even acceptable speed inference can be achieved on edge GPUs, i.e., 4 FPS for NVIDIA JETSON TX2 (1.4$\times\sim$4$\times$faster than other counterparts), 1 FPS for NVIDIA NANO (1.5$\times\sim$6.7$\times$faster). Codes release SpeedDETR.

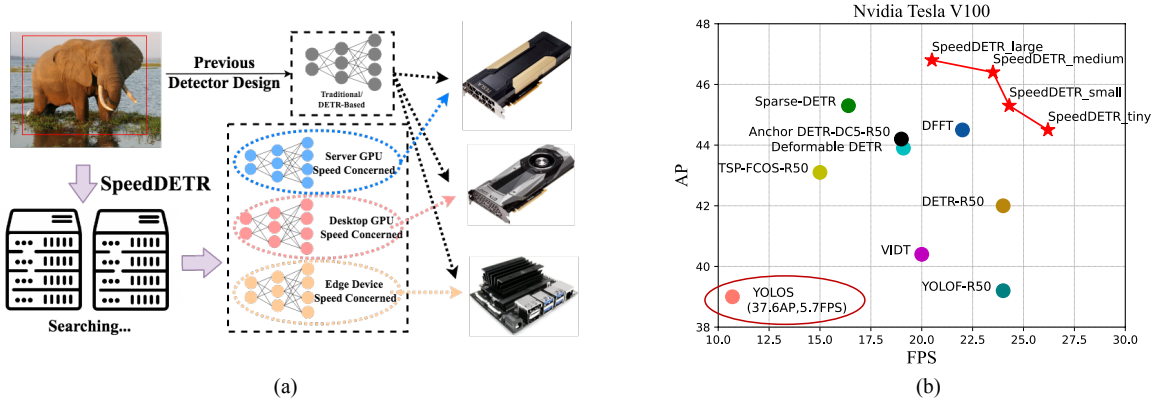
[*]Equal contribution [1]Northeastern University, Boston, MA, U.S.A [2]Futurewei Technologies, Santa Clara, CA, U.S.A [3]Peking University, Beijing, China [4]Tsinghua University, Beijing, China [5]CVL, ETH, Switzerland. Correspondence to: Hao Tang <hao.tang@vision.ee.ethz.ch>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

1. Introduction

With excellent long-range modeling capabilities, Transformers (Bahdanau et al., 2015; Parikh et al., 2016) have recently made a stunning resurgence in the form of Vision Transformers (ViTs) (Dosovitskiy et al., 2021), showing strong versatility in computer vision tasks, e.g., image classification (Dosovitskiy et al., 2021), object detection (Carion et al., 2020; Dai et al., 2022), semantic segmentation (Zheng et al., 2021), depth estimation (Yang et al., 2021a), video understanding (Zhou et al., 2018), pose estimation (Li et al., 2022a), and house generation (Tang et al., 2023). Currently, there are two main branches to apply the transformer structure in object detection. One is developing ViTs as effective backbones (Dosovitskiy et al., 2021; Wang et al., 2021b; Cao et al., 2021) in traditional frameworks such as Faster RCNN (Ren et al., 2015), Mask RCNN (He et al., 2017), and RetinaNet (Lin et al., 2017b), which constantly sets the state-of-art for the object detection task. The other one is the DETR series which utilizes an encoder-decoder transformer design that reduces object detection to an end-to-end set prediction problem. With removing components, e.g., non-maximum suppression (NMS), the DETR series can be more easily supported by a wide range of hardware platforms than other detection methods, e.g., YOLO series (Redmon et al., 2016). Thanks to its capability to process multimodal data, DETR is largely deployed as the dominant technique in 3D detection for autonomous driving (Wang et al., 2022a; Liu et al., 2022b; Li et al., 2022c).

However, to fully unleash the advantages of transformer architectures, we need to address the following challenges before the DETR series becomes an indispensable staple in future object detection systems. (i) Although the self-attention mechanism is a key defining feature of transformer architectures, a well-known concern is its computation and memory complexity (e.g., at least 200 GFLOPs in traditional frameworks; 100 GFLOPs in DETR series). This hinders scalability in many settings, let alone deployment on resource-constrained devices. (ii) Despite the impressive theoretical efficiency of some efficient designs (Zhu et al., 2020; Roh et al., 2021; Chen et al., 2022a; Song et al., 2021), researchers (Colleman et al., 2021) have found it challenging to transfer the theoretical results into real speedup, especially on multi-core processors, e.g., GPUs. The challenges



(a) (b)

Figure 1: (a) Comparison with previous works in real-world object detection. (b) The trade-off between performance (AP) and hardware efficiency (FPS) for different detection methods. SpeedDETR gets faster inference (1.09×∼3.6×) with 1.5%∼9.2% higher AP than state-of-the-art DETR-based detectors.

are two-fold: a) Most previous approaches do not have optimizations for specific hardware deployment, such as memory access cost, the degree of parallelism, and compiler characteristics, which can have a non-trivial effect on hardware throughput during inference. b) The existing literature lacks speed-aware guidance for model design but solely uses the hardware-agnostic FLOPs as an inadequate proxy for efficiency. For original self-attention operation, the quadratic memory complexity further enlarges the discrepancy between the theoretical FLOPs and the practical speed. Note that it has been validated by previous works that the latency on CPUs has a strong correlation with FLOPs (Han et al., 2021; Xie et al., 2020). Therefore, we mainly address this low hardware efficiency on the GPU platform, which is more challenging and less investigated.

In this paper, we build a novel detection method named SpeedDETR: speed-aware transformer for end-to-end object detector (see Figure 1), which achieves both higher detection precision and realistic speedup across various computing platforms (e.g., server GPUs and edge devices). Given a target device, we directly use the latency, rather than the FLOPs, to guide our detector design.

On-device efficiency. 1) Testing network latency on different device platforms is laborious due to the wide variety of candidates (transformer-based or CNN-based structures) and their associated properties, in addition to the difficulty in obtaining accurate results of tiny structures on edge devices. To this end, we introduce a novel latency prediction model to efficiently estimate the realistic latency of a network by simultaneously considering the network and hardware properties (memory access pattern and scheduling strategies/degree of parallelism). 2) According to our model profiling on high-end GPUs/low-end GPUs (NVIDIA V100/GTX 1080 Ti), we figure out and modify the inefficient building blocks by proposed fusion techniques.

Decent detection performance. 1) Following the visual modeling process from local to global, we introduce a backbone design principle with a two-phase design space. We further propose a new network pipeline consisting of a hardware-oriented backbone followed by the semantic-augmented module to enhance rich low-level semantics at the current level. 2) With the help of global attention to extract abstract-level semantics, the proposed task-coupled single-level prediction system reduces conflicts between the classification and regression tasks and provides consistent predictions. 3) Starting from a supernet with the design paradigm, we provide an effective speed-driven slimming method to obtain a new family of SpeedDETR.

Compared with current DETR works, SpeedDETR achieves 1.09×∼3.6× speedup with superior detection performance (1.5%∼9.2% higher AP) on high-end server GPUs, Nvidia V100, and 1.2×∼3.2× speedup on server low-end GPUs, GTX 1080 Ti; On edge GPUs, SpeedDETR reaches 1.4×∼4× speedup on NVIDIA JETSON TX2, and 1.5×∼6.7× speedup on NVIDIA NANO. It is worth noting that SpeedDETR is proposed as a general framework. The whole framework or the backbone design can be directly used for various downstream tasks, e.g., semantic segmentation and multimodal on-vehicle 3D detection.

Our contributions are summarized as follows:

- We propose a latency prediction model, which can efficiently estimate the latency of network candidates by considering network properties, hardware memory access pattern, and degree of parallelism.
- Combined with the effective visual modeling process and the hardware specifications, we design SpeedDETR , which builds the end-to-end speed-aware transformer-based detector by the practical latency instead of the theoretical FLOPs. To the best of our knowledge, SpeedDETR is the first framework that directly optimizes the real latency in the design phase of transformer-based detectors.

- Results on MS COCO verify that SpeedDETR can surpass other DETR-based methods in terms of precision and runtime speed on various GPU platforms.

2. Background and Related Works

Transformer-based detection frameworks. ViTs have been widely developed as powerful backbones in traditional detection frameworks such as Faster RCNN and RetinaNet. However, their backbones are usually inserted directly into the frameworks without optimizing the efficiency issues so that these detectors achieve high precision at the expense of computational efficiency, which prevents their usage in real-world applications with limited resources.

End-to-end detectors do not require sophisticated post-processing like NMS and achieve matchings between targets and candidates by the Hungarian algorithm. DETR (Carion et al., 2020) utilizes an encoder-decoder transformer framework. Since self-attention naturally has a high degree of freedom due to the lack of inductive bias, similar to convolutional layers, DETR holds on a large training cost (500 epochs). (Zhu et al., 2020) and (Wang et al., 2022b) exploit multi-scale deformable encoders or anchor points to accelerate training. Although better performance and fast convergence are achieved through these works, the computation cost has not yet been optimized (over 170 GFLOPs). And these excellent detection performances are typically obtained on high-end GPUs or servers with enormous computation costs (e.g., 170∼500 GFLOPs). Compared with them, SpeedDETR is the first work to explore the speed-aware end-to-end transformer-based detectors to achieve comparable detection precision and fast runtime inference on multiple devices while maintaining efficiency training.

Hardware-aware network design. Some existing works have considered the realistic latency at the network design stage. Some test speed directly on targeted devices and extract guidelines for hand-designed efficient models (Ma et al., 2018). Some search for fast models by the neural architecture search (NAS) technique (Tan et al., 2019). However, speed tests of our proposed structures on different hardware can be very laborious due to the large variety of candidates and the corresponding properties. In addition, it is not easy to gather accurate results of tiny structures on edge. On the contrary, our latency prediction model can directly predict the inference speed on given computing platforms.

Network Architecture Search. Architecture search aims to automate the process of designing neural network architectures. The goal is to discover high-performing architectures with minimal human intervention. One popular approach to architecture search is the Supernet method (Guo et al., 2020; Zoph et al., 2018). In this method, a single neural network is constructed to represent a large space of possible architectures. During training, the Supernet learns to assign weights to different network parts, effectively selecting a subset of the architecture to be used for a given task. This approach can significantly reduce the computational cost of searching for architectures since the Supernet can be trained once and reused for different tasks. Therefore, to design the backbone of different sizes and latency to meet the demands of specific hardware, we build a supernet to search for the desired architecture automatically.

3. Methodology

In this section, we first propose latency prediction modeling to efficiently and accurately predict the on-device latency of an architecture on the target device. Further, we describe fusion implementation techniques with hardware-efficient structures. Based on these structures and the process of vision modeling, we design the basic design paradigm of SpeedDETR. Specifically, we introduce the efficient embedding module and hardware-oriented backbone, which is also strong at feature capturing and fusing. Also, to remove the training-inefficient decoder and ensure the non-destructive recognition performance, we propose a strong transformer encoder, i.e., TSP. Finally, under the guidance of the latency prediction model, we generate a family of models, SpeedDETR, by the search algorithm (Guo et al., 2020).

3.1. Overview

This paper mainly has two methodological designs: speed-aware design on DETR and encoder-only design.

Speed-aware design on DETR. In contrast to other works (Chen et al., 2022b; Mehta & Rastegari, 2021) that primarily focus on computational requirements, our approach directly optimizes the on-device speed of the model. Our proposed approach involves two strategies: a) Latency prediction modeling: Our method accurately considers various factors such as target hardware properties, model type, model size, and data granularity. It mathematically describes computation latency and data movement latency, enabling precise prediction of the actual throughput of each layer (Figure 3). This theoretical model is compatible with general GPU architectures, making it versatile and readily usable. b) Fusion techniques and efficient design: We introduce practical model implementation optimization, combining it with robust vision modeling. We explore the design space and address the utilization of BN fusing in a transformer. Also, we propose and validate the efficient modeling of global attention through convolutional modulation.

Encoder-only design. The traditional DETR family (Zhu et al., 2020; Carion et al., 2020) typically consists of an encoder and a decoder. However, the decoder component has been identified as a factor that hampers model training

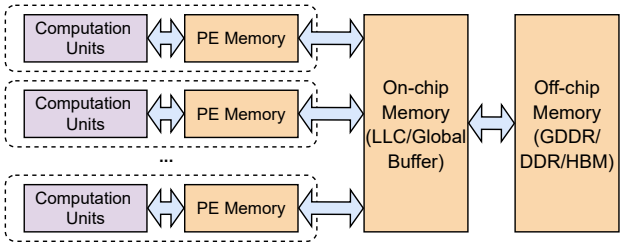


Figure 2: **Hardware Modeling**.

Table 1: **Hardware properties.**

NAME	PE	FP32	FREQUENCY(MHZ)	BANDWIDTH (G)
NVIDIA V100	80	64	1390	690
NVIDIA GTX 1080	20	64	1710	325
NVIDIA JETSON TX2	2	128	1280	58.9
NVIDIA NANO	1	128	925	26.1

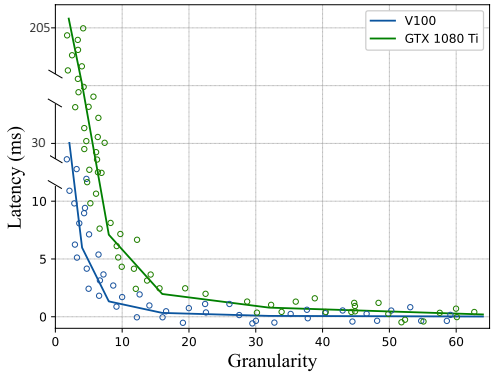


Figure 3: **Latency prediction results.**

efficiency (Sun et al., 2021; Gao et al., 2021; Zhu et al., 2020; Carion et al., 2020). To overcome the performance decline associated with removing the decoder, we propose two robust transformer encoders in the feature capturing and fusing stages (backbone Figure 5, Figure 6) and the class/box network (TSP, Figure 5). For the backbone, unlike other approaches (Liu et al., 2021b), we simultaneously enhance texture-level and abstract-level information in each stage. To reinforce texture-level semantics, we facilitate information interaction between stages by fusing the low-resolution feature from the last stage into the current feature map. In the TSP, we leverage efficient global attention to couple the classification and regression tasks, thereby reducing conflicts between them.

3.2. Latency prediction modeling

We introduce a latency prediction model E, which can directly predict the latency of runtime design choices on any target device to seek optimal model settings on any platform efficiently. For one design choice, the latency predictor E takes the hardware properties H, the layer type T, the channel dimension C, and the input granularity G as input and predicts the latency l of the block: $l = E(H, T, C, G)$.

Hardware modeling. We model a hardware device as multiple processing engines (PEs), and parallel computation can be executed on these PEs (degree of parallelism). As shown in Figure 2, we model the memory system as a three-level structure (Hennessy & Patterson, 2011): 1) off-chip memory, 2) on-chip global memory, and 3) memory in PE. Thanks to such a hardware model, we can accurately predict the latency of data movement and computation.

Latency prediction model. It can be mainly separated into three steps: 1) *Input/output shape definition.* Calculating the input and output shapes is the first step in determining an operation's latency. 2) *Operation-to-hardware mapping.* We map the operations to hardware. We have modeled a hardware device as multiple processing engines (PEs). We first consecutively split the output feature map into multiple tiles. One tile is assigned to one PE to execute the computation. 3) *Latency estimation.* We evaluate each tile's latency, including the data movement latency and the computation latency: $l{=}l_{data}{+}l_{compute}$. For the total data movement latency l_{data}, based on the hardware modeling (Figure 2), we add the input and output data movement latency together: $l_{data}{=}l_{in}{+}l_{out}$, which is estimated by the hardware bandwidth and the input granularity G (same as the resolution scale). To simplify the prediction model, we assume that each PE only moves the appropriate input feature maps and weights once to compute an output tile. For the computation latency of each tile $l_{compute}$, we use PE's maximum throughput of FP32 computation and the FLOPs of computing an output tile. The total computation latency $l_{compute}$ can be deduced by the number of tiles and PEs. We test four types of hardware devices, and their properties are listed in Table 1. It shows that the server GPU V100 is the most powerful hardware device with the most processing engines (#PE). Therefore, the computation with quadratic memory complexity, e.g., self-attention, could easily fall into a memory-bounded operation on V100 because of its high parallelism. A more detailed description of our latency prediction model is presented in Appendix A.6.

Empirical validation. We use the public tool PyTorch Profiler (Paszke et al., 2019) to measure the latency of the network structure on target devices. In Figure 3, we utilize one block (including MHSA and FFN parts) of the 1st stage in Swin-T to evaluate the effectiveness of our latency prediction model. And the resolution of the input image is (3, 224, 224). We vary the input granularity G to evaluate the performance of our prediction model. Figure 3 shows the contrast between our predictions and the actual testing latency on the Nvidia V100 GPU and GTX 1080 Ti. We can see that our predictor can accurately estimate the actual latency in a wide range of input granularities.

The latency prediction model is a training-free theoretical model specifically designed for general-purpose GPU hardware. In contrast to other works (Chen et al., 2022b; Mehta & Rastegari, 2021) that primarily focus on computation

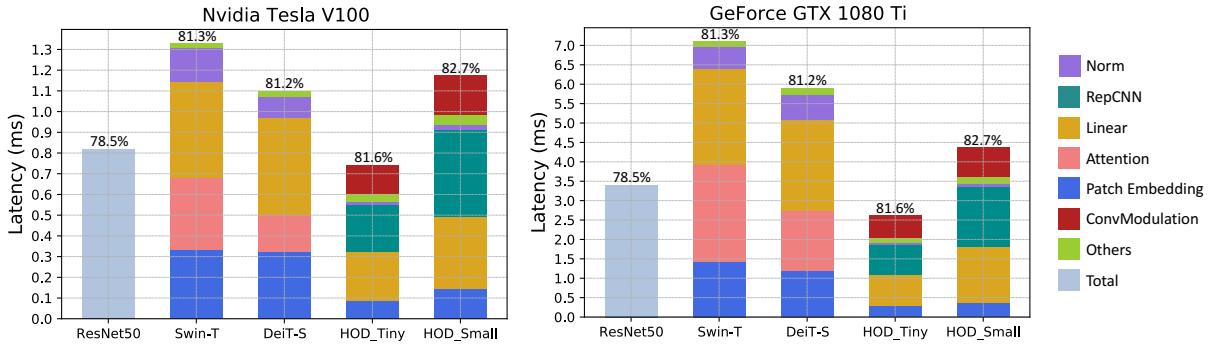


Figure 4: **Device speed profiling**. Results are obtained on NVIDIA Tesla V100 and GeForce GTX 1080 Ti. The on-device speed for frequently used backbone and various operators is reported. The accuracy is tested on the ImageNet-1K dataset (Deng et al., 2009).

amounts, our approach directly optimizes the on-device speed of the model, taking into consideration factors such as memory access cost and degree of parallelism. Moreover, our method proves to be more efficient than the commonly used technique of hardware profiling for predicting the on-device speed of networks. As a result, this modeling approach can be generalized to other networks as well. Specifically, our approach transforms all networks into fixed matrix operations on GPU platforms, utilizing techniques such as General Matrix Multiply (GeMM). The latency prediction model accurately evaluates the speed of matrix multiplication and the associated data movement, making it applicable to general networks.

3.3. Fusion techniques of model implementation

Designing and deploying efficient network architectures for resource-limited devices have significantly improved by consistently reducing parameter count and floating-point operations (FLOPs) and improving accuracy. However, these classical efficiency metrics, like FLOPs, do not consider the memory cost and the degree of parallelism. In this work, we simultaneously improve the network runtime speed and detection performance by identifying and modifying the building blocks that are not hardware-friendly. To achieve that, we deploy the common neural networks to one high-end GPU (V100) and one low-end GPU (GTX 1080 Ti) and benchmark their speed, as shown in Figure 4, whereby the following proposed fusion techniques.

Fusing batch normalization (BN) into the preceding fully-connected layer. From the analysis of various backbones compared in Figure 4, LN constitutes around 10%∼15% of the total latency for the entire network. Dynamic normalization, such as layer norm (LN), gathers running statistics at the inference stage, thus causing more speed delay. In contrast, BN is more memory-friendly/computation-friendly because we fuse BN into the preceding fully-connected layer so that the data movement and the related PE computation can be excluded. Thus, we modify the WMSA/SWMSA in Swin (Liu et al., 2021b) by replacing LN-Linear with Linear-BN, dubbed WMSA$_{bn}$/SWMSA$_{bn}$ (see Appendix A.2). Compared to the LN-based design, a 13%∼15% speedup is achieved with negligible accuracy degradation (<0.3%). Therefore, we use WMSA$_{bn}$/SWMSA$_{bn}$ as our design candidates.

Fusing multiple branches into one single branch in reparameterized CNNs. In a multi-branch structure, the data movement cost is greatly increased because the activation values of each branch are saved into PE memory or even on-chip memory (when the PE memory is not enough) to compute the subsequent tensor in the graph. Also, the synchronization cost caused by multiple branches affects the overall runtime (Hu et al., 2018). Hence, we apply RepCNN (Ding et al., 2021) as a network component, fusing multiple branches into more single-branch substructures during inference. As a result, it is easy to evenly distribute the computation among multiple PEs, preventing the imbalanced computation of PEs brought on by imbalanced computation overheads of multiple branches. Such operator fusion benefits memory access and parallel computation on multiple PEs (Detailed structures are shown in Appendix A.1).

3.4. Architecture design

In this section, we introduce SpeedDETR, a speed-aware transformer-based object detector (Figure 5). The hardware-oriented detector (HOD) backbone extracts feature at four scales and sends them to the following task-coupled single-level prediction (TSP) system. TSP first combines the multi-scale features into single-level feature maps, then adjusts feature maps to reduce conflicts between classification and regression tasks, and finalizes the detection task.

Design principle. According to the granularity of the data flow inside ResNet (He et al., 2016), we divide the backbone into four stages S. Due to feature scales with a trend from the local to the global visual receptive field, we introduce the HOD block design 6(a). Then, one semantic-augmented module is added after one HOD block (residual effect) to

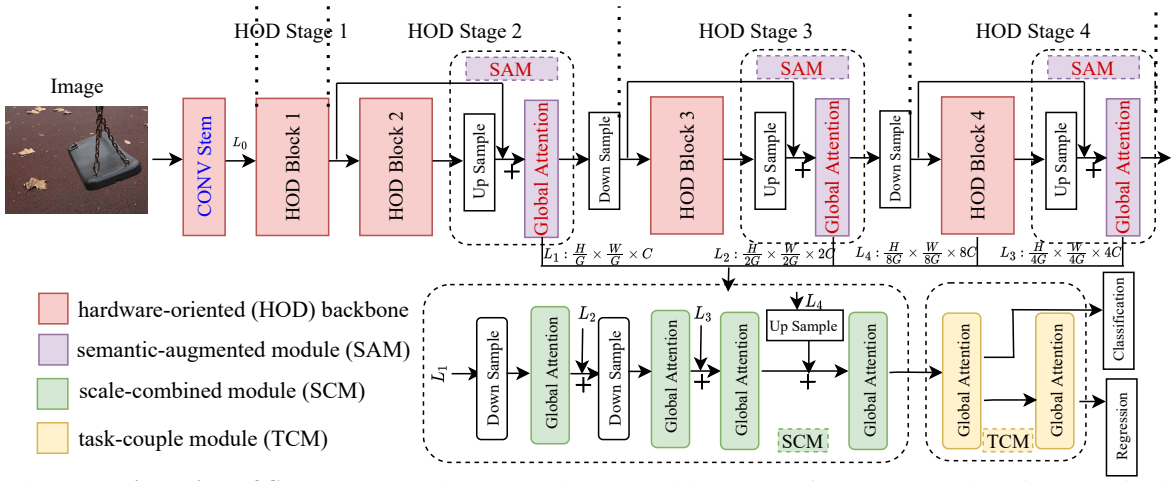


Figure 5: **The basic design of SpeedDETR.** The TCM and SCM enable us to perform accurate detection on a single-level feature map. We also propose a HOD backbone to maintain detection precision while improving the on-device speed. G is the granularity and C is the channel-wise dimension of the feature map. G=8 for V100.

enhance the low-level semantic information in each HOD stage. We provide the *two-phase design space* (DP) of HOD backbone as Figure 6(a):

$$\begin{aligned} DP^{1}_{i,local,s=1,2,3,} &\in \{RepCNN^{i}, WMSA^{i}_{bn}, SWMSA^{i}_{bn}\}, \\ DP^{2}_{i,local,s=4,} &\in \{WMSA^{i}_{bn}, SWMSA^{i}_{bn}\}, \\ DP^{1,2}_{i,glocal} &\in \{ConvModula\}, \end{aligned} \tag{1}$$

where *local* represents the candidates of local-wise attention while *global* for the candidates of global attention; the 1st phase covers S_1, S_2, S_3 of the backbone, and the $2nd$ phase for S_4; i denotes the i^{th} block; instead of calculating the similarity score matrix (sttention matrix (Vaswani et al., 2017)), we simplify self-attention by modulating the value V with convolutional features as Figure 6(b). Our approach uses convolutional modulation rather than self-attention to build relationships since they are more memory-efficient, particularly when processing high-resolution images. More details for convolutional modulation are shown in Appendix A.3.

Image embedding module. Figure 4 demonstrates that patch embedding is a speed bottleneck on multiple platforms. This is because patch embedding often employs a convolutional layer that is non-overlapping and has a large kernel size and stride, which is not well supported by most compilers and acceleration techniques (e.g., Winograd). Thus, we leverage a convolution stem with fast downsampling, consisting of three hardware-efficient 3×3 convolutions with stride 2. For an input image $x \in \mathbb{R}^{H\times W\times 3}$, we first divide it into $\frac{H\times W}{G\times G}$ patches and feed these patches to the convolution stem to obtain input embeddings L_0 of size $\frac{H}{G} \times \frac{W}{G} \times C$.

Hardware-oriented detector backbone. Each HOD block (Figure 6(a)) is designed to effectively capture the local (texture-level semantics) and global information (abstract-level semantics). So several consecutive local-wise attention is applied to extract texture-level semantics. Then we enhance abstract-level semantics in the feature map through global attention. A semantic-augmented module, which includes an upsampling layer and global attention, is inserted into every two consecutive HOB blocks (except between Stage 1,2) to further enhance low-level semantics in the current stage as shown in Figure 5. Note that for hardware efficiency, we introduce RepCNN into the design space of local-wise attention, which can also act as a global interconnection, reducing the number of SWMSA$_{bn}$, the speed bottleneck for GPU structure (Pan et al., 2022).

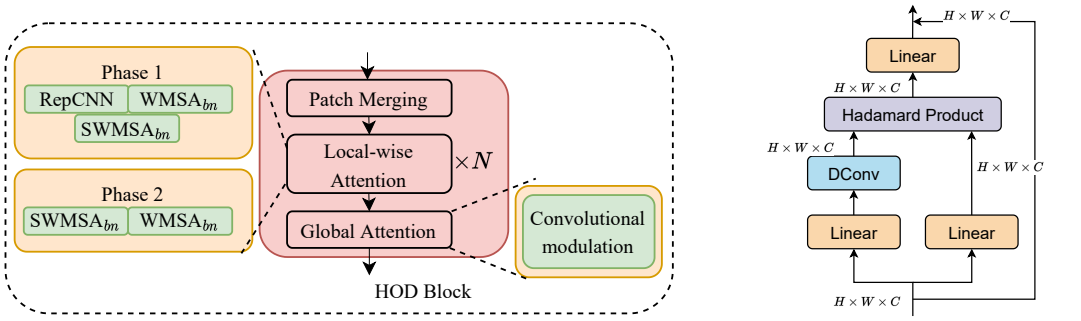
Task-coupled single-level prediction system. To reduce the large memory I/o overhead and the high parallelism caused by multi-branch, we transform multi-level into single-level prediction by *scale-combined module* (Figure 6(b)) with global attention. The process is

$$\begin{aligned} s_0 &= L_1, \\ s_1 &= ConvModula(down(s_0) + L_2), \\ s_2 &= ConvModula(down(s_1) + L_3), \\ s_3 &= ConvModula(s_2 + up(L_4)), \end{aligned} \tag{2}$$

s_{out}=s_3 is the final aggregated feature.

One-stage detectors perform object classification and localization independently with two separate branches (e.g., decoupled heads). We propose *task-couple module* in Figure 6(c), which guides learning task interactions and eliminates conflicts between task-specific features by stacking global attention blocks Y=$ConvModula$. The global attention block Y_1 couples and splits the single-level feature s_{out} into two parts. After that, Y_2 encodes one part for the subsequent regression task:

$$y^{cls}, y^{reg}_1 = Y_1(s_{out}), \quad y^{reg}_2 = Y_2(y^{reg}_1). \tag{3}$$



(a) HOD block and two-phase design space (b) Convolutional modulation

Figure 6: **Some sub-structures deployed in the SpeedDETR.**

3.5. Training

Supernet Design. We use the *two-phase design space* (DP) as the search space and train the supernet for the HOD backbone. We only search the backbone's structures, dimensions C, and input granularity G, while the TSP uses fixed structures with dimensions adapted to the backbone.

Speed-aware model slimming. It has three steps:

1) We train the supernet with the Gumble Softmax sampling (Liu et al., 2018) to get the importance score for the blocks within each DP.

2) We utilize the latency prediction model E (Section 3.2) to estimate the on-device speed of each candidate.

3) We perform speed-aware model slimming on the supernet obtained from step 1) by FPS evaluated with predictor E. Specifically, we use the score r of each candidates to define the importance score of DP_i as $\frac{r_i^{RepCNN}+r_i^{WMSA}}{r_i^{SWMSA}}$ for S_1, S_2, S_3, and $\frac{r_i^{WMSA}}{r_i^{SWMSA}}$ for S_4. We obtain the importance score for each S by summing up the scores for all DP within that S. Then we define the evolution process (all performed in the current least important S): a) remove the first $SWMSA$; b) remove the first $WMSA$; c) reduce the width by multiples of 16. Then we predict the current latency l, evaluate the accuracy drop of each evolution, and decide by $latency\_AP\_drop(-\% * l)$. Our gradual slimming on the single-width supernet is memory-efficient compared to the multiple-width supernet. This process is repeated until reaching target throughput (Appendix A.4).

4. Experiments

4.1. Settings

Object detection. We evaluate our proposed SpeedDETR on the challenging MS COCO benchmark (Lin et al., 2014) following the commonly used setting (Chen et al., 2021). The standard mean average precision (AP) metric is used to measure detection under different IoU thresholds and object scales. It contains around 160K images of 80 categories. We train SpeedDETR with the standard $1\times$ (12 epochs) and $3\times$ (36 epochs) training configurations as introduced in (Cao et al., 2021). The HOD backbone is pre-trained on ImageNet (Deng et al., 2009) with the same setting as (Cao et al., 2021). We use the AdamW optimizer with a batch size of 32, an initial learning rate of $1e{-}4$, and a weight decay of 0.05. The learning rate is stepped down by a factor of 0.1 at the 67% and 89% of training epochs. We conduct all experiments on 8 V100 GPUs. The model configurations are provided in the Appendix A.7. Since SpeedDETR conducts the single-level dense prediction on a single feature map, we use the uniform matching strategy proposed by YOLOF (Chen et al., 2021) to ensure that all ground-truth boxes match with the same number of positive anchors regardless of their sizes.

Latency prediction. We test the on-device efficiency for four hardware platforms as shown in Table 1, including server GPU (Tesla V100), desktop GPU (GTX1080), and edge devices (Jetson TX2 and Nvidia Nano). The number of PE, the floating-point computation (FP32) in a PE, the frequency, and the bandwidth are the main factors considered by our latency prediction model. The PE of server GPUs is usually larger than that of IoT devices. For all models and computing platforms, the batch size was set to 1.

4.2. Main results

Comparison with conventional frameworks. As shown in Part (1&2) of Table 2, anchor-based methods converge fast within only 12 epochs, and the transformer-based methods generally outperform CNN-based methods but with higher GFLOPs. Focal-Tiny-RetinaNet achieves 7.8% higher AP than the original RetinaNet at the expense of 32% computation costs. Generally, such good performance comes with over 170 GFLOPs high computational costs. YOLOF is an efficient transformer-based one-stage method with 103 GFLOPs, but with lower AP (<40%). MobileFormer aims at an efficient design, which concentrates on the FLOPs reduction other than the real runtime speed.

Table 2: Comparison of SpeedDETR and common detection methods on MS COCO benchmark. The table is divided into four parts: 1) anchor-based methods; 2) SpeedDETR trained for 12 epochs; 3) DETR-based methods; 4) SpeedDETR trained for 36 epochs. SpeedDETR achieves competitive precision with fewer inference FPS and training epochs. FPS is measured with a batch size 1 of 800×1333 resolution on a single Tesla V100 GPU.

Method	Epochs	AP (%)	AP_{50} (%)	AP_{75} (%)	AP_S (%)	AP_M (%)	AP_L (%)	GFLOPs	FPS
Faster RCNN-FPN-R50 (Ren et al., 2015)	36	40.2	61.0	43.8	24.2	43.5	52	180	-
Focal-Tiny-RetinaNet (Yang et al., 2021b)	12	43.7	-	-	-	-	-	265	-
Swin-Tiny-RetinaNet (Liu et al., 2021b)	12	42.0	-	-	-	-	-	245	-
YOLOF-R50 (Chen et al., 2021)	12	39.2	58.6	42.7	22.3	43.9	50.8	103	24
RetinaNet (Lin et al., 2017b)	12	35.9	55.7	38.5	19.4	39.5	48.2	201	-
Mobile-Former (Chen et al., 2022b)	12	34.2	53.4	36.0	19.9	36.8	45.3	322	-
SpeedDETR$_{tiny}$	12	41.2	60.2	43.5	21.4	44.6	55.7	67	**26.2**
SpeedDETR$_{small}$	12	42.0	59.5	44.5	23.2	45.6	56.3	69	24.3
SpeedDETR$_{medium}$	12	**43.2**	**60.9**	**46.2**	**23.9**	**47.1**	**58.7**	73	23.5
DETR-R50 (Carion et al., 2020)	500	42.0	62.4	44.2	20.5	45.8	61.1	86	24
WB-DETR (Liu et al., 2021a)	500	41.8	63.2	44.8	19.4	45.1	62.4	98	-
PnP-DETR (Wang et al., 2021a)	500	41.8	62.1	44.4	21.2	45.3	60.8	-	-
UP-DETR (Dai et al., 2021)	150	40.5	60.8	42.6	19.0	44.4	60.0	-	-
YOLOS (Fang et al., 2021)	150	37.6	-	-	-	-	-	172	5.7
Anchor DETR-DC5-R50 (Wang et al., 2022b)	50	44.2	64.7	47.5	24.7	48.2	60.6	151	19
Deformable DETR (Zhu et al., 2020)	50	43.9	62.6	47.7	26.4	47.1	58.0	173	19.1
SMCA-R50 (Gao et al., 2021)	50	43.7	63.6	47.2	24.2	47.0	60.4	152	-
SAM-DETR-R50-DC5 (Zhang et al., 2022)	50	43.3	64.4	46.2	25.1	46.9	61.0	210	-
TSP-FCOS-R50 (Sun et al., 2021)	36	43.1	62.3	47.0	26.6	46.8	55.9	189	15
DAB-DETR-R50 (Liu et al., 2022a)	50	42.6	63.2	45.6	21.8	46.2	61.1	100	-
Conditional DETR-R50 (Meng et al., 2021)	50	40.9	61.8	43.3	20.8	44.6	59.2	90	-
VIDT (Song et al., 2021)	50	40.4	59.6	43.3	23.2	42.5	55.8	-	20
Sparse-DETR (Roh et al., 2021)	50	45.3	65.8	49.3	28.4	48.3	60.1	105	16.4
DFFT (Chen et al., 2022a)	36	44.5	63.6	48	24.5	49.0	60.7	**62**	22
SpeedDETR$_{tiny}$	36	44.5	64.9	47.8	24.6	48.1	59.9	67	**26.2**
SpeedDETR$_{small}$	36	45.3	64.2	48	25	48.9	60.5	69	24.3
SpeedDETR$_{medium}$	36	46.4	65.5	49.8	26.1	50.5	63.0	73	23.5
SpeedDETR$_{large}$	36	**46.8**	**66.2**	**50.4**	**28.5**	**50.6**	**63.2**	108	20.5

Compared to the aforementioned approaches, our SpeedDETR can improve both on-device speed and detection performance. SpeedDETR$_{tiny}$ is 9% faster than YOLOF with 1% higher AP (41.2%>39.2%); SpeedDETR$_{medium}$ reduces 192 GFLOPs from the best-precision Focal-Tiny-RetinaNet at only a 0.5% AP reduction.

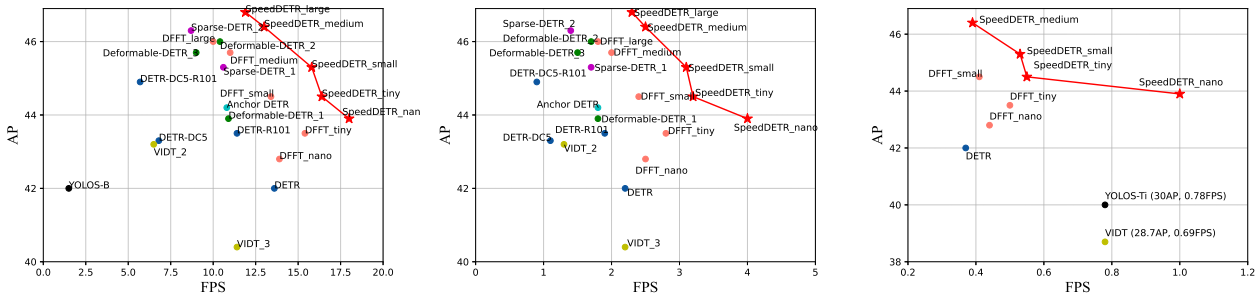
Comparison with DETR series. As shown in Part (3&4) of Table 2, DETR only needs 24 FPS to achieve 42.0% AP during inference but requires 500 epochs to converge. WB-DETR and PnP-DETR need 500 epochs to achieve 41.8% AP as well. Other DETR-based methods improve the convergence speed but less efficient. For example, the GFLOPs of Deformable DETR, Anchor DETR, and TSP-FCOS are around 1.7×∼2.2× larger than DETR. And even though these three have 15∼19 FPS on the high-end GPU, Tesla V100, they cannot achieve acceptable speeds on resource-constraint devices, e.g., Nvidia Nano (Figure 7). On the contrary, SpeedDETR can achieve superior detection precision, sacrificing neither convergence nor inference speed. Compared with classical DETR, our SpeedDETR$_{tiny}$ achieves superior detection performance (42.0% vs. 44.5%). with 2.2 FPS faster inference and 14× training efficiency. Considering current state-of-the-art DETR frameworks, our model can reach 1.09×∼3.6× speedup with 1.5%∼9.2% AP higher precision.

4.3. Runtime speed on multiple devices

To evaluate the hardware throughput, we implement the model on the other three GPU platforms (Table 1). We report the average FPS of over 100 inferences. As shown in Figure 7, our method outperforms existing efficient DETR frameworks on both hardware efficiency and detection performance. Note that SpeedDETR even achieves acceptable inference speed on edge GPU devices, i.e., 2.3 FPS ∼ 4 FPS for NVIDIA JETSON TX2 and 1 FPS for NVIDIA NANO. Other methods need to consider the memory limitations and parallelism of the on-device runtime, so their performance can be degraded further on resource-limited devices and do not even give results in a reasonable time. This also constrains their scalability for practical application scenarios. Because of the low speed, Deformable-DETR, Sparse-DETR, and Anchor DETR cannot be tested accurately on the NVIDIA NANO (26.1 Gbps memory bandwidth), which has the highest memory I/O limitation. The possible reason is that the backbone of these works has a quadratic memory complexity, which can become the computation-bound operation in some memory-limited devices.

4.4. Ablation study

We conduct ablation studies to validate the effectiveness of the *latency prediction modeling* (Section 3.2, Section 3.3)



(a) SpeedDETR on GeForce GTX 1080 Ti. (b) SpeedDETR on Jetson TX2. (c) SpeedDETR on Nvidia Nano.

Figure 7: Comparison on multiple resource-limited devices of SpeedDETR.

Table 3: Major components.

HOD	SCM	TCM	AP (%)	GFLOPs
-	-	-	33.5	**50**
✓	-	-	37.8	51
✓	✓	-	39.7	63
✓	-	✓	39.6	56
✓	✓	✓	**41.2**	67

Table 4: Operator fusion.

BN Fusion	Branch Fusion	Latency
-	-	38.1
✓	-	32.8
-	✓	30.6
✓	✓	**26.2**

and *architecture design* (Section 3.4). We conduct all the experiments on SpeedDETR$_{tiny}$ that is trained for 12 epochs.

4.4.1. Analysis of Prediction Modeling

More granularities settings. We test various granularity settings on the HOD backbone of SpeedDETR$_{tiny}$ to examine the effects of G. The results on the Tesla-V100 GPU are presented in Figure 8. We increase G from 2 and 64, and this procedure consistently improves the realistic efficiency of the V100 GPU. It can be found that the finest granularity (G=2) causes substantial inefficiency despite the mAP improvement. Otherwise, the coarsest granularity (G=64) benefits the speedup with large detection precision degradation. This trend also holds on the GTX 1080 Ti, Jetson TX2, and Nvidia Nano.

Fusion techniques. We investigate the effect of our model fusion introduced in Section 3.3. SpeedDETR$_{tiny}$ (G=8) is tested. The results in Table 4 present that each technique of model fusion promotes the practical latency of a block, as the overhead on memory access is effectively reduced. Moreover, branch merging can also reduce the imbalanced computation of PEs.

4.4.2. Analysis of Components in SpeedDETR

Major components. We evaluate the efficiency of the major components in SpeedDETR. We disable each component by replacing it with a vanilla method, as they are not easily removable from our detection framework. In Table 3, we (1) use Swin-Transformer instead of our hardware-oriented detector (HOD) backbone with similar GFLOPs (line 1 vs. line 2); (2) disable the scale-combined module (SCM) (lines 1, 2, 4) by directly upsampling the last stage's outputs to $H\times W$ and feed them to task-couple module (TCM); (3) replace the TAE module with YOLOF's head (lines 1–3).

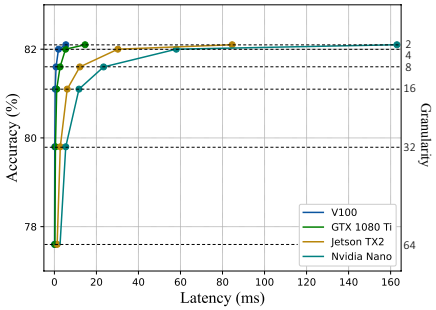


Figure 8: **Various granularity settings on the HOD backbone of SpeedDETR.**

Firstly, the HOD backbone increases precision from 33.5% to 37.8%, indicating that it can acquire stronger semantic features that are more suited for the detection task. SCM further improves the precision to 39.7% by aggregating multi-scale features into one feature map. Disabling SCM would decrease the precision by 1.6% (39.8% vs. 41.2%). Finally, adding TCM would increase the prevision by 1.5% (41.2% vs. 39.7%). This verifies the necessity of using TCM to align and encode both the classification and regression features. This also suggests that the SCM can capture multi-scale information when aggregating semantic information.

5. Conclusion

In this paper, we propose a novel speed-aware transformer for end-to-end object detectors, achieving high-speed inference on multiple devices. Combined with the visual modeling process and the proposed latency prediction model, SpeedDETR exceeds current DETR-based methods by 1.5%∼9.2% AP with 1.09×∼3.6× speedup on high-end GPU. Compared to other approaches, SpeedDETR has more potential for practical applications due to its scalability on resource-limited devices. How to improve the precision of transformer-based detectors on edge will be our next effort.

Acknowledgements

This work was supported by Futurewei Technologies, Inc. in a research internship.

References

Bahdanau, D., Cho, K. H., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

Bolya, D., Zhou, C., Xiao, F., and Lee, Y. J. Yolact: Real-time instance segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9157–9166, 2019.

Cao, H., Wang, Y., Chen, J., Jiang, D., Zhang, X., Tian, Q., and Wang, M. Swin-unet: Unet-like pure transformer for medical image segmentation. *arXiv preprint arXiv:2105.05537*, 2021.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. End-to-end object detection with transformers. In *European Conference on Computer Vision (ECCV)*, pp. 213–229. Springer, 2020.

Chen, P., Zhang, M., Shen, Y., Sheng, K., Gao, Y., Sun, X., Li, K., and Shen, C. Efficient decoder-free object detection with transformers. *arXiv preprint arXiv:2206.06829*, 2022a.

Chen, Q., Wang, Y., Yang, T., Zhang, X., Cheng, J., and Sun, J. You only look one-level feature. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13039–13048, 2021.

Chen, Y., Dai, X., Chen, D., Liu, M., Dong, X., Yuan, L., and Liu, Z. Mobile-former: Bridging mobilenet and transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5270–5279, 2022b.

Colleman, S., Verelst, T., Mei, L., Tuytelaars, T., and Verhelst, M. Processor architecture optimization for spatially dynamic neural networks. In *2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 1–6. IEEE, 2021.

Dai, L., Liu, H., Tang, H., Wu, Z., and Song, P. Ao2-detr: Arbitrary-oriented object detection transformer. *IEEE Transactions on Circuits and Systems for Video Technology*, 2022.

Dai, Z., Cai, B., Lin, Y., and Chen, J. Up-detr: Unsupervised pre-training for object detection with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1601–1610, 2021.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., and Sun, J. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13733–13742, 2021.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

Fang, Y., Liao, B., Wang, X., Fang, J., Qi, J., Wu, R., Niu, J., and Liu, W. You only look at one sequence: Rethinking transformer in vision through object detection. *Advances in Neural Information Processing Systems*, 34:26183–26197, 2021.

Gao, P., Zheng, M., Wang, X., Dai, J., and Li, H. Fast convergence of detr with spatially modulated co-attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3621–3630, 2021.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., and Sun, J. Single path one-shot neural architecture search with uniform sampling. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*, pp. 544–560. Springer, 2020.

Han, Y., Huang, G., Song, S., Yang, L., Zhang, Y., and Jiang, H. Spatially adaptive feature refinement for efficient inference. *IEEE Transactions on Image Processing*, 30: 9345–9358, 2021.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

Hennessy, J. L. and Patterson, D. A. *Computer architecture: a quantitative approach*. Elsevier, 2011.

Hou, Q., Lu, C.-Z., Cheng, M.-M., and Feng, J. Conv2former: A simple transformer-style convnet for visual recognition. *arXiv preprint arXiv:2211.11943*, 2022.

Hu, J., Shen, L., and Sun, G. Squeeze-and-excitation networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7132–7141, 2018.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Li, W., Liu, H., Tang, H., Wang, P., and Van Gool, L. Mhformer: Multi-hypothesis transformer for 3d human pose estimation. In *CVPR*, 2022a.

Li, Y., Yuan, G., Wen, Y., Hu, E., Evangelidis, G., Tulyakov, S., Wang, Y., and Ren, J. Efficientformer: Vision transformers at mobilenet speed. *arXiv preprint arXiv:2206.01191*, 2022b.

Li, Z., Wang, W., Li, H., Xie, E., Sima, C., Lu, T., Yu, Q., and Dai, J. Bevformer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers. *arXiv preprint arXiv:2203.17270*, 2022c.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017a.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017b.

Liu, F., Wei, H., Zhao, W., Li, G., Peng, J., and Li, Z. Wb-detr: Transformer-based detector without backbone. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2979–2987, 2021a.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Liu, S., Li, F., Zhang, H., Yang, X., Qi, X., Su, H., Zhu, J., and Zhang, L. Dab-detr: Dynamic anchor boxes are better queries for detr. *arXiv preprint arXiv:2201.12329*, 2022a.

Liu, Y., Wang, T., Zhang, X., and Sun, J. Petr: Position embedding transformation for multi-view 3d object detection. *arXiv preprint arXiv:2203.05625*, 2022b.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 10012–10022, 2021b.

Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 116–131, 2018.

Mehta, S. and Rastegari, M. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*, 2021.

Meng, D., Chen, X., Fan, Z., Zeng, G., Li, H., Yuan, Y., Sun, L., and Wang, J. Conditional detr for fast training convergence. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3651–3660, 2021.

Pan, Z., Cai, J., and Zhuang, B. Fast vision transformers with hilo attention. *arXiv preprint arXiv:2205.13213*, 2022.

Parikh, A., Täckström, O., Das, D., and Uszkoreit, J. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2249–2255, 2016.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Redmon, J. and Farhadi, A. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

Roh, B., Shin, J., Shin, W., and Kim, S. Sparse detr: Efficient end-to-end object detection with learnable sparsity. *arXiv preprint arXiv:2111.14330*, 2021.

Song, H., Sun, D., Chun, S., Jampani, V., Han, D., Heo, B., Kim, W., and Yang, M.-H. Vidt: An efficient and effective fully transformer-based object detector. *arXiv preprint arXiv:2110.03921*, 2021.

Sun, Z., Cao, S., Yang, Y., and Kitani, K. M. Rethinking transformer-based set prediction for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 3611–3620, 2021.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

Tang, H., Zhang, Z., Shi, H., Li, B., Shao, L., Sebe, N., Timofte, R., and Van Gool, L. Graph transformer gans for graph-constrained house generation. In *CVPR*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Wang, T., Yuan, L., Chen, Y., Feng, J., and Yan, S. Pnp-detr: Towards efficient visual analysis with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4661–4670, 2021a.

Wang, W., Xie, E., Li, X., Fan, D.-P., Song, K., Liang, D., Lu, T., Luo, P., and Shao, L. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021b.

Wang, Y., Guizilini, V. C., Zhang, T., Wang, Y., Zhao, H., and Solomon, J. Detr3d: 3d object detection from multi-view images via 3d-to-2d queries. In *Conference on Robot Learning*, pp. 180–191. PMLR, 2022a.

Wang, Y., Zhang, X., Yang, T., and Sun, J. Anchor detr: Query design for transformer-based detector. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 2567–2575, 2022b.

Xie, Z., Zhang, Z., Zhu, X., Huang, G., and Lin, S. Spatially adaptive inference with stochastic feature sampling and interpolation. In *European conference on computer vision*, pp. 531–548. Springer, 2020.

Yang, G., Tang, H., Ding, M., Sebe, N., and Ricci, E. Transformer-based attention networks for continuous pixel-wise prediction. In *ICCV*, 2021a.

Yang, J., Li, C., Zhang, P., Dai, X., Xiao, B., Yuan, L., and Gao, J. Focal self-attention for local-global interactions in vision transformers, 2021b.

Zhang, G., Luo, Z., Yu, Y., Cui, K., and Lu, S. Accelerating detr convergence via semantic-aligned matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 949–958, 2022.

Zheng, S., Lu, J., Zhao, H., Zhu, X., Luo, Z., Wang, Y., Fu, Y., Feng, J., Xiang, T., Torr, P. H., et al. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6881–6890, 2021.

Zhou, L., Zhou, Y., Corso, J. J., Socher, R., and Xiong, C. End-to-end dense video captioning with masked transformer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8739–8748, 2018.

Zhu, X., Su, W., Lu, L., Li, B., Wang, X., and Dai, J. Deformable detr: Deformable transformers for end-to-end object detection. In *International Conference on Learning Representations (ICLR)*, 2020.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

A. Appendix

A.1. Ablation

Replacement of RepCNN in the HOB backbone. To analyze the trade-off between the detection precision and the efficiency of the RepCNN utilization, we replace the WMSA$_{bn}$/SWMSA$_{bn}$ with the RepCNN in the $1st$ and $2nd$ stage. We experimentally find that detection performance is improved by 0.3%∼0.5% AP after replacing WMSA with RepCNN in S_1 of our backbone. The precision is only improved by 0.1% AP after replacing with RepCNN layers in the whole S_1, S_2. So a dedicated design is required here to extract the texture-level information effectively. The detailed structure of RepCNN is illustrated in Figure 9.

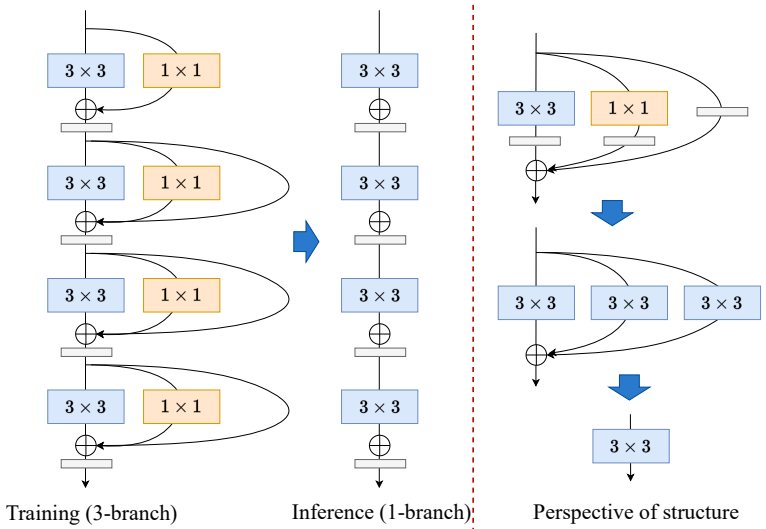


Figure 9: **RepCNN structure**. We also show the training status and the inference status of this structure, respectively.

Semantic-augmented module (SAM). Similar to FPN (Lin et al., 2017a), SAM enhances the semantic information from high-level characteristics to low-level ones to produce richer low-level semantic features for object detection tasks. For a fair comparison, we disable the two encoders and directly feed features from the backbone to RetinaNet's head, a multi-level feature head that accepts four-scale features. The results are shown in Table 6. While both SAM and FPN improve the precision, SAM obtains 0.6% higher AP with 9GFLOPs less than FPN. Therefore, the global convolutional modulation suits the transformer better than FPN. Including SAM within the forward pass can obtain an even stronger model.

HOD stage breakdown. We study how the global convolutional modulation (GCM) and the semantic-augmented module (SAM) contribute to HOD's performance. We only modify the backbone network without the SCM and TCM modules. Table 5 shows that switching from SW-MSA to our global convolutional modulation can improve 1.1% precision (39.9% vs. 38.8%) without significantly impacting GFLOPs (44 vs. 45). Adding the SAM module further increases the precision from 39.9% to 41.2%. These two observations suggest that SAM can enhance performance, and attention is not the primary cause.

Scale-combined module (SCM). SCM aggregates multi-scale features into one feature map to reduce the computational costs of the inference stage. We compare SCM with a similar design in YOLOF (Chen et al., 2021), which exploits a dilated encoder to convert features from multiple scales. Table 7 shows that SCM improves 1.6% AP from the dilated encoder of YOLOF.

Task-couple module (TCM). Benefiting from the global convolutional modulation's capability of modeling semantic relations, TCM handles task conflicts in a coupled head and generates task-aligned predictions in a single pass. As shown in the first row of Figure 11, after replacing TCM with YOLOF's head in the baseline model, the best anchors for classification (red) and localization (orange) are distant. This is because YOLOF uses a task-unaligned decoupled head that leads to inconsistent predictions of classification and localization. Comparatively, our TCM provides aligned predictions with high classification and IOU scores.

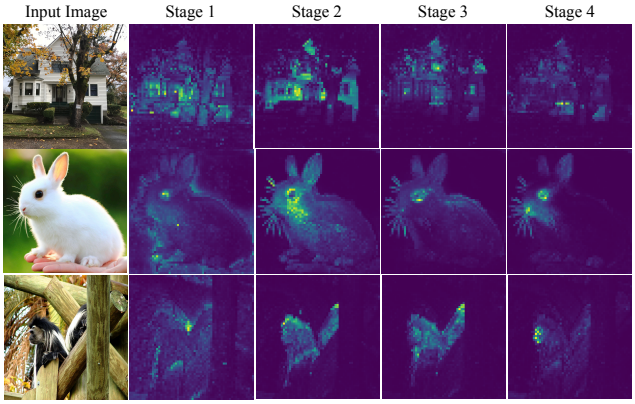


Figure 10: Visualization of the feature map obtained by each HOD stage.

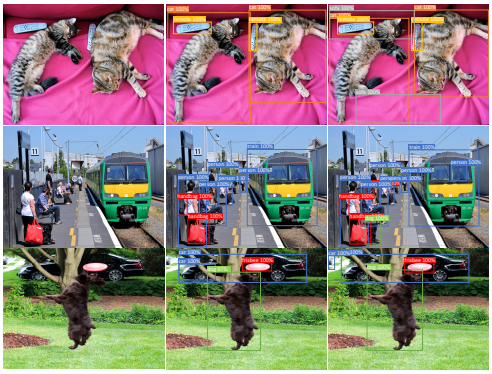


Figure 11: Object detection comparison.

Table 5: Hardware-oriented Detector (HOD) breakdown.

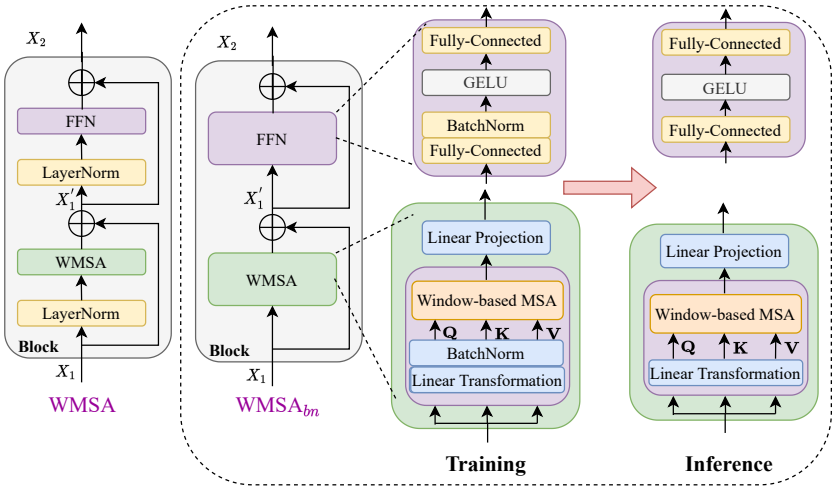
GCM	SAM	AP (%)	GFLOPs
-	-	38.8	**44**
✓	-	39.9	45
✓	✓	**41.2**	67

Table 6: Semantic-augmented module (SAM) breakdown.

SAA	FPN	AP (%)	GFLOPs
-	-	36.9	**323**
✓	-	**38.4**	336
-	✓	37.8	345

Table 7: Scale-combined module (SCM) breakdown.

Method	AP (%)	GFLOPs
YOLOF	39.6	**62**
SpeedDETR	**41.2**	67



$$f^* = W_{linear} \cdot (W_{bn} \cdot f + b_{bn}) + b_{linear}$$
$$W = W_{linear} \cdot W_{bn}$$
$$b = W_{linear} \cdot b_{bn} + b_{linear}$$

Model	Norm	Top-1 Acc (%)	FPS
Swin-T	LN	**81.2**	717.4
	WMSA_{bn}	81.0	**866**
Swin-S	LN	**83.0**	415.2
	WMSA_{bn}	82.8	**467.2**

Model-level comparison of WMAS and WMSA_{bn} with ImageNet-1K on RTX3090 Ti.

Figure 12: **WMSA_{bn}/SWMSA_{bn} structure**. 13%∼21% speedup can be achieved with <0.3 accuracy degradation on ImageNet-1K.

A.2. BN-based Swin-Transformer

We modify the basic structure of Swin-Transformer, WMSA/SWMSA, into WMSA_{bn}/SWMSA_{bn} as shown in Figure 12. Compared to the original design with LN-Linear, a 13%∼21% speedup is harvested with negligible accuracy degradation (<0.3%) on small models. In this paper, we use WMSA_{bn}/SWMSA_{bn} as our design candidates.

A.3. Convolutional modulation

Following (Hou et al., 2022), we replace the self-attention inside the transformer layer with a convolutional modulation layer. As shown in Figure 6(b), we modulate the value V with convolutional features. Let $X \in \mathbb{R}^{H \times W \times C}$ be input tokens, and we use depthwise convolution with kernel size $k \times k$ and the Hadamard product to calculate the output:

$$
\begin{aligned}
Z &= A \odot V, \\
A &= DConv_{k\times k}(W_1 X), \\
V &= W_2 X,
\end{aligned} \tag{4}
$$

where $\odot$ is the Hadamard product, W_1 and W_2 are the weight matrics of two linear layers, and $DConv_{k\times k}$ denotes the depthwise convolution. The linear layers can be used to achieve the information interaction between channels. The weighted sum of all the pixels in the square area is the output for each spatial location. Our methods use convolution instead of self-attention to create associations, which are more memory-efficient (linear memory complexity), especially when processing high-resolution images. Due to the modulation operation, our method differs from traditional residual blocks and can adapt to the input content.

A.4. Speed-aware model slimming strategy

We have introduced our training pipeline based on the search algorithm in this work (Li et al., 2022b). We have changed the search space, the method to generate the runtime latency adapted to our task and application scenarios, and the evolution step. Please refer to (Li et al., 2022b) for more training details.

We provide the details of the proposed fast speed-aware model slimming strategy in Algorithm 1. The proposed speed-aware model slimming strategy is speed-oriented for the target device, which does not need retraining for each sub-network. The importance score for each device-design choice is estimated based on the trainable architecture parameter r, which can be obtained by Gumble Softmax.

Algorithm 1 Speed-aware Model Slimming.

Given: Latency prediction model E; Target latency (FPS) T;
Two-phase design space $DP = \{RepCNN, WMSA_{bn}, SWMSA_{bn}\}_{dim=16\times}, \{WMSA_{bn}, SWMSA_{bn}\}_{dim=16\times}$;
Requirement: Final latency budget: $\sum l \approx T$
Super-net Pretraining:
foreach *epoch* **do**
  foreach *iteration* **do**
    foreach $DP_{i,j}$ **do**
      $\kappa_{i+1} = \sum_n \frac{e^{(r_i^n+\varepsilon_i^n)/\tau}}{\sum_n e^{(r_i^n+\varepsilon_i^n)/\tau}} \cdot DP_{i,j}(\kappa_i)$
    end
    $\pounds = criterion\_loss\_function(output, label)$;
    Backpropagate (£);
    Update parameters;
  end
end
Δ Obtain the Supernet.
Speed-Driven Model Slimming:
E $\in$ {Layer Reduction (LR), Width Reduction (WR), WMSA Reduction (WMR), SWMSA Reduction (SR)};
Calculate the importance of $DP_{i,j}$ through $\mathbb{M}_{i,j} = \frac{r_i^{RepCNN}+r_i^{WMSA}}{r_i^{SWMSA}}$ or $\frac{r_i^{WMSA}}{r_i^{SWMSA}}$;

while $\sum L > \top$ **do**
  $LR \leftarrow argmin_{\mathbb{M}_{i,j}}(DP_{i,j})$,
  $SR \leftarrow argmin_{\sum_j \mathbb{M}_{i,j}}, (DP_{i,j})$,
  $WMR \leftarrow argmin_{\sum_j \mathbb{M}_{i,j}}(DP_{i,j})$,
  $WR \leftarrow argmin_{\sum_j \mathbb{M}_{i,j}}(DP_{i,j})$,
  Adjust the channel dimension C in $argmin_{\sum_j \mathbb{M}_{i,j}}(DP_{i,j})$;
  Conduct Evolution = $argmin_{AP_{drop} * l_{i,j}}(E)$
end
Δ Obtain the Subnet with the target Latency/FPS.

Train the searched architecture from scratch:
SDG-Training method;
Δ Obtain the final model.

Table 8: Comparison of SpeedDETRand basic YOLO version on MS COCO dataset. FPS is measured on a single Tesla V100 GPU.

Method	AP (%)	FPS
YOLOv3 + Darknet-53 (Redmon & Farhadi, 2018)	43.9	29.4
YOLACT-700+R-101 (Bolya et al., 2019)	39.6	23.4
SpeedDETR$_{nano}$	**43.9**	**29.5**

A.5. Comparison with YOLO series

We also compare SpeedDETR with the basic version of the YOLO series as Table 8, YOLOv3, and YOLACT-700, and validate the exceptional trade-off between the precision and speed. Note that SpeedDETR is freely ported to multimodal domains, e.g., autonomous driving, which is demanding with the YOLO series.

A.6. Latency prediction model

The inputs of the latency prediction model include: 1) the structure configuration of a candidate block, 2) the spatial granularity G, 3) the channel dimension C, and 4) the hardware properties are shown in Table 1. The latency of a candidate block is predicted according to the following three steps.

Input/output shape definition. Calculating the input and output shapes is the first step in determining an operation's latency. Taking the MSA operation as an example, the input of this operation is the activation with the shape of $C_{in}\times H\times W$, where C_{in} is the number of input channels, and H and W are the resolutions of the feature map. The shape of the output tensor is $\frac{H}{G}\times\frac{W}{G}\times C_{out}$, where $\frac{H}{G}\times\frac{W}{G}$ is the number of output patches, C_{out} is the number of output channels and G is the spatial granularity.

Operation-to-hardware mapping. We map the operations to hardware. We have modeled a hardware device as multiple processing engines (PEs). We first consecutively split the output feature map into multiple tiles. Specifically, the shape of each tile is TP $\times$ TC $\times$ TS1 $\times$ TS2 $(\frac{H}{G}\times\frac{W}{G})\times T_C\times T_G\times T_G$. These split tiles are assigned to multiple PEs. The computation of each tile is executed in a PE.

Latency estimation. We evaluate each tile's latency, including the data movement latency and the computation latency: $l{=}l_{data}{+}l_{compute}$.

1)*Data movement latency* l_{data}. We model the memory system of hardware as a three-level architecture (Hennessy & Patterson, 2011): off-chip memory, on-chip global memory, and local memory in PE. The input data and weight data first move from the off-chip memory to the on-chip global memory. To simplify the latency prediction model, we assume that the hardware can fully utilize the off-chip memory bandwidth.

The data used to calculate the output tiles is moved from the on-chip global memory to each PE's local memory. The latency of data movement to local memory is estimated by its bandwidth and efficiency. To make the prediction model simpler, we assume that each PE only moves the appropriate input feature maps and weights once to compute an output tile. The time from off-chip memory to on-chip global memory and the time from on-chip global memory to local memory are added together to compute the input data movement latency l_{in}: $l_{in}{=}l_{off2on}{+}l_{global2local}$. The output data are transferred from local memory to on-chip global memory and subsequently to off-chip memory, in contrast to the input data: $l_{out}{=}l_{local2global}{+}l_{on2off}$. By combining the input and output data movement latency, we can determine the overall data movement latency: $l_{data}{=}l_{in}{+}l_{out}$.

The granularity G impacts the latency of data movement because when it is small, more input data will be transferred to numerous PEs to compute various output patches, dramatically increasing the number of on-chip memory movements. This explains why a larger G will significantly increase the practical efficiency, according to the experiment results in the paper.

2)*Computation latency* $l_{compute}$. The maximal FP32 computation throughput of the PE and the FLOPs required to compute an output tile are used to estimate the computation latency of each tile. The number of tiles and PEs can be used to determine the overall computation latency.

Stage	Resolution	Type	Config	SpeedDETR		
				Nano	Tiny	Base
Image Embed.	$\frac{H}{2} \times \frac{W}{2}$	Image Embed.	Patch Size	k=3x3,s=2		
			Embed. Dim.	48	64	64
	$\frac{H}{4} \times \frac{W}{4}$	Image Embed.	Patch Size	k=3x3,s=2		
			Embed. Dim.	96	128	128
	$\frac{H}{8} \times \frac{W}{8}$	Image Embed.	Patch Size	k=3x3,s=2		
			Embed. Dim.	96	128	128
1	$\frac{H}{8} \times \frac{W}{8}$	Local Attention	RepCNN=$\begin{bmatrix} Embed. & Kernel \\ Stride & Exp \end{bmatrix}$	[96, 3, 1, 4]×1	[128, 3, 1, 4] ×1	[128, 3, 1, 4] ×1
			SWMSA$_{bn}$=$\begin{bmatrix} Embed. & D_{QK} \\ Heads & Exp \end{bmatrix}$	[96, 96, 3, 4] ×1	[128, 128, 4, 4] ×1	[128, 128, 4, 4] ×1
		Global Attention	Convolutional Modulation=$\begin{bmatrix} Embed. & D_{QK} \\ Heads & Exp \end{bmatrix}$	[96, 96, 3, 4] ×1	[128,128, 4, 4] ×1	[128, 128, 4, 4]×1
2	$\frac{H}{16} \times \frac{W}{16}$	Patch Embed.	Patch Size	k=3X3, s=2		
			Embed. Dim.	96	128	128
		Local Attention	RepCNN=$\begin{bmatrix} Embed. & Kernel \\ Stride & Exp \end{bmatrix}$	[96, 3, 1, 4] ×1	[128, 3, 1, 4] ×1	[128, 3,1,4] ×1
			SWMSA$_{bn}$=$\begin{bmatrix} Embed. & D_{QK} \\ Heads & Exp \end{bmatrix}$	[96, 96, 3, 4]×1	[128, 128, 4, 4]×1	[128, 128, 4, 4]×1
		Global Attention	Convolutional Modulation=$\begin{bmatrix} Embed. & D_{QK} \\ Heads & Exp \end{bmatrix}$	[96, 96, 3, 4]×1	[128, 128, 4, 4]×1	[128, 128, 4, 4]×1
		SAM	Convolutional Modulation=$\begin{bmatrix} Embed. & D_{QK} \\ Heads & Exp \end{bmatrix}$	[96, 96, 3, 4] ×1	[128, 128, 4, 4]×1	[128, 128, 4, 4]×1
3	$\frac{H}{32} \times \frac{W}{32}$	Patch Embed.	Patch Size	k=3X3, s=2		
			Embed. Dim.	192	256	224
		Local Attention	{RepCNN=$\begin{bmatrix} Embed. & Kernel \\ Stride & Exp \end{bmatrix}$, SWMSA$_{bn}$=$\begin{bmatrix} Embed. & D_{QK} \\ Heads & Exp \end{bmatrix}$}	{[192, 3,1,4] , [192, 192, 6, 4]}×2	{[256, 3,1,4] , [256, 256, 8, 4]}×2	{[224, 3,1,4] , [224, 224, 7, 4]}×5
			{WMSA$_{bn}$=$\begin{bmatrix} Embed. & D_{QK} \\ Heads & Exp \end{bmatrix}$, SWMSA$_{bn}$=$\begin{bmatrix} Embed. & D_{QK} \\ Heads & Exp \end{bmatrix}$}	{[192, 192, 6, 4], [192, 192, 6, 4]}×1	{[256, 256, 8, 4], [256, 256, 8, 4]}×1	{[224, 224, 7, 4], [224, 224, 7, 4]}×4
		Global Attention	Convolutional Modulation=$\begin{bmatrix} Embed. & D_{QK} \\ Heads & Exp \end{bmatrix}$	[192, 192, 6, 4]×1	[256, 256, 8, 4]×1	[224, 224, 7, 4]×1
		SAM	Convolutional Modulation=$\begin{bmatrix} Embed. & D_{QK} \\ Heads & Exp \end{bmatrix}$	[96, 96, 3, 4]×1	[128, 128, 4, 4]×1	[128, 128, 4, 4]×1
4	$\frac{H}{64} \times \frac{W}{64}$	Patch Embed.	Patch Size	k=3X3, s=2		
			Embed. Dim.	288	384	384
		Local Attention	{WMSA$_{bn}$=$\begin{bmatrix} Embed. & D_{QK} \\ Heads & Exp \end{bmatrix}$, SWMSA$_{bn}$=$\begin{bmatrix} Embed. & D_{QK} \\ Heads & Exp \end{bmatrix}$}	{[288, 288, 9, 4], [288, 288, 9, 4]}×1	{[384, 384, 12, 4], [384, 384, 12, 4]}×1	{[384, 384, 12, 4], [384, 384, 12, 4]}×1
		Global Attention	Convolutional Modulation=$\begin{bmatrix} Embed. & D_{QK} \\ Heads & Exp \end{bmatrix}$	[288, 288, 9, 4]×1	[384, 384, 12, 4]×1	[384, 384, 12, 4]×1
		SAM	Convolutional Modulation=$\begin{bmatrix} Embed. & D_{QK} \\ Heads & Exp \end{bmatrix}$	[192, 192, 6, 4]×1	[256, 256, 8, 4]×1	[224, 224, 7, 4]×1

Table 9: Detailed architectures of the HOD backbone of SpeedDETR. D_{QK} is the dimension of Queries and Keys. Exp refers to the expansion ratio of the MLP block.

A.7. Model configuration of SpeedDETR

The detailed network architectures for the backbone of SpeedDETR-nano, SpeedDETR-tiny, and SpeedDETR-base are provided in Table 9. We report the resolution and number of blocks for each stage. In addition, the width of SpeedDETR is specified as the embedding dimension (Embed., Dim.). As for the MHSA block, the dimension of Query and Key is provided.