

An Extendable, Efficient and Effective Transformer-based Object Detector

Hwanjun Song, Deqing Sun, Sanghyuk Chun, Varun Jampani, Dongyoong Han,
Byeongho Heo, Wonjae Kim, Ming-Hsuan Yang

Abstract—Transformers have been widely used in numerous vision problems especially for visual recognition and detection. Detection transformers are the first fully end-to-end learning systems for object detection, while vision transformers are the first fully transformer-based architecture for image classification. In this paper, we integrate Vision and Detection Transformers (ViDT) to construct an effective and efficient object detector. ViDT introduces a reconfigured attention module to extend the recent Swin Transformer to be a standalone object detector, followed by a computationally efficient transformer decoder that exploits multi-scale features and auxiliary techniques essential to boost the detection performance without much increase in computational load. In addition, we extend it to ViDT+ to support joint-task learning for object detection and instance segmentation. Specifically, we attach an efficient multi-scale feature fusion layer and utilize two more auxiliary training losses, IoU-aware loss and token labeling loss. Extensive evaluation results on the Microsoft COCO benchmark dataset demonstrate that ViDT obtains the best AP and latency trade-off among existing fully transformer-based object detectors, and its extended ViDT+ achieves 53.2AP owing to its high scalability for large models. The source code and trained models are available at <https://github.com/naver-ai/vidt>.

Index Terms—Vision Transformers, Detection Transformers, Object Detection, Instance Segmentation

1 INTRODUCTION

OBJECT detection aims to predict both the bounding box and object class for each object of interest in an image. Recent deep object detectors rely heavily on meticulously designed components, such as anchor generation and non-maximum suppression [1], [2]. As a result, the performance of these object detectors depends on specific postprocessing steps, which involve complex pipelines and make fully end-to-end training difficult.

Motivated by the recent success of Transformers [3] in NLP, numerous models have been developed for various vision tasks, especially in recognition and detection. Carion et al. [4] propose the Detection Transformers (DETR) to replace the meticulously designed components with a transformer encoder and decoder architecture, which serves as a neck component to bridge a CNN body for feature extraction and a detector head for prediction. As such, DETR enables end-to-end training of deep object detectors. On the other hand, Dosovitskiy et al. [5] show that a fully-transformer backbone without any convolutional layers, Vision Transformer (ViT), achieves state-of-the-art results on image classification benchmarks. DETR and ViT have been shown to learn effective representation models without relying strongly on human inductive biases, *e.g.*, meticulously designed components in object detection (DETR), convolutional layers and pooling mechanisms for locality-aware designs (ViT). However, no attempt has been made to synergize DETR and ViT for a better object detection architecture. In this work, we integrate both approaches to construct a fully transformer-based, end-to-end object detector that achieves state-of-the-art performance without increasing computational load.

Straightforward integration of DETR and ViT can be

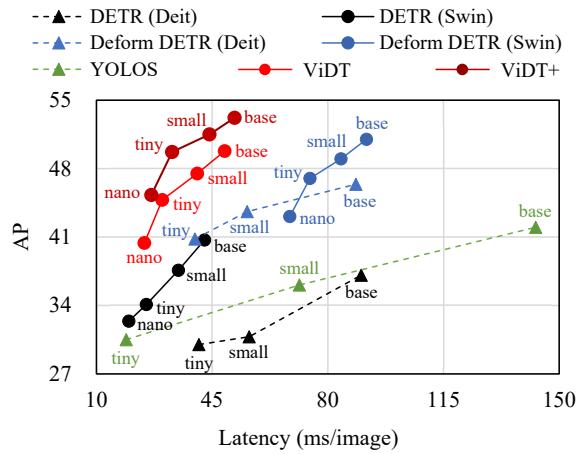


Fig. 1. Performance of recent object detectors in terms of average precision (AP) and latency. Detailed The text in the plot indicates the backbone model size. The latency was measured with batch size 1 of 800×1333 resolution on NVIDIA A100 GPU. AP and latency (milliseconds) are summarized in Table 3.

achieved by replacing the ResNet backbone (body) of DETR with ViT as shown in Figure 2(a). This naive integration, DETR (ViT)¹, has two limitations. First, as the original ViT suffers from the quadratic increase in complexity w.r.t. image size, this approach does not scale up well. Furthermore, the attention operation at the transformer encoder and decoder (*i.e.*, the “neck” component) adds significant computational overhead to the detector. Thus, the naive integration of DETR and ViT would cause high latency, as shown in the blue lines of Figure 1.

1. We refer to each model based on the combinations of its body and neck. For example, DETR (DeiT) indicates that DeiT (vision transformers) is integrated with DETR (detection transformers).

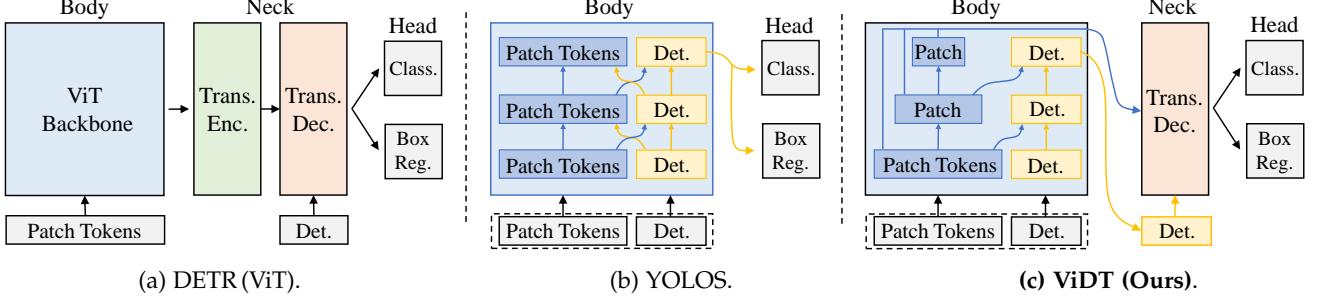


Fig. 2. Pipelines of fully transformer-based object detectors. DETR (ViT) denotes Detection Transformer using ViT as its body. ViDT exploits merits of DETR(ViT) and YOLOS and achieves the best AP and latency trade-off among fully transformer-based object detectors.

Recently, Fang et al. [6] propose the YOLOS model by appending the detection tokens [DET] to the patch tokens [PATCH] (see Figure 2(b)), where [DET] tokens are learnable embeddings to specify different objects to detect. YOLOS is a neck-free architecture and removes the additional computational costs from the neck encoder. However, YOLOS shows limited performance because it cannot exploit additional optimization techniques on the neck architecture, *e.g.*, multi-scale features and auxiliary loss. In addition, YOLOS can only accommodate the original transformer due to its architectural limitation, resulting in a quadratic complexity w.r.t. the input size.

In this paper, we propose a novel integration of Vision and Detection Transformers (ViDT) (see Figure 2(c)). Our contributions are three-fold. First, ViDT introduces the Re-configured Attention Module (RAM), to facilitate any ViT variant to handle the appended [DET] and [PATCH] tokens for object detection. Thus, we can integrate the Swin Transformer [7] backbone with RAM to be an object detector and obtain high scalability using its local attention mechanism with linear complexity. Second, ViDT adopts a lightweight encoder-free neck architecture to reduce the computational overhead while still enabling the additional optimization techniques on the neck module. Note that the neck encoder is unnecessary because RAM directly extracts fine-grained representation for object detection, *i.e.*, [DET] tokens. As a result, ViDT achieves better performance than its neck-free counterparts. Finally, we extend the vanilla ViDT to an end-to-end architecture named ViDT+, thereby enabling multi-task learning of object detection and instance segmentation. For effective multi-task learning, ViDT+ equips efficient feature pyramid layers on top of its body for multi-scale feature fusion, and leverages two additional training losses *i.e.*, IoU-aware loss [8] and token labeling loss [9]. ViDT+ only adds 1M learnable parameters and barely reduces its inference speed than ViDT, but achieves a significant accuracy gain.

ViDT has three architectural advantages over existing approaches. First, similar to YOLOS, ViDT takes [DET] tokens as the additional input, maintaining a fixed scale for object detection, but constructs hierarchical representations starting with small-sized image patches for [PATCH] tokens. Second, ViDT can use the hierarchical (multi-scale) features and additional techniques without a significant computation overhead. Thus, as a fully transformer-based object detector, ViDT facilitates better integration of vision and detection transformers. Third, ViDT accommodates numerous tasks and transformer models. It can be extended to be an

end-to-end architecture for multi-task learning, and easily combined with ViT variants such as CoaT [10] and PVT-v2 [11] other than the Swin Transformer. Extensive experiments on Microsoft COCO [12] show that ViDT is highly scalable even for large ViT models, such as Swin-base with 0.1 billion parameters, and achieves the best AP and latency trade-off among existing fully transformer-based detectors. In particular, ViDT+ with Swin-base achieves 53.2AP for object detection, which is 2.6AP higher than that of the vanilla ViDT.

This work is an extensive version of our ICLR 2022 [13]. Compared to the ICLR 2022 work, this paper includes the following additional contributions: (1) a new joint-learning framework named ViDT+, an extension of the vanilla ViDT, by incorporating three additional components: an efficient pyramid feature fusion module, a unified query representation module, and an IoU-aware and token labeling losses, (2) a detailed computational complexity analysis of the proposed VIDT compared to the YOLOS detector, (3) a performance comparison for object detection and instance segmentation compared with other CNN-based state-of-the-art methods, (4) an ablation study for the three new components of ViDT+ and a complete analysis of all the proposed components, (5) the reconfigured attention module combined with ViT variants such as CoaT and PVT-v2 other than the Swin Transformer.

2 PRELIMINARIES

Object detection is a task of predicting a set of bounding boxes and classification labels for each object of interest. There have been significant efforts to develop efficient and effective detection backbones and pipelines [14], [15]. Typically, as a pre-trained backbone is fine-tuned in single- and two-stage fashion; single-stage detectors produce predictions directly w.r.t. anchors or a grid of possible object centers, e.g., SSD [16] and YOLO [17], while two-stage detectors predict bounding boxes w.r.t. region proposals, e.g., Faster R-CNN [18]. The success of modern object detectors has achieved very high detection accuracy, but they heavily depend on some meticulously designed components, such as anchor generation and non-maximum suppression [1], [2]. In addition, finding the optimal trade-off between detection accuracy and inference speed is not a trivial task. In this paper, we study the integration of vision and detection transformers and show its potential to be a new generic detection pipeline, achieving good performance trade-offs without the meticulously designed components.

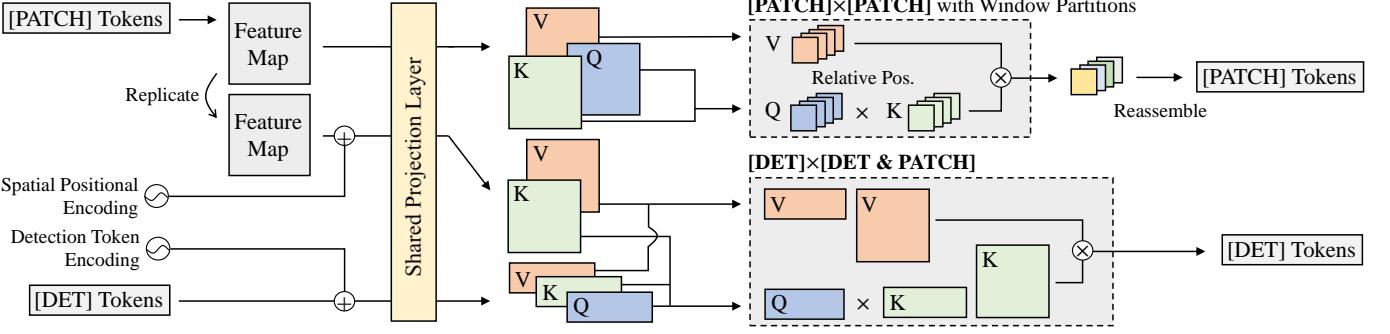


Fig. 3. Reconfigured Attention Module (Q: query, K: key, V: value). The skip connection and feedforward networks following the attention operation is omitted just for ease of exposition.

Vision transformers process an image as a sequence of small image patches, thereby facilitating consideration of interaction among patches at all positions (*i.e.*, global attention). However, the original ViT [5] cannot be easily scaled to a wide range of vision tasks due to its high computational complexity, which increases quadratically with respect to image size. The Swin Transformer [7] alleviates the complexity issue by introducing the notion of shifted windows that support local attention and patch reduction operations, thereby improving compatibility for dense prediction tasks such as object detection and semantic segmentation. A few approaches use vision transformers as detector backbones but achieve limited success [6], [7], [19]. In this work, we significantly improve the performance of detectors with transformer backbones by the proposed architectural changes and extensions for multi-task learning.

Detection transformers eliminate the meticulously designed components (*e.g.*, anchor generation and non-maximum suppression) by combining convolutional network backbones and Transformer encoder-decoders. While the original DETR [4] achieves high detection performance, it suffers from slow convergence compared to previous detectors. For example, DETR requires 500 epochs while the Faster R-CNN [18] needs only 37 epochs [20] for training. To mitigate the issue, Zhu et al. [21] propose Deformable DETR which introduces deformable attention for utilizing multi-scale features as well as expediting the slow training process of DETR. In this paper, we use the Deformable DETR as our base detection transformer framework and integrate it with the three recent vision transformers.

DETR (ViT) is a straightforward integration of DETR and ViT, which uses ViT as a feature extractor, followed by the transformer encoder-decoder in DETR. As illustrated in Figure 2(a), it is a *body-neck-head* structure; the representation of input [PATCH] tokens are extracted by the ViT backbone and then directly fed to the transformer-based encoding and decoding pipeline. To predict multiple objects, a fixed number of learnable [DET] tokens are provided as additional input to the decoder. Subsequently, the output embeddings by the decoder are considered as final predictions through the detection heads for classification and box regression. Since DETR (ViT) does not modify the backbone at all, it can be flexibly changed to any latest ViT model, *e.g.*, Swin Transformer. Additionally, its neck decoder facilitates the aggregation of multi-scale features and the use of additional

techniques, such as auxiliary decoding loss and iterative box refinement, which help detect objects of different sizes and speed up the training process [21]. However, the attention operation at the neck encoder adds significant computational overhead to the detector. In contrast, ViDT resolves this issue by directly extracting fine-grained [DET] features from the Swin Transformer with RAM without maintaining the transformer encoder in the neck architecture.

YOLOS [6] is a ViT architecture for object detection with minimal modifications. As illustrated in Figure 2(b), YOLOS consists of a *neck-free* structure by appending randomly initialized learnable [DET] tokens to the sequence of input [PATCH] tokens. Since all the embeddings for [PATCH] and [DET] tokens interact via global attention, the final [DET] tokens are generated by the fine-tuned ViT backbone and then directly generate predictions through the detection heads without requiring any neck layer. While the naive DETR (ViT) suffers from the computational overhead from the neck layer, YOLOS enjoys efficient computations by treating the [DET] tokens as additional input for ViT. YOLOS shows that 2D object detection can be accomplished in a pure sequence-to-sequence manner, but this solution entails *two* inherent limitations:

- (1) YOLOS inherits the drawback of the original ViT; the high computational complexity attributed to the global attention operation. As illustrated in Figure 1, YOLOS shows poor latency compared with other fully transformer-based detectors, especially when its model size becomes larger, *i.e.*, small \rightarrow base. Thus, YOLOS is *not scalable* for the large model.
- (2) YOLOS does not benefit from using any additional techniques essential for better detection performance, *e.g.*, multi-scale features, due to the absence of the neck layer. Although YOLOS used the same DeiT backbone with Deformable DETR (DeiT), its AP was lower than the straightforward integration.

In contrast, the encoder-free neck architecture of ViDT enjoys additional optimization techniques from Zhu et al. [21], resulting in faster convergence and better performance. Further, our RAM enables us to combine the Swin Transformer² and sequence-to-sequence paradigm for detection.

2. It is not limited to the Swin Transformer. Our reconfiguration scheme can be easily applied to other variants with simple modifications. See Appendix B.1 for the combination with CoaT and PVT-v2.

3 ViDT: VISION AND DETECTION TRANSFORMERS

ViDT first reconfigures the attention model of the Swin Transformer to support standalone object detection while fully reusing the parameters of the Swin Transformer. Next, it incorporates an encoder-free neck layer to exploit multi-scale features and two essential techniques: auxiliary decoding loss and iterative box refinement.

3.1 Reconfigured Attention Module (RAM)

Applying patch reduction and local attention scheme of the Swin Transformer to the sequence-to-sequence paradigm is challenging because (1) the number of [DET] tokens must be maintained at a fixed-scale and (2) the lack of locality between [DET] tokens. To address these issues, we introduce a reconfigured attention module (RAM) that decomposes a single global attention associated with [PATCH] and [DET] tokens into the *three* different attention, namely [PATCH] \times [PATCH], [DET] \times [DET], and [DET] \times [PATCH] attention. Based on the decomposition, the efficient schemes of the Swin Transformer are applied only to [PATCH] \times [PATCH] attention, which is the heaviest part of computational complexity, without breaking the two constraints on [DET] tokens. As illustrated in Figure 3, these modifications fully reuse all the parameters of the Swin Transformer by sharing projection layers for [DET] and [PATCH] tokens, and perform the three different attention operations:

As a standalone object detector, RAM must be accompanied by three attention operations:

- [PATCH] \times [PATCH] Attention: The initial [PATCH] tokens are progressively calibrated across the attention layers such that they aggregate the key contents in the global feature map (*i.e.*, a spatial form of [PATCH] tokens) according to the attention weights, which are computed by \langle query, key \rangle pairs. For [PATCH] \times [PATCH] attention, the Swin Transformer performs local attention on each window partition, but its shifted window partitioning in successive blocks bridges the windows of the preceding layer, providing connections among partitions to capture global information. We use a similar policy to generate hierarchical [PATCH] tokens. Thus, the number of [PATCH] tokens is reduced by a factor of 4 at each stage; the resolution of feature maps decreases from $H/4 \times W/4$ to $H/32 \times W/32$ over a total of four stages, where H and W denote the width and height of the input image.
- [DET] \times [DET] Attention: Similar to YOLOS, we append one hundred learnable [DET] tokens as the additional input to the Swin Transformer. As the number of [DET] tokens specify the number of objects to detect, their number must be maintained with a fixed-scale over the transformer layers. In addition, [DET] tokens do not have any locality unlike the [PATCH] tokens. Hence, for [DET] \times [DET] attention, we perform global self-attention while maintaining the number of them; this attention helps each [DET] token to localize a different object by capturing the relationship between them.
- [DET] \times [PATCH] Attention: We consider cross-attention between [DET] and [PATCH] tokens, and generate an object embedding per [DET] token. For each [DET] token, the key contents in [PATCH] tokens are aggregated to represent

the target object. Since the [DET] tokens specify different objects, it produces different object embeddings for diverse objects in the image. Without the cross-attention, it is infeasible to realize the standalone object detector. As shown in Figure 3, ViDT binds [DET] \times [DET] and [DET] \times [PATCH] attention to process them at once to increase efficiency.

We replace all the attention modules in the Swin Transformer with the proposed RAM, which receives [PATCH] and [DET] tokens (as shown in “Body” of Figure 2(c)) and then outputs their calibrated new tokens by performing the three different attention operations in parallel.

Positional Encoding. ViDT adopts different positional encodings for different types of attention. For [PATCH] \times [PATCH] attention, we use the relative position bias [22] originally used in the Swin Transformer. In contrast, the learnable positional encoding is added for [DET] tokens for [DET] \times [DET] attention because there is no particular order between [DET] tokens. However, for [DET] \times [PATCH] attention, it is crucial to inject *spatial bias* to the [PATCH] tokens due to the permutation-equivariant in transformers, ignoring spatial information of the feature map. Thus, ViDT adds the sinusoidal-based spatial positional encoding to the feature map, which is reconstructed from the [PATCH] tokens for [DET] \times [PATCH] attention, as can be seen from the left side of Figure 3. We present a thorough analysis of various spatial positional encodings in Section 5.2.1.

Use of [DET] \times [PATCH] Attention. Applying cross-attention between [DET] and [PATCH] tokens adds additional computational overhead to the Swin Transformer, especially when it is activated at the bottom layer due to the large number of [PATCH] tokens. To minimize such computational overhead, ViDT only activates the cross-attention at the last stage (the top level of the pyramid) of the Swin Transformer, which consists of two transformer layers that receives [PATCH] tokens of size $H/32 \times W/32$. Thus, only self-attention for [DET] and [PATCH] tokens are performed for the remaining stages except the last one. In Section 5.2.1, we show that this design choice helps achieve the highest FPS, while achieving similar detection performance as when cross-attention is enabled at every stage.

Binding [DET] \times [DET] and [DET] \times [PATCH] Attention. We bind the two attention modules by a simple implementation. In this work, [DET] \times [DET] and [DET] \times [PATCH] attention generate new [DET] tokens, which aggregate relevant contents in [DET] and [PATCH] tokens, respectively. Since the two attention modules share exactly the same [DET] query embedding obtained after the projection as shown in Figure 3, they can be processed at once by performing matrix multiplication between $[DET]_Q$ and $[DET]_K \cdot [PATCH]_K$ embeddings, where Q, K are the key and query, and $[\cdot]$ is the concatenation. Then, the obtained attention map is applied to the $[DET]_V \cdot [PATCH]_V$ embeddings, where V is the value and d is the embedding dimension,

$$\begin{aligned} [DET]_{new} = \\ \text{Softmax}\left(\frac{[DET]_Q [[DET]_K, [PATCH]_K]^\top}{\sqrt{d}}\right) [[DET]_V, [PATCH]_V]. \end{aligned} \quad (1)$$

TABLE 1. Summary of computational complexity for different attention operations used in YOLOS and ViDT (RAM), where P and D are the number of [PATCH] and [DET] tokens, respectively ($D \ll P$).

Attention Type	YOLOS	ViDT
[PATCH] \times [PATCH] Attention	$\mathcal{O}(d^2P + dP^2)$	$\mathcal{O}(d^2P + dk^2P)$
[DET] \times [DET] Attention	$\mathcal{O}(d^2D + dD^2)$	$\mathcal{O}(d^2D + dD^2)$
[DET] \times [PATCH] Attention	$\mathcal{O}(dPD)$	$\mathcal{O}(d^2(P + D) + dPD)$
Total Complexity	$\mathcal{O}(d^2(P + D) + d(P + D)^2)$	$\mathcal{O}(d^2(P + D) + dk^2P + dD^2 + dPD)$

Embedding Dimension of [DET] tokens. In this work, [DET] \times [DET] attention is performed across every stage, and the embedding dimension of [DET] tokens increases gradually like [PATCH] tokens. For a [PATCH] token, its embedding dimension is increased by concatenating nearby [PATCH] tokens in a grid. However, this mechanism does not apply to [DET] tokens since we maintain the same number of [DET] tokens for detecting a fixed number of objects in a scene. Hence, we simply repeat a [DET] token multiple times along the embedding dimension to increase its size. This allows [DET] tokens to reuse all the projection and normalization layers in the Swin Transformer without any modification³.

3.2 Encoder-free Neck Structure

To exploit multi-scale feature maps, ViDT incorporates a decoder of multi-layer deformable transformers [21]. In the DETR family (Figure 2(a)), a transformer encoder is required at the neck to transform features extracted from the backbone for image classification into the ones suitable for object detection; the encoder is generally computationally expensive since it involves [PATCH] \times [PATCH] attention. However, ViDT maintains only a transformer decoder as its neck, in that the Swin Transformer with RAM directly extracts fine-grained features suitable for object detection as a standalone object detector. Thus, the neck structure of ViDT is computationally efficient.

The decoder receives two inputs from the Swin Transformer with RAM: (1) [PATCH] tokens generated from each stage (*i.e.*, four multi-scale feature maps, $\{\mathbf{x}^l\}_{l=1}^L$ where $L = 4$) and (2) [DET] tokens generated from the last stage. The overview is illustrated in “Neck” of Figure 2(c). In each deformable transformer layer, [DET] \times [DET] attention is performed first. For each [DET] token, multi-scale deformable attention is applied to produce a new [DET] token, aggregating a small set of key contents sampled from the multi-scale feature maps $\{\mathbf{x}^l\}_{l=1}^L$,

$$\text{MSDeformAttn}([\text{DET}], \{\mathbf{x}^l\}_{l=1}^L) = \sum_{m=1}^M \mathbf{W}_m \left[\sum_{l=1}^L \sum_{k=1}^K A_{mlk} \cdot \mathbf{W}'_m \mathbf{x}^l (\phi_l(\mathbf{p}) + \Delta p_{mlk}) \right], \quad (2)$$

where m indices the attention head and K is the total number of sampled keys for content aggregation. In addition, $\phi_l(\mathbf{p})$ is the reference point of the [DET] token re-scaled for the l -th level feature map, while Δp_{mlk} is the sampling offset for deformable attention; and A_{mlk} is the attention weights of the K sampled contents. \mathbf{W}_m and \mathbf{W}'_m are the projection matrices for multi-head attention.

3. When combined RAM with CoaT and PVT-v2, which use convolutional layers for token pooling, we add linear layers to handle the embedding dimension of [DET] tokens.

Auxiliary Techniques for Additional Improvements. The decoder of ViDT follows the standard structure of multi-layer transformers, generating refined [DET] tokens at each layer. Hence, ViDT leverages the two auxiliary techniques used in (Deformable) DETR for additional improvements:

- **Auxiliary Decoding Loss** [4]: Detection heads consisting of two feedforward networks (FFNs) for box regression and classification are attached to every decoding layer. All the training losses from detection heads at different scales are added. This helps the model output the correct number of objects without non-maximum suppression.
- **Iterative Box Refinement** [21]: Each decoding layer refines bounding boxes based on predictions from the detection head in the previous layer. Thus, the box regression process progressively improves through the decoding layers.

These two techniques are essential for transformer-based object detectors because they significantly enhance detection performance without compromising detection efficiency. We provide an ablation study of their effectiveness for object detection in Section 5.2.2.

3.3 Complexity Analysis: ViDT vs. YOLOS

We analyze the computational complexity of the proposed RAM compared with the attention used in YOLOS, based on building blocks of the original and Swin Transformer models [3], [7].

Let P and D be the number of [PATCH] and [DET] tokens ($D \ll P$ in practice, *e.g.*, $P = 66,650$ and $D = 100$ at the first stage of ViDT with the resolution of 800×1333 images), respectively. The computational complexity of the attention module for YOLOS and ViDT (RAM) is then derived as below, also summarized in Table 1:

- **YOLOS Attention:** [DET] tokens are simply appended to [PATCH] tokens to perform global self-attention on [PATCH, DET] tokens (*i.e.*, $P + D$ tokens). Thus, the computational complexity is $\mathcal{O}(d^2(P + D) + d(P + D)^2)$, which is *quadratic* to the number of [PATCH] tokens. If breaking down the total complexity, we obtain $\mathcal{O}((d^2P + dP^2) + (d^2D + dD^2) + dPD)$, where the first and second terms are for the global self-attention for [PATCH] and [DET] tokens, respectively, and the last term is for the global cross-attention between them.
- **ViDT (RAM) Attention:** RAM performs the three different attention operations (with the complexities): (1) [PATCH] \times [PATCH] local self-attention with window partition, $\mathcal{O}(d^2P + dk^2P)$; (2) [DET] \times [DET] global self-attention, $\mathcal{O}(d^2D + dD^2)$; (3) [DET] \times [PATCH] global cross-attention, $\mathcal{O}(d^2(P + D) + dPD)$. In total, the computational complexity of RAM is $\mathcal{O}(d^2(P + D) + dk^2P + dD^2 + dPD)$, which is *linear* to the number of [PATCH] tokens.

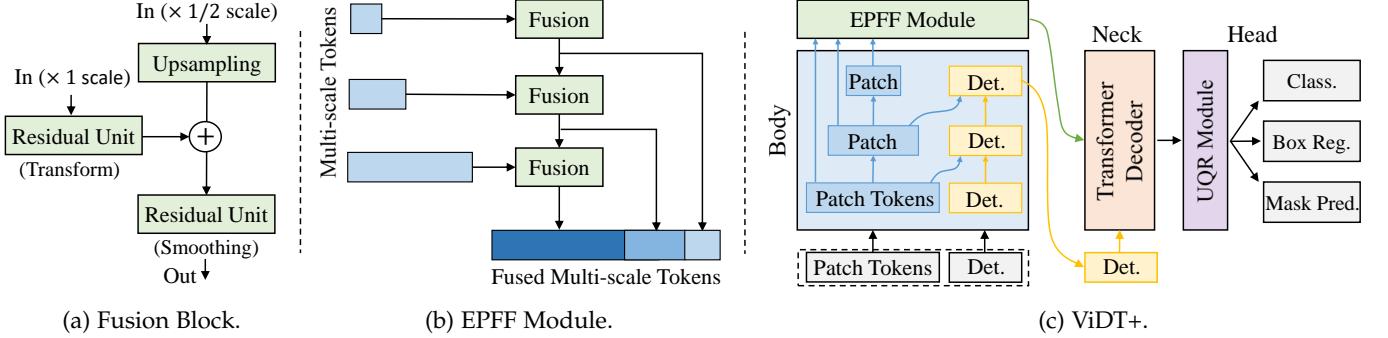


Fig. 4. Architecture overview of ViDT+: (a) two inputs with different scales are fused by a fusion block, (b) multiple fusion blocks are used in the EEEF module to mix multi-scale input tokens, returning their concatenated single output, and (c) the extended ViDT+ equips the EEEF module for multi-scale feature fusion and the UQR module [23] for end-to-end multi-task learning.

Consequently, the computational complexity of ViDT(RAM) is much lower than that of the attention module used in YOLOS since $D \ll P$; RAM achieves the *linear* complexity to the patch tokens, while YOLOS suffers from the quadratic complexity.

4 ViDT+: EXTENSION TO MULTI-TASK LEARNING

The proposed ViDT significantly improves both the computational complexity and the performance of the end-to-end transformer-based detector. We additionally analyze two drawbacks of ViDT; (1) The multi-scale PATCH tokens are *linearly* fused, failing to extract complex complementary information; (2) ViDT does not apply to other tasks, such as instance segmentation. Resolving the two issues are not trivial. For the former, we need to develop an efficient way of fusing the features non-linearly without compromising inference speed. For the latter, existing FPN-style networks for DETR family detectors cannot be trained in an end-to-end manner for multi-task learning [4]. We address the drawbacks by adding three components, namely, Efficient Pyramid Feature Fusion (EPFF), Unified Query Representation (UQR), and IoU-aware and Token Labeling Losses. The method that incorporates all three components is referred to as ViDT+, as illustrated in Figure 4(c).

4.1 Efficient Pyramid Feature Fusion Module

All the multi-scale [PATCH] tokens of ViDT are fed into the encoder-free neck component of ViDT without any processing. Then, the [PATCH] tokens are decoded into object embeddings, *i.e.*, the final [DET] tokens. As in Eq. (2), the multi-scale deformable attention of the decoder fuses the multi-scale [PATCH] tokens *linearly* via weighted aggregation, but only a few sampled [PATCH] tokens (K tokens per scale) are used for computational efficiency. We thus introduce a simple but efficient pyramid feature fusion (EPFF) module, which fuses all the available multi-scale tokens *non-linearly* using multiple CNN fusion blocks before putting them into the decoder, as illustrated in Figure 4(a) and 4(b). The proposed EPFF extracts complementary information from feature maps at different scales more effectively than the previous simple linear aggregation.

Specifically, all the [PATCH] tokens with multiple scales from the body's different stages are assembled to form multi-scale feature maps $\{\mathbf{x}^l\}_{l=1}^L$ with the same size of

channel dimension⁴. Subsequently, they are fused in a top-down manner. Each fusion block in Figure 4(a) receives two input feature maps for pyramid feature fusion: (1) a feature map \mathbf{x}^l for a target l -th scale (higher resolution) and (2) the fused feature map from the predecessor fusion block for the feature map \mathbf{x}^{l+1} (lower resolution),

$$\mathbf{x}_{fuse}^l = \overbrace{\text{ResUnit}}^{\text{smoothing}} \left(\overbrace{\text{Upsample}(\mathbf{x}_{fuse}^{l+1}) + \text{ResUnit}(\mathbf{x}^l)}^{\text{interpolation}} \right), \quad (3)$$

where the Upsample operator resizes the low-resolution feature map to fuse with the high-resolution one via bilinear interpolation, and the two ResUnits are the bottleneck residual block for feature transform and feature smoothing, respectively. As a result, fused multi-scale features are obtained, flattened along the spatial dimension, and concatenated for all scales as input to the neck decoder, as shown in Figure 4(b). As analyzed in Section 5.1, this module only adds 1M parameters and greatly increases detection and segmentation accuracy without compromising inference speed.

4.2 Unified Query Representation Module

Object detection facilitates joint supervision of multi-task learning and instance segmentation [24], [25], [26]. Hence, we add a unified query representation (UQR) module [23] at the beginning of prediction heads, as shown on the right side of Figure 4(c).

Unfortunately, [DET] tokens of the DETR-based approaches only correspond to objects to detect. Consequently, [DET] tokens cannot be converted to 2D segmentation tasks, and it makes DETR-based approaches fail to perform object detection and instance segmentation tasks simultaneously [23]. To address this issue, Dong et al. [23] propose the UQR module, which transforms the ground-truth 2D binary mask of each object into the frequency domain using discrete cosine transform⁵ (DCT) [27], generating a ground-truth mask vector per object to predict. Given a ground-truth segmentation mask S , a ground-truth mask vector v is encoded by sampling the low-frequency components from $\mathbf{F} = \mathbf{A}\mathbf{S}\mathbf{A}^\top$, where \mathbf{A} is the transform matrix. Hence, the

4. We use 256 channel dimension for compatibility with a typical deformable transformer decoder [21].

5. Sparse coding and PCA can also be used for the conversion, but DCT is reported to show the best instance segmentation results [23].

final [DET] tokens are used directly to predict the ground-truth mask vector using a FFN, which outputs the predicted mask vectors \hat{v} . This procedure is conducted in parallel with classification and box regression for multi-task learning. As a result, the overall loss function for joint supervision can be formulated as $\ell_{joint} = \ell_{det} + \lambda_{seg} \ell_{l1}(v, \hat{v})$, where λ_{seg} is a coefficient for the instance segmentation task (see Appendix A.4.2 for details). In addition, at evaluation and testing time, the predicted mask vector \hat{v} can be converted to the estimated 2D binary mask \hat{S} through the inverse sampling and transformation $\hat{S} = A^{-1}\hat{F}(A^\top)^{-1}$, where \hat{F} is the inverse sampling result from \hat{v} .

4.3 IoU-aware and Token Labeling Losses

For dense prediction tasks, the model can capture more diverse aspects of provided inputs when properly incorporating multiple independent objectives. Thus, we introduce two additional objectives for training, namely IoU-aware loss and token labeling loss, eventually leading to a considerable performance gain with our proposed ViDT+ model. Note that they do not slow down the model inference speed at test time as they are activated only for training:

- IoU-aware Loss [8], [29]: Predicting the IoU score directly using the final [DET] token helps increase detection confidence, alleviating the mismatch between expected and ground-truth bounding boxes. Hence, we add a new FFN branch to predict the IoU score between the predicted bounding box \hat{b}_i and ground-truth one b_i . Then, the IoU-aware loss is formulated as

$$\ell_{aware} = \frac{1}{B} \sum_{i=1}^B \text{BCE}(\text{FFN}([\text{DET}]_i), \text{IoU}(b_i, \hat{b}_i)), \quad (4)$$

where $[\text{DET}]_i$ is the final [DET] token (returning from the neck decoder) corresponding to the i -th object; and B and BCE are the total number of objects in the input image and binary cross-entropy loss function, respectively.

- Token Labeling Loss [9], [29]: Token labeling allows to solve multiple token-level recognition problems by assigning each [PATCH] token with an individual location-specific supervision generated by a machine annotator. Here, we leverage the ground-truth segmentation mask to assign the location-specific class label per [PATCH] token. First, the segmentation mask is interpolated to align with the resolution of the feature map generated from the l -th stage of the body (*i.e.*, the resolution of x^l); hence, the interpolated mask S^l is regarded as token-level soft class labels for the l -th feature map. Then, the token labeling loss is formulated as

$$\ell_{token} = \frac{1}{L} \sum_{l=1}^L \frac{1}{P^l} \sum_{i=1}^{P^l} \text{Focal}(\text{FFN}([\text{PATCH}]_i^l), S^l[i]), \quad (5)$$

where $[\text{PATCH}]_i^l$ is the i -th [PATCH] token in the feature map x^l from the l -th stage of the body, and $S^l[i]$ returns the token-level soft label corresponding to the $[\text{PATCH}]_i^l$ token; and L and P^l is the number of scales and tokens in the feature map x^l , respectively. Focal is the focal loss function [30] and FFN is the classification layer.

These two losses are added with their respective coefficients to the joint learning loss if activated. We detail the complete objective of ViDT+ in Appendix A.4.2.

5 EVALUATION

In this section, we present thorough experimental results with evaluations against the state-of-the-art approaches.

Dataset. We carry out object detection experiments on the Microsoft COCO 2017 benchmark dataset [12]. All the fully transformer-based object detectors are trained on 118K training images and tested on 5K validation images following the standard setups [4].

Algorithms. We evaluate ViDT and ViDT+ against two fully transformer-based object detection pipelines, namely DETR (ViT) and YOLOS. Since DETR (ViT) follows the general pipeline of (Deformable) DETR by replacing its ResNet backbone with other ViT variants, we use one original ViT and one latest ViT variant, DeiT and Swin Transformer as its backbone without any modification. As YOLOS is strongly coupled with the original ViT architecture, only DeiT is used for evaluation. Table 2 summarizes all the ViT models pre-trained on ImageNet used for evaluation. Note that publicly available pre-trained models are used except for Swin-nano. In addition, we configure Swin-nano⁶ comparable to DeiT-tiny, which is trained on ImageNet with the identical setting. Overall, with respect to the number of parameters, Deit-tiny, -small, and -base are comparable to Swin-nano, -tiny, and -base, respectively. More details on the evaluated detectors are presented in Appendix A.2.

Implementation Details. All the algorithms are implemented using PyTorch and executed using four NVIDIA Tesla A100 GPUs. We train ViDT and ViDT+ using AdamW [31] with the same initial learning rate of 10^{-4} for its body, neck and head. In contrast, following the (Deformable) DETR setting, DETR (ViT) is trained with the initial learning rate of 10^{-5} for its pre-trained body (ViT backbone) and 10^{-4} for its neck and head. YOLOS and ViDT (w.o. Neck) are trained with the same initial learning rate of 5×10^{-5} , which is the original setting of YOLOS for the neck-free detector. We do not change any hyperparameters used in the transformer encoder and decoder for (Deformable) DETR; thus, the neck decoder of ViDT also consists of six deformable transformer layers using exactly the same hyperparameters. The number of learnable [DET] tokens are set to be 100 and 300 for ViDT and ViDT+, respectively. The effect of different number of [DET] tokens are discussed in Section 5.3.1. Auxiliary decoding loss and iterative box refinement are applied to the compared methods if applicable.

Regarding the resolution of input images, we use scale augmentation that resizes them such that the shortest side is at least 480 and at most 800 pixels while the longest is at most 1333 [20]. More details of the experiment configuration can be found in Appendix A.3–A.5. All the source code and trained models will be made available to the public at <https://github.com/naver-ai/vidt>.

5.1 Experiments with Microsoft COCO Benchmark

5.1.1 Object Detection

Table 3 shows evaluation results in terms of AP^{box} , FPS, and number of parameters, where two variants of DETR (ViT)

6. Swin-nano is designed such that the number of channels in the hidden layer is half that of Swin-tiny. Please see Appendix A.1.

TABLE 2. Summary on the ViT backbone. “dist.” is the distillation strategy for classification [28].

Backbone	Type (Size)	Train Data	Epochs	Resolution	Params	ImageNet Acc.
DeiT	DeiT-tiny (dist.)	ImageNet-1K	300	224 × 224	6M	74.5
	DeiT-small (dist.)	ImageNet-1K	300	224 × 224	22M	81.2
	DeiT-base (dist.)	ImageNet-1K	300	384 × 224	87M	85.2
Swin Transformer	Swin-nano	ImageNet-1K	300	224 × 224	6M	74.9
	Swin-tiny	ImageNet-1K	300	224 × 224	28M	81.2
	Swin-small	ImageNet-1K	300	224 × 224	50M	83.2
	Swin-base	ImageNet-22K	90	224 × 224	88M	86.3

TABLE 3. Comparison of ViDT and ViDT+ with other compared detectors on COCO2017 val set. Two neck-free detectors, YOLOS and ViDT (w.o. Neck) are trained for 150 epochs due to the slow convergence. FPS is measured with batch size 1 of 800 × 1333 resolution on a single Tesla A100 GPU, where the value inside the parentheses is measured with batch size 4 of the same resolution to maximize GPU utilization.

Method	Backbone	Epochs	AP ^{box}	AP ₅₀ ^{box}	AP ₇₅ ^{box}	AP _S ^{box}	AP _M ^{box}	AP _L ^{box}	Param.	FPS
DETR	DeiT-tiny	50	30.0	49.2	30.5	9.9	30.8	50.6	23M	24.4 (28.5)
	DeiT-small	50	32.4	52.5	33.2	11.3	33.5	53.7	39M	17.8 (20.3)
	DeiT-base	50	37.1	59.2	38.4	14.7	39.4	52.9	104M	11.1 (14.2)
	Swin-nano	50	27.8	47.5	27.4	9.0	29.2	44.9	24M	50.6 (86.1)
	Swin-tiny	50	34.1	55.1	35.3	12.7	35.9	54.2	45M	39.9 (58.6)
	Swin-small	50	37.6	59.0	39.0	15.9	40.1	58.9	66M	28.7 (39.2)
	Swin-base	50	40.7	62.9	42.7	18.3	44.1	62.4	104M	23.4 (30.9)
Deformable DETR	DeiT-tiny	50	40.8	60.1	43.6	21.4	43.4	58.2	18M	25.1 (36.4)
	DeiT-small	50	43.6	63.7	46.5	23.3	47.1	62.1	34M	18.0 (24.0)
	DeiT-base	50	46.4	67.3	49.4	26.7	50.1	65.4	99M	11.3 (13.8)
	Swin-nano	50	43.1	61.4	46.3	25.9	45.2	59.4	18M	14.6 (17.2)
	Swin-tiny	50	47.0	66.8	50.8	28.1	49.8	63.9	39M	13.4 (15.7)
	Swin-small	50	49.0	68.9	52.9	30.3	52.8	66.6	60M	11.9 (13.9)
	Swin-base	50	51.4	71.7	56.2	34.5	55.1	67.5	98M	10.9 (12.7)
YOLOS	DeiT-tiny	150	30.4	48.6	31.1	12.4	31.8	48.2	6M	52.5 (61.3)
	DeiT-small	150	36.1	55.7	37.6	15.6	38.4	55.3	30M	14.0 (24.8)
	DeiT-base	150	42.0	62.2	44.5	19.5	45.3	62.1	104M	7.0 (12.2)
ViDT (w.o. Neck)	Swin-nano	150	28.7	48.6	28.5	12.3	30.7	44.1	7M	72.4 (96.9)
	Swin-tiny	150	36.3	56.3	37.8	16.4	39.0	54.3	29M	51.8 (60.4)
	Swin-small	150	41.6	62.7	43.9	20.1	45.4	59.8	52M	33.5 (38.4)
	Swin-base	150	43.2	64.2	45.9	21.9	46.9	63.2	91M	26.3 (29.6)
ViDT	Swin-nano	50	40.4	59.6	43.3	23.2	42.5	55.8	15M	40.8 (76.0)
	Swin-tiny	50	44.8	64.5	48.7	25.9	47.6	62.1	37M	33.5 (51.2)
	Swin-small	50	47.5	67.7	51.4	29.2	50.7	64.8	60M	24.7 (34.6)
	Swin-base	50	49.2	69.4	53.1	30.6	52.6	66.9	99M	20.5 (27.1)
ViDT+	Swin-nano	50	45.3	62.3	48.9	27.3	48.2	61.5	16M	37.6 (69.3)
	Swin-tiny	50	49.7	67.7	54.2	31.6	53.4	65.9	38M	30.4 (47.6)
	Swin-small	50	51.2	69.5	55.9	33.8	54.5	67.8	61M	22.6 (32.4)
	Swin-base	50	53.2	71.6	58.3	36.0	57.1	69.2	100M	19.3 (25.7)

are simply named DETR and Deformable DETR, respectively. A summary plot is also provided in Figure 1.

Highlights. Among detection-only methods, ViDT achieves the best trade-off between AP^{box} and FPS. With its high scalability, it performs well even with Swin-base of 0.1 billion parameters, which is 2x faster than Deformable DETR with similar AP^{box}. In addition, ViDT shows 40.4AP^{box} only with 15M parameters; it is 6.3–12.6AP^{box} higher than those of DETR (swin-nano) and DETR (swin-tiny), which exhibit similar FPS of 39.9–50.6. ViDT+ shows considerable AP^{box} improvements over the vanilla ViDT with a little decrease in FPS. For example, ViDT+ (swin-nano) achieves 45.3AP^{box} only with 16M parameters, which is even higher than that of ViDT (swin-tiny) with 37M parameters. In particular, to the best of our knowledge, 53.2AP^{box} of VIDT+ (Swin-base) is the highest among existing fully transformer-based detectors. In the detailed analysis below, we only compare the

vanilla ViDT with other detection-only architectures since ViDT+ requires multiple extended modules, such as EPFF and UQR, and leverages extra knowledge from instance segmentation.

ViDT vs. Deformable DETR. Thanks to the use of multi-scale features, Deformable DETR exhibits high detection performance in general. Nevertheless, its encoder and decoder structure in the neck becomes a critical bottleneck in computation. In particular, the encoder with multi-layer deformable transformers adds considerable overhead to transform multi-scale features by attention. Thus, it achieves low FPS although it achieves higher AP^{box} with a relatively small number of parameters. In contrast, ViDT removes the need for a transformer encoder in the neck by using the Swin Transformer with RAM as its body, directly extracting multi-scale features suitable for object detection.

ViDT (w.o. Neck) vs. YOLOS. For the comparison with

TABLE 4. Evaluations of ViDT with DETR and Deformable DETR using the CNN backbone (ResNet-50) on COCO2017 val set.

Method	Backbone	Epochs	AP^{box}	AP_{50}^{box}	AP_{75}^{box}	AP_S^{box}	AP_M^{box}	AP_L^{box}	Param.	FPS
DETR	ResNet-50	500	42.0	62.4	44.2	20.5	45.8	61.1	41M	54.1 (99.3)
DETR-DC5	ResNet-50	500	43.3	63.1	45.9	22.5	47.3	61.1	41M	29.7 (34.3)
DETR-DC5	ResNet-50	50	35.3	55.7	36.8	15.2	37.5	53.6	41M	29.7 (34.3)
Deformable DETR	ResNet-50	50	45.4	64.7	49.0	26.8	48.3	61.7	40M	27.2 (43.6)
ViDT	Swin-tiny	50	44.8	64.5	48.7	25.9	47.6	62.1	37M	33.5 (51.2)
ViDT	Swin-tiny	150	47.2	66.7	51.4	28.4	50.2	64.7	37M	33.5 (51.2)

TABLE 5. Comparison of ViDT+ with other multi-task learning methods for instance segmentation on COCO2017 val set. To distinguish AP for detection and segmentation, we put ‘seg’ and ‘box’ superscripts into AP at the first row, respectively. Except ViDT+, all the results are borrowed from [23].

Method	Backbone	Epochs	AP^{seg}	AP_S^{seg}	AP_M^{seg}	AP_L^{seg}	AP^{box}	AP_S^{box}	AP_M^{box}	AP_L^{box}
Mask R-CNN [24]	ResNet-50+FPN	36	37.5	21.1	39.6	48.3	41.3	24.2	43.6	51.7
HTC [32]	ResNet-50+FPN	36	39.7	22.6	42.2	50.6	44.9	-	-	-
SOLOv2 [33]	ResNet-50+FPN	72	38.8	16.5	41.7	56.2	40.4	20.5	44.2	53.9
QueryInst [34]	ResNet-50+FPN	36	40.6	23.4	42.5	52.8	45.6	-	-	-
SOLQ [23]	ResNet50	50	39.7	21.5	42.5	53.1	47.8	27.6	50.9	61.6
ViDT+	Swin-tiny	50	39.5	21.5	43.4	58.2	49.7	31.6	53.4	65.9
Mask R-CNN [24]	ResNet-101+FPN	36	38.8	21.8	41.4	50.5	41.3	24.2	43.6	51.7
HTC [32]	ResNet-101+FPN	36	40.8	23.0	43.5	52.6	44.3	-	-	-
SOLOv2 [33]	ResNet-101+FPN	72	39.7	17.3	42.9	57.4	42.6	22.3	46.7	56.3
QueryInst [34]	ResNet-101+FPN	36	42.8	24.6	45.0	55.5	48.1	-	-	-
SOLQ [23]	ResNet-101	50	40.9	22.5	43.8	54.6	48.7	28.6	51.7	63.1
ViDT+	Swin-small	50	40.8	22.6	44.3	60.1	51.2	33.8	54.5	67.8

TABLE 6. AP and FPS comparison with different selective cross-attention strategies.

Stage Ids	{1, 2, 3, 4}		{2, 3, 4}		{3, 4}		{4}		{}	
Metric	AP^{box}	FPS	AP^{box}	FPS	AP^{box}	FPS	AP^{box}	FPS	AP^{box}	FPS
ViDT w.o. Neck	29.0	46.9	28.8	55.2	28.5	59.5	28.7	72.4	FAIL	75.8
ViDT w. Neck	40.3	31.6	40.1	34.2	40.3	36.1	40.4	40.8	37.1	41.2

TABLE 7. Results for different spatial encodings for [DET] \times [PATCH] cross-attention.

Method	None	Pre-addition		Post-addition	
Type	None	Sin.	Learn.	Sin.	Learn.
AP^{box}	23.7	28.7	27.4	28.0	24.1

YOLOS, we train ViDT without using its neck component. These two neck-free detectors achieve relatively low AP^{box} compared with other detectors in general. In terms of speed, YOLOS exhibits much lower FPS than ViDT (w.o. Neck) because of its quadratic computational complexity for attention. However, as ViDT (w.o. Neck) extends the Swin Transformers with RAM, it requires linear complexity for attention. Hence, it achieves comparable AP^{box} as YOLOS for various backbone sizes, but with higher FPS.

Comparison with Other Possible Integration. It is of great interest to see whether better integration can also be achieved by (1) Deformable DETR without its neck encoder as its neck decoder also has [DET] \times [PATCH] cross-attention, or (2) YOLOS with VIDT’s neck decoder because of the use of multiple auxiliary techniques. Such integration is actually not effective; the former significantly drops AP^{box} , while the latter has a much greater drop in FPS than an increase in AP^{box} . The detailed analysis can be found in Appendix B.2.

Comparison with Detectors using CNN Backbone. We compare ViDT with (Deformable) DETR using the ResNet-50 backbone, as summarized in Table 4, where all the results except ViDT are borrowed from [4], [21], and DETR-DC5 is a

TABLE 8. Effect of auxiliary decoding loss and iterative box refinement loss with YOLOS (DeiT-tiny) and ViDT (Swin-nano).

	Aux. ℓ	Box Ref.	Neck	AP^{box}	Δ
YOLOS	✓			30.4	
	✓	✓		29.2	-1.2
ViDT				20.1	-10.3
	✓			28.7	
	✓	✓		27.2	-1.6
	✓	✓	✓	22.9	-5.9
	✓		✓	36.2	+7.4
	✓	✓	✓	40.4	+11.6

modification of DETR to use a dilated convolution at the last stage in ResNet. For fair comparisons, we use ViDT (Swin-tiny) with similar parameter numbers in the following experiments. In general, ViDT shows better trade-offs between AP^{box} and FPS even compared with (Deformable) DETR with the ResNet-50. Specifically, ViDT achieves considerably higher FPS than those of DETR-DC5 and Deformable DETR with competitive AP^{box} . When training ViDT for 150 epochs, ViDT outperforms other compared methods using the ResNet-50 backbone in terms of both AP^{box} and FPS.

5.1.2 Instance Segmentation

Table 5 shows evaluation results of ViDT+ with other methods for multi-task learning of object detection and instance segmentation, based on AP^{box} and AP^{seg} . For fair comparisons w.r.t the number of parameters, Swin-tiny and Swin-

TABLE 9. Effect of additional losses and modules for ViDT+ with Swin-nano.

IoU-aware	Token Label	EPFF	UQR	Epoch	AP ^{box}	AP ₅₀ ^{box}	AP ₇₅ ^{box}	AP _S ^{box}	AP _M ^{box}	AP _L ^{box}	Param.	FPS
✓				50	40.4	59.6	43.3	23.2	42.5	55.8	15M	40.8 (76.0)
✓	✓			50	41.0	59.5	44.1	22.8	43.7	56.7	15M	40.8 (76.0)
✓	✓	✓		50	41.2	59.5	44.4	23.5	44.0	57.5	15M	40.8 (76.0)
✓	✓	✓	✓	50	42.5	60.9	45.3	23.5	45.3	59.0	16M	37.6 (69.3)
✓	✓	✓	✓	50	45.3	62.3	48.9	27.3	48.2	61.5	16M	37.6 (69.3)

TABLE 10. Performance change with different number of detection tokens regarding AP^{box}, Param, and FPS.

Model Metric	ViDT (Swin-nano)			ViDT (Swin-tiny)		
	AP ^{box}	Param.	FPS	AP ^{box}	Param.	FPS
100 Tokens	40.4	15M	40.8	44.8	37M	33.5
300 Tokens	42.1	15M	40.4	46.7	38M	33.0
500 Tokens	42.3	16M	40.0	46.9	38M	32.4

TABLE 11. Performance change with different number of training epochs regarding AP^{box}, Param, and FPS.

Model Metric	ViDT (Swin-nano)			ViDT (Swin-tiny)		
	AP ^{box}	Param.	FPS	AP ^{box}	Param.	FPS
50 Epochs	40.4	15M	40.8	44.8	37M	33.5
150 Epochs	42.6	15M	40.8	47.2	37M	33.5

small backbones are used for ViDT+, which have similar numbers of parameters to ResNet-50 and ResNet-101.

For the instance segmentation task, ViDT+ obtains AP^{seg} comparable to the state-of-the-art methods, such as QueryInst [34], and SOLQ [23]. It is noteworthy that ViDT+ achieves the best segmentation performance for medium- and large-size objects, although it is not the best for small-size objects. This can be attributed to that vector encoding of the 2D segmentation mask using DTC loses detailed information about small objects. In addition, ViDT+ achieves a considerable gain for the object detection task, which can be justified by its AP^{box} much higher than other multi-task learning methods. Quantitatively, the detection performance of ViDT+ is 1.9–9.9 AP^{box} higher than other methods. These results show that ViDT can be easily extended to ViDT+ with better detection performance.

5.2 Ablation Study

5.2.1 Reconfigured Attention Module

We extend the Swin Transformer with RAM to extract fine-grained features for object detection without maintaining an additional transformer encoder in the neck. We provide an ablation study on the two main considerations for RAM, which lead to high accuracy and speed. To reduce the effect of secondary factors, we mainly use our neck-free version, ViDT (w.o. Neck), for the ablation study.

Selective [DET] × [PATCH] Cross-Attention. The addition of cross-attention to the Swin Transformer inevitably entails computational overheads, particularly when the number of [PATCH] is large. To alleviate such overheads, we selectively enable cross-attention in RAM at the last stage of the Swin Transformer; this is shown to greatly improve FPS, but barely drop AP^{box}. Table 6 summarizes AP^{box} and FPS when using different selective strategies for the cross-attention, where the Swin Transformer consists of four stages. It is interesting that all the strategies exhibit similar AP^{box} as long as cross-attention is activated at the last stage. Since

TABLE 12. Performance trade-off by decoding layer drop regarding AP^{box}, Param, and FPS.

Model Metric	ViDT (Swin-nano)			ViDT (Swin-tiny)		
	AP ^{box}	Param.	FPS	AP ^{box}	Param.	FPS
0 Drop	40.4	15M	40.8	44.8	37M	33.5
1 Drop	40.2	14M	43.7	44.8	36M	35.2
2 Drop	40.0	13M	46.7	44.5	35M	36.7
3 Drop	38.6	12M	48.8	43.6	34M	38.5
4 Drop	36.8	11M	55.3	41.9	33M	41.0
5 Drop	32.5	9M	57.6	38.0	32M	43.2

features are extracted in a bottom-up manner as they go through the stages, it seems difficult to directly obtain useful information about the target object at the low level of stages. Thus, only using the last stage is the best design choice in terms of high AP^{box} and FPS due to the smallest number of [PATCH] tokens. Meanwhile, the detection fails completely or the performance significantly drops if all the stages are not involved due to the lack of interaction between [DET] and [PATCH] tokens that spatial positional encoding is associated with. A more detailed analysis of the cross- and self-attention is provided in appendices B.3 and B.4.

Spatial Positional Encoding. Spatial positional encoding is essential for [DET] × [PATCH] attention in RAM. Typically, the spatial encoding can be added to the [PATCH] tokens before or after the projection layer (see Figure 3), and we call the former “pre-addition” and the latter “post-addition”. For each one, we can design the encoding in a sinusoidal or learnable manner [4]. Table 7 contrasts the results with different spatial positional encodings with ViDT (w.o. Neck). Overall, pre-addition results in performance improvement higher than post-addition, and specifically, the sinusoidal encoding is better than the learnable one. Thus, the 2D inductive bias of the sinusoidal spatial encoding is more helpful in object detection. In particular, pre-addition with the sinusoidal encoding increases AP^{box} by 5.0 compared to not using any encoding.

5.2.2 Auxiliary Decoding and Iterative Box Refinement

We analyze the performance improvement of auxiliary decoding loss and iterative box refinement. To validate the efficacy of these two modules, we extend them for the neck-free detector such as YOLOS where each is applied to the encoding layers in the body, as opposed to the conventional way of using the decoding layers in the neck. Table 8 shows the performance of the two neck-free detectors, YOLOS and ViDT (w.o. Neck), decreases considerably with the two techniques. The use of these two modules in the encoding layers is likely to negatively affect the feature extraction of the transformer encoder. In contrast, an opposite trend is observed with the neck component. Since the neck decoder is decoupled with the feature extraction in the body, the two techniques make a synergistic effect and

TABLE 13. Complete component analysis. “Brown” and “Teal” colors indicate the performance of vanilla ViDT and its extension to ViDT+, respectively. “Violet” color indicates the performance of fully optimized ViDT+.

#	Added Module/Technique	ViDT+ (Swin-nano)			ViDT+ (Swin-tiny)			ViDT+ (Swin-small)		
		AP ^{box}	Param.	FPS	AP ^{box}	Param.	FPS	AP ^{box}	Param.	FPS
(1)	+ RAM	28.7	7M	72.4	36.3	29M	51.8	41.6	52M	33.5
(2)	+ Encoder-free Neck	40.4	15M	40.8	44.8	37M	33.5	47.5	60M	24.7
(3)	+ IoU-aware & Token Label	41.0	15M	40.8	45.9	37M	33.5	48.5	60M	24.7
(4)	+ EPFF Module	42.5	16M	38.0	47.1	38M	30.9	49.3	61M	23.0
(5)	+ UQR Module	43.9	16M	38.0	47.9	38M	30.9	50.1	61M	23.0
(6)	+ 300 [DET] Tokens	45.3	16M	37.6	49.7	38M	30.4	51.2	61M	22.6
(7)	+ 150 Training Epochs	47.6	16M	37.6	51.4	38M	30.4	52.3	61M	22.6
(8)	+ Decoding Layer Drop	47.0	14M	41.9	50.8	36M	33.9	51.8	59M	24.6

thus show significant improvements in AP^{box}. These results demonstrate the use of the neck decoder in ViDT to improve object detection performance.

5.2.3 Components for extension to ViDT+

For the extension to ViDT+, two additional modules are incorporated on top of the vanilla ViDT, *i.e.*, EPFF module for pyramid feature fusion and UQR module for multi-task learning, and also two independent objectives are added for better optimization, *i.e.*, IoU-aware and token labeling losses. Table 9 summarizes the performance improvement of ViDT(Swin-nano) when adding them incrementally to the vanilla ViDT; the trend of performance improvement is almost the same for Swin backbones of difference sizes. IoU-aware and token labeling losses are added first since they do not affect the inference time. Then, the remaining EPFF and UQR modules are added to support effective multi-task learning. With the Swin-nano backbone, the extension to ViDT+ only adds 1M parameters but AP^{box} improves from 40.4 to 45.3. This is a significant performance gain for the better trade-off between accuracy and speed, considering that the runtime performance dropped by 3.2 FPS.

5.3 Optimization to Performance Boosting

The number of detection tokens and training epochs affect the detection performance w.r.t AP^{box} and FPS. Hence, we analyze the performance change by using different numbers of them. In addition, we introduce a decoding layer dropping scheme, which further increases the FPS of the ViDT model by simply dropping a few decoding layers in the neck component without compromising AP^{box}.

5.3.1 Increasing the Number of Detection Tokens

Table 10 shows the performance change of ViDT with different number of [DET] tokens. When used 200 more [DET] tokens, *i.e.*, 300 tokens, the detection accuracy of ViDT improves by 1.7–1.9AP^{box} with little increase in FPS. Thus, the accuracy gain outweighs the loss of efficiency. However, there is only a slight improvement in AP^{box} when using 500 [DET] rather than 300 [DET] tokens, although FPS decreases almost linearly. Therefore, we can use 300 [DET] tokens for a better trade-off between accuracy and speed.

5.3.2 Increasing the Number of Training Epochs

Table 11 shows the performance of ViDT trained with a different number of epochs. The increase in the number of training epochs provides performance improvement in ViDT. When used 3 times longer training epochs, *i.e.*, 50

epochs → 150 epochs, the detection accuracy of ViDT improves by 2.2–2.4AP^{box} without any performance drop in FPS. When the computational budget allows, ViDT has the potential to achieve better detection performance. We report the performance of ViDT+ with longer training epochs in our complete analysis of Section 5.4.

5.3.3 Decoding Layer Dropping

ViDT has six layers of transformers as its neck decoder. We emphasize that not all layers of the decoder are required at inference time for high performance. Table 12 shows the performance of ViDT when dropping its decoding layer one by one from the top in the inference step. Although there is a trade-off relationship between accuracy and speed as the layers are detached from the model, there is no significant AP^{box} drop even when the two layers are removed. Although this scheme is not designed for performance evaluation with other methods in Table 3 with other methods, we can accelerate the inference speed of a trained ViDT model to over 10% by dropping its two decoding layers without a much decrease in AP^{box}.

5.4 Complete Component Analysis

In this section, we combine all the proposed components (even with longer training epochs and decoding layer drop) to achieve high accuracy and speed for object detection. As summarized in Table 13, there are eight components for extension: (1) RAM to extend Swin Transformer as a standalone object detector, (2) the neck decoder to exploit multi-scale features with two auxiliary techniques, (3) the IoU-aware and token labeling losses for fine-grained supervision per token, (4) the EPFF module to fuse multi-scale features non-linearly via pyramid feature fusion, (5) the UQR model for extra supervision from multi-task learning, (6) the use of more detection tokens, (7) the use of longer training epochs, and (8) decoding layer drop to further accelerate inference speed. For the full model, it achieves 47.0AP^{box} with very high FPS by only using 14M parameters when using Swin-nano as its backbone. Further, it achieves 50.8–51.8AP^{box} with reasonable FPS when using Swin-tiny and Swin-small backbones. This indicates that a fully transformer-based object detector has the potential to be used as a generic object detector when further developed in the future.

6 CONCLUSION

In this work, we explore the integration of vision and detection transformers to build an effective and efficient

object detector. The proposed ViDT significantly improves the scalability and flexibility of transformer models to achieve high accuracy and inference speed. The computational complexity of its attention modules is linear w.r.t. image size, and ViDT synergizes several essential techniques to boost the detection performance. Further, it can be easily extended to support multi-task learning of object detection and instance segmentation. The joint learning framework named ViDT+ improves its detection performance considerably without compromising its efficiency. On the Microsoft COCO benchmark, ViDT+ achieves 53.2AP^{box} with a large Swin-base backbone, and 47.0AP^{box} with the smallest Swin-nano backbone using only 14M parameters, suggesting the potential of using the proposed model for complex computer vision tasks.

REFERENCES

- [1] C. Papageorgiou and T. Poggio, "A trainable system for object detection," *International Journal of Computer Vision*, vol. 38, no. 1, pp. 15–33, 2000. [1](#) [2](#)
- [2] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *International Journal of Computer Vision*, vol. 128, no. 2, pp. 261–318, 2020. [1](#) [2](#)
- [3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017. [1](#) [5](#) [3](#)
- [4] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *ECCV*, 2020. [1](#) [3](#) [5](#) [6](#) [7](#) [9](#) [10](#)
- [5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2021. [1](#) [3](#)
- [6] Y. Fang, B. Liao, X. Wang, J. Fang, J. Qi, R. Wu, J. Niu, and W. Liu, "You only look at one sequence: Rethinking transformer in vision through object detection," *arXiv preprint arXiv:2106.00666*, 2021. [2](#), [3](#), [1](#)
- [7] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin Transformer: Hierarchical vision transformer using shifted windows," in *ICCV*, 2021. [2](#), [3](#), [5](#)
- [8] S. Wu, X. Li, and X. Wang, "IoU-aware single-stage object detector for accurate localization," *Image and Vision Computing*, vol. 97, p. 103911, 2020. [2](#), [7](#)
- [9] Z.-H. Jiang, Q. Hou, L. Yuan, D. Zhou, Y. Shi, X. Jin, A. Wang, and J. Feng, "All tokens matter: Token labeling for training better vision transformers," in *NeurIPS*, 2021. [2](#) [7](#)
- [10] W. Xu, Y. Xu, T. Chang, and Z. Tu, "Co-scale conv-attentional image transformers," in *ICCV*, 2021. [2](#)
- [11] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, "Pvtv2: Improved baselines with pyramid vision transformer," *Computational Visual Media*, vol. 8, no. 3, pp. 1–10, 2022. [2](#)
- [12] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *ECCV*, 2014. [2](#) [7](#)
- [13] H. Song, D. Sun, S. Chun, V. Jampani, D. Han, B. Heo, W. Kim, and M.-H. Yang, "ViDT: An efficient and effective fully transformer-based object detector," in *ICLR*, 2022. [2](#)
- [14] Y. Li, H. Mao, R. Girshick, and K. He, "Exploring plain vision transformer backbones for object detection," *arXiv preprint arXiv:2203.16527*, 2022. [2](#)
- [15] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "Yolox: Exceeding yolo series in 2021," *arXiv preprint arXiv:2107.08430*, 2021. [2](#)
- [16] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *ECCV*, 2016, pp. 21–37. [2](#)
- [17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016, pp. 779–788. [2](#)
- [18] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *NeurIPS*, 2015. [2](#) [3](#)
- [19] B. Heo, S. Yun, D. Han, S. Chun, J. Choe, and S. J. Oh, "Rethinking spatial dimensions of vision transformers," in *ICCV*, 2021. [3](#), [1](#)
- [20] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2," <https://github.com/facebookresearch/detectron2>, 2019. [3](#) [7](#)
- [21] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: Deformable transformers for end-to-end object detection," in *ICLR*, 2021. [3](#) [5](#), [6](#), [9](#), [1](#)
- [22] H. Hu, Z. Zhang, Z. Xie, and S. Lin, "Local relation networks for image recognition," in *CVPR*, 2019. [4](#)
- [23] B. Dong, F. Zeng, T. Wang, X. Zhang, and Y. Wei, "SOLQ: Segmenting objects by learning queries," in *NeurIPS*, 2021. [6](#), [9](#), [10](#)
- [24] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *ICCV*, 2017. [6](#), [9](#)
- [25] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Simultaneous detection and segmentation," in *ECCV*, 2014. [6](#)
- [26] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 1, pp. 142–158, 2015. [6](#)
- [27] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE transactions on Computers*, vol. 100, no. 1, pp. 90–93, 1974. [6](#)
- [28] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *ICML*, 2021. [8](#)
- [29] J. Lin, X. Mao, Y. Chen, L. Xu, Y. He *et al.*, "D²ETR: Decoder-only detr with computationally efficient cross-scale attention," 2021. [7](#)
- [30] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *CVPR*, 2017. [7](#), [1](#)
- [31] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *ICLR*, 2019. [7](#), [2](#)
- [32] K. Chen, J. Pang, J. Wang, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Shi, W. Ouyang *et al.*, "Hybrid task cascade for instance segmentation," in *CVPR*, 2019. [9](#)
- [33] X. Wang, R. Zhang, T. Kong, L. Li, and C. Shen, "Solov2: Dynamic and fast instance segmentation," in *NeurIPS*, 2020. [9](#)
- [34] Y. Fang, S. Yang, X. Wang, Y. Li, C. Fang, Y. Shan, B. Feng, and W. Liu, "Instances as queries," in *ICCV*, 2021. [9](#), [10](#)
- [35] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions," in *ICCV*, 2021. [1](#)
- [36] C.-F. Chen, Q. Fan, and R. Panda, "CrossViT: Cross-attention multi-scale vision transformer for image classification," in *ICCV*, 2021. [1](#)
- [37] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *CVPR*, 2019. [1](#)



Hwanjun Song is a Research Scientist at the NAVER AI Lab. He worked as a Research Intern at Google Research in 2020 and received his Ph.D. degree in the Graduate School of Data Science from KAIST, Daejeon, Korea, in 2021. He is interested in designing advanced methodologies to handle data scale and quality issues, which are two main real-world challenges for AI. He was sponsored by Microsoft through Azure for Research from 2016 to 2018, and received the Qualcomm Innovation Award in 2019.



Deqing Sun is a staff research scientist and manager at Google working on computer vision and machine learning. He received a Ph.D. degree in Computer Science from Brown University. He served as an area chair for CVPR/ECCV/BMVC, and co-organized several workshops/tutorials at CVPR/ECCV/SIGGRAPH. He is a recipient of the best paper honorable mention award at CVPR 2018, the first prize in the robust optical flow competition at CVPR 2018 and ECCV 2020, the PAMI Young Researcher award in 2020, and the Longuet-Higgins prize at CVPR 2020.



Sanghyuk Chun is a lead research scientist at the NAVER AI Lab. He was a research engineer at an advanced recommendation team in Kakao Corp from 2016 to 2018. He received his Master's and Bachelor's degrees in Electronical Engineering from KAIST, Daejeon, Korea, in 2016 and 2014, respectively. His research interests focus on reliable machine learning and vision-and-language.



Varun Jampani is a researcher at Google Research working in the areas of machine learning and computer vision. His main research interests include self-supervised visual discovery, content-adaptive neural networks, and novel view synthesis. He obtained his PhD with highest honors at MPI for Intelligent Systems, Germany. He obtained his BTech and MS from IIT-Hyderabad, India, where he was a gold medalist. He received Best Paper Honorable Mention award at CVPR2018.



Dongyo Han is a lead research scientist at NAVER AI Lab. His current research interests lie in machine learning and computer vision, in particular, novel deep neural networks design and training methods. He received his B.S., M.S., and Ph.D. degrees in electrical engineering from Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2011, 2013, and 2018, respectively.



Byeongho Heo received the bachelor's and Ph.D. degrees in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2012 and 2019, respectively. In 2019, he joined the AI LAB, NAVER Corporation, as a Research Scientist, where he is currently working. His current research interests include vision transformer, knowledge distillation, image classification, and optimizer for deep learning.



Wonjae Kim is a research scientist at NAVER AI Lab. Before joining NAVER, He worked as a research scientist at Kakao corporation from 2018 to 2021. He received his Master's and Bachelor's degrees in computer science and engineering from Seoul national university, in 2018 and 2016, respectively. His research interests include vision-and-language representation learning, human-computer interaction, and information visualization.



Ming-Hsuan Yang is a professor of Electrical Engineering and Computer Science with the University of California, Merced, CA, USA. He received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign, USA, in 2000. He served as an Associate Editor of the IEEE Transactions on Pattern Analysis and Machine Intelligence from 2007 to 2011, and is an Associate Editor of the International Journal of Computer Vision, and Image and Vision Computing. He received the NSF CAREER Award in 2012, and Google Faculty Award in 2009. He is a fellow of the IEEE and the ACM..

APPENDIX A EXPERIMENTAL DETAILS

A.1 Swin-nano Architecture

Due to the absence of Swin models comparable to Deit-tiny, we configure Swin-nano, which is a $0.25\times$ model of Swin-tiny such that it has 6M training parameters comparable to Deit-tiny. Table 1 summarizes the configuration of Swin Transformer models available, including the newly introduced Swin-nano; S1–S4 indicates the four stages in Swin Transformer. The performance of all the pre-trained Swin Transformer models are summarized in Table 2 in the manuscript.

A.2 Detection Pipelines of All Compared Detectors

All the compared fully transformer-based detectors are composed of either (1) *body–neck–head* or (2) *body–head* structure, as summarized in Table 2. The main difference of ViDT is the use of reconfigured attention modules(RAM) for Swin Transformer, allowing the extraction of fine-grained detection features directly from the input image. Thus, Swin Transformer is extended to a standalone object detector called ViDT(w.o. Neck). Further, its extension to ViDT allows to use multi-scale features and multiple essential techniques for better detection, such as auxiliary decoding loss and iterative box refinement, by only maintaining a transformer decoder at the neck. Except for the two neck-free detector, YOLOS and ViDT(w.o. Neck), all the pipelines maintain multiple FFNs; that is, a single FFNs for each decoding layer at the neck for box regression and classification.

We believe that our proposed RAM can be combined with even other latest efficient vision transformer architectures, such as PiT [19], PVT [35] and Cross-ViT [36]. We leave this as future work.

A.3 Hyperparameters of Neck Transformers

The transformer decoder at the neck in ViDT introduces multiple hyperparameters. We follow exactly the same setting used in Deformable DETR. Specifically, we use six layers of deformable transformers with width 256; thus, the channel dimension of the [PATCH] and [DET] tokens extracted from Swin Transformer are reduced to 256 to be utilized as compact inputs to the decoder transformer. For each transformer layer, multi-head attention with eight heads is applied, followed by the point-wise FFNs of 1024 hidden units. Furthermore, an additive dropout of 0.1 is applied before the layer normalization. All the weights in the decoder are initialized with Xavier initialization. For (Deformable) DETR, the tranformer decoder receives a fixed number of learnable detection tokens. We set the number of detection tokens to 100, which is the same number used for YOLOS and ViDT.

A.4 Implementation

A.4.1 Detection Head for Prediction

The last [DET] tokens produced by the body or neck are fed to a 3-layer FFNs for bounding box regression and linear projection for classification,

$$\hat{B} = \text{FFN}_{3\text{-layer}}([\text{DET}]) \quad \text{and} \quad \hat{P} = \text{Linear}([\text{DET}]). \quad (1)$$

TABLE 1. Swin Transformer Architecture, where S1, S2, S3, and S4 stand for the number of stages.

Model Name	Channel Dim.	S1	S2	S3	S4
Swin-nano	48	2	2	6	2
Swin-tiny	96	2	2	6	2
Swin-small	128	2	2	18	2
Swin-base	192	2	2	18	2

For box regression, the FFNs produce the bounding box coordinates for d objects, $\hat{B} \in [0, 1]^{d \times 4}$, that encodes the normalized box center coordinates along with its width and height. For classification, the linear projection uses a softmax function to produce the classification probabilities for all possible classes including the background class, $\hat{P} \in [0, 1]^{d \times (c+1)}$, where c is the number of object classes. When deformable attention is used on the neck in Table 2, only c classes are considered without the background class for classification. This is the original setting used in DETR, YOLOS [4], [6] and Deformable DETR [21].

A.4.2 Loss Function for Training

All the methods adopts the loss function of (Deformable) DETR. Since the detection head return a fixed-size set of d bounding boxes, where d is usually larger than the number of actual objects in an image, Hungarian matching is used to find a bipartite matching between the predicted box \hat{B} and the ground-truth box B . In total, there are three types of training loss: a classification loss ℓ_{cl} ⁷, a box distance ℓ_{l_1} , and a GIoU loss ℓ_{iou} [37],

$$\begin{aligned} \ell_{cl}(i) &= -\log \hat{P}_{\sigma(i), c_i}, \quad \ell_{l_1}(i) = \|B_i - \hat{B}_{\sigma(i)}\|_1, \text{ and} \\ \ell_{iou}(i) &= 1 - \left(\frac{|B_i \cap \hat{B}_{\sigma(i)}|}{|B_i \cup \hat{B}_{\sigma(i)}|} - \frac{|\mathbf{B}(B_i, \hat{B}_{\sigma(i)}) \setminus B_i \cup \hat{B}_{\sigma(i)}|}{|\mathbf{B}(B_i, \hat{B}_{\sigma(i)})|} \right), \end{aligned} \quad (2)$$

where c_i and $\sigma(i)$ are the target class label and bipartite assignment of the i -th ground-truth box, and \mathbf{B} returns the largest box containing two given boxes. Thus, the final loss of object detection is a linear combination of the three types of training loss,

$$\ell_{det} = \lambda_{cl}\ell_{cl} + \lambda_{l_1}\ell_{l_1} + \lambda_{iou}\ell_{iou}. \quad (3)$$

The coefficient for each training loss is set to be $\lambda_{cl} = 1$, $\lambda_{l_1} = 5$, and $\lambda_{iou} = 2$. If we leverage auxiliary decoding loss, the final loss is computed for every detection head separately and merged with equal importance. Furthermore, if ViDT is extended to a multi-task learning framework named ViDT+, the model is trained via the joint-learning loss composed of detection and segmentation losses, introducing three additional coefficients λ_{seg} , λ_{aware} , and λ_{token} ,

$$\begin{aligned} \ell_{joint} &= \ell_{det} + \lambda_{seg}\ell_{seg} \\ &\quad + \lambda_{aware}\ell_{aware} + \lambda_{token}\ell_{token}, \end{aligned} \quad (4)$$

where ℓ_{seg} is the l1 loss between predicted and ground-truth segmentation vectors, ℓ_{aware} and ℓ_{token} are the IoU-aware and token labeling losses in Eqs. (4) and (5). Their coefficients are set to be $\lambda_{seg} = 3.0$, $\lambda_{aware} = 2.0$, and $\lambda_{token} = 2.0$, respectively.

⁷ Cross-entropy loss is used with standard transformer architectures, while focal loss [30] is used with deformable transformer architecture.

TABLE 2. Comparison of detection pipelines for all available fully transformer-based object detectors, where \dagger indicates that multi-scale deformable attention is used for neck transformers.

Pipeline Method Name	Body Feature Extractor	Neck Tran. Encoder	Neck Tran. Decoder	Head Prediction
DETR (DeiT)	DeiT Transformer	○	○	Multiple FFNs
DETR (Swin)	Swin Transformer	○	○	Multiple FFNs
Deformable DETR (DeiT)	DeiT Transformer	○ \dagger	○ \dagger	Multiple FFNs
Deformable DETR (Swin)	Swin Transformer	○ \dagger	○ \dagger	Multiple FFNs
YOLOS	DeiT Transformer	×	×	Single FFNs
ViDT (w.o. Neck)	Swin Transformer+RAM	×	×	Single FFNs
ViDT	Swin Transformer+RAM	×	○ \dagger	Multiple FFNs

TABLE 3. Variations of Deformable DETR, YOLOS, and ViDT with respect to their neck structure. They are trained for 50 epochs with the same configuration used in our main experimental results.

Method	Backbone	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L	Param.	FPS
Deformable DETR – neck encoder	Swin-nano	43.1 34.0	61.4 52.8	46.3 35.6	25.9 18.0	45.2 36.3	59.4 48.4	17M 14M	14.6 (17.2) 42.9 (81.4)
YOLOS + neck decoder	DeiT-tiny	30.4 38.1	48.6 57.1	31.1 40.2	12.4 20.1	31.8 40.2	48.2 56.0	6M 13M	52.5 (61.3) 33.6 (52.3)
ViDT + neck encoder	Swin-nano	40.4 46.1	59.6 64.1	43.3 49.7	23.2 28.5	42.5 48.7	55.8 61.7	15M 19M	40.8 (76.0) 26.4 (40.3)

TABLE 4. Comparison of ViDT combined with varying ViT backbones on COCO2017 val set. FPS is measured with batch size 1 of 800×1333 resolution on a single Tesla A100 GPU, where the value inside the parentheses is measured with batch size 4 of the same resolution to maximize GPU utilization.

Backbone	Epochs	AP ^{box}	AP ₅₀ ^{box}	AP ₇₅ ^{box}	AP _S ^{box}	AP _M ^{box}	AP _L ^{box}	Param.	FPS
Swin-nano	50	40.4	59.6	43.3	23.2	42.5	55.8	15M	40.8 (76.0)
Swin-tiny	50	44.8	64.5	48.7	25.9	47.6	62.1	37M	33.5 (51.2)
CoaT-lite-tiny	50	41.0	59.8	44.1	23.7	43.7	56.2	13M	35.0 (72.9)
CoaT-lite-small	50	44.0	63.0	47.1	26.7	46.6	60.6	28M	24.8 (46.4)
PVT-v2-b0	50	39.7	58.3	42.4	22.4	42.0	56.0	11M	39.7 (75.0)
PVT-v2-b2	50	47.7	67.2	51.6	28.5	50.8	64.8	35M	22.4 (35.1)

A.5 Training Configuration

We train ViDT for 50 epochs using AdamW [31] with the same initial learning rate of 10^{-4} for its body, neck and head. The learning rate is decayed by cosine annealing with batch size of 16, weight decay of 1×10^{-4} , and gradient clipping of 0.1. In contrast, ViDT (w.o. Neck) is trained for 150 epochs using AdamW with the initial learning rate of 5×10^{-5} by cosine annealing. The remaining configuration is the same as for ViDT.

Regarding DETR(ViT), we follow the setting of Deformable DETR. Thus, all the variants of this pipeline are trained for 50 epochs with the initial learning rate of 10^{-5} for its pre-trained body (ViT backbone) and 10^{-4} for its neck and head. Their learning rates are decayed at the 40-th epoch by a factor of 0.1. Meanwhile, the results of YOLOS are borrowed from the original paper [6] except YOLOS (DeiT-tiny); since the result of YOLOS (DeiT-tiny) for 800×1333 is not reported in the paper, we train it by following the training configuration suggested by authors.

APPENDIX B

SUPPLEMENTARY EVALUATION

B.1 ViDT Combined with Other ViT Backbones

The proposed RAM allows for any ViT variants to be a standalone object detector by applying their attention mechanisms to $[\text{PATCH}] \times [\text{PATCH}]$ attention. That is, the Swin’s

attention is simply replaced with the other one. The rest $[\text{DET}] \times [\text{DET}]$ and $[\text{DET}] \times [\text{PATCH}]$ attention operations are exactly the same regardless of ViT types. By doing this, we combine ViDT with two other state-of-the-art ViT architectures for object detection, namely CoaT [10], and PVT-v2 [11]. Table 4 summarizes their performance compared with ViDT (Swin) on COCO2017 benchmark data. For a fair comparison, we use the backbones whose model size is in the range of ‘nano’ – ‘small’.

For the models with 10M–15M parameters, all the backbones show similar AP^{box} of 39.7–41.0, but Swin-nano and PVT-v2-b0 are more efficient than CoaT-lite-tiny in terms of FPS. On the other hand, for the models with 28M–35M parameters, Swin-tiny provides the best trade-off between AP and FPS. Although PVT-v2-b2 achieves the highest AP^{box} of 47.7, its FPS is 11.1 lower than Swin-tiny. CoaT-lite-small shows FPS similar to PVT-v2-b2 but its AP is very poor compared with other counterparts. Overall, Swin is the most efficient in terms of FPS, while PVT-v2 is the most effective in terms of the number of parameters.

B.2 Variations of Existing Pipelines

We study more variations of existing detection methods by modifying their original pipelines in Table 2. Thus, we remove the neck encoder of Deformable DETR to increase its efficiency, while adding a neck decoder to YOLOS to lever-

age multi-scale features along with auxiliary decoding loss and iterative box refinement. Note that these modified versions follow exactly the same detection pipeline with ViDT, maintaining a *encoder-free* neck between their body and head. Table 3 summarizes the performance of all the variations in terms of AP, FPS, and the number of parameters.

Deformable DETR shows significant improvement in FPS (+14.4) but its AP drops sharply (−9.1) when its neck encoder is removed. Thus, it is difficult to obtain fine-grained object detection representation directly from the raw ViT backbone without using an additional neck encoder. However, ViDT compensates for the effect of the neck encoder by adding [DET] tokens into the body (backbone), thus successfully removing the computational bottleneck without compromising AP; it maintains 6.4 higher AP compared with the neck encoder-free Deformable DETR (the second row) while achieving similar FPS. This can be attributed to that RAM has a great contribution to the performance w.r.t AP and FPS, especially for the trade-off between them.

YOLOS shows a significant gain in AP (+7.7) while losing FPS (−11.0) when the neck decoder is added. Unlike Deformable DETR, its AP significantly increases even without the neck encoder due to the use of a standalone object detector as its backbone (i.e., the modified DeiT in Figure 2(b)). However, its AP is lower than ViDT by 2.3AP. Even worse, it is not scalable for large models because of its quadratic computational cost for attention. Therefore, in the aspects of accuracy and speed, ViDT maintains its dominance compared with the two carefully tuned baselines.

For a complete analysis, we additionally add a neck encoder to ViDT. The inference speed of ViDT degrades drastically by 13.7 because of the self-attention for multi-scale features at the neck encoder. However, it is interesting to see the improvement of AP by 5.7 while adding only 3M parameters; it is 3.0 higher even than Deformable DETR. This indicates that lowering the computational complexity of the encoder and thus increasing its utilization could be another possible direction for a fully transformer-based object detector.

B.3 [DET] × [PATCH] Attention in RAM

In Section 5.2.1, it turns out that the cross-attention in RAM is only necessary at the last stage of Swin Transformer; all the different selective strategies show similar AP as long as cross-attention is activated at the last stage. Hence, we analyze the attention map obtained by the cross-attention in RAM. Figure 1 shows attention maps for the stages of Swin Transformer where cross-attention is utilized; it contrasts (a) ViDT with cross-attention at all stages and (b) ViDT with cross-attention at the last stage. Regardless of the use of cross-attention at the lower stage, it is noteworthy that the finally obtained attention map at the last stage is almost the same. In particular, the attention map at Stage 1–3 does not properly focus the features on the target object, which is framed by the bounding box. In addition, the attention weights (color intensity) at Stage 1–3 are much lower than those at Stage 4. Since features are extracted from a low level to a high level in a bottom-up manner as they go through the stages, it seems difficult to directly get information about

TABLE 5. AP and FPS comparison with different [DET] × [DET] self-attention strategies with ViDT.

#	Stage Id				Swin-nano	
	1	2	3	4	AP	FPS
(1)	✓	✓	✓	✓	40.4	40.8 (76.0)
(2)		✓	✓	✓	40.3	41.0 (76.4)
(3)			✓	✓	40.4	41.1 (77.0)
(4)				✓	40.1	41.5 (78.1)
(5)					39.7	41.7 (78.5)

the target object with such low-level features at the lower level of stages. Therefore, this analysis provides strong empirical evidence for the use of selective [DET] × [PATH] cross-attention.

B.4 [DET] × [DET] Attention in RAM

Another possible consideration for ViDT is the use of [DET] × [DET] self-attention in RAM. We conduct an ablation study by removing the [DET] × [DET] attention one by one from the bottom stage, and summarize the results in Table 5. When all the [DET] × [DET] self-attention are removed, (5) the AP drops by 0.7, which is a meaningful performance degradation. On the other hand, as long as the self-attention is activated at the last two stages, (1) – (3) all the strategies exhibit similar AP. Therefore, only keeping [DET] × [DET] self-attention at the last two stages can further increase FPS (+0.3) without degradation in AP. This observation could be used as another design choice for the AP and FPS trade-off. Therefore, we believe that [DET] × [DET] self-attention is meaningful to use in RAM.

APPENDIX C

PRELIMINARIES: TRANSFORMERS

A transformer is a deep model that entirely relies on the self-attention mechanism for machine translation [3]. In this section, we briefly revisit the standard form of the transformer.

Single-head Attention. The basic building block of the transformer is a self-attention module, which generates a weighted sum of the values (contents), where the weight assigned to each value is the attention score computed by the scaled dot-product between its query and key. Let W_Q , W_K , and W_V be the learned projection matrices of the attention module, and then the output is generated by

$$\text{Attn}(Z) = \text{softmax}\left(\frac{(ZW_Q)(ZW_K)^\top}{\sqrt{d}}\right)(ZW_V) \in \mathbb{R}^{hw \times d},$$

where $W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$. (5)

Multi-head Attention. It is beneficial to maintain multiple heads such that they repeat the linear projection process k times with different learned projection matrices. Let W_{Q_i} , W_{K_i} , and W_{V_i} be the learned projection matrices of the i -th attention head. Then, the output is generated by the concatenation of the results from all heads,

$$\begin{aligned} \text{Multi-Head}(Z) = \\ [\text{Attn}_1(Z), \text{Attn}_2(Z), \dots, \text{Attn}_k(Z)] \in \mathbb{R}^{hw \times d}, \\ \text{where } \forall_i W_{Q_i}, W_{K_i}, W_{V_i} \in \mathbb{R}^{d \times (d/k)}. \end{aligned} \quad (6)$$

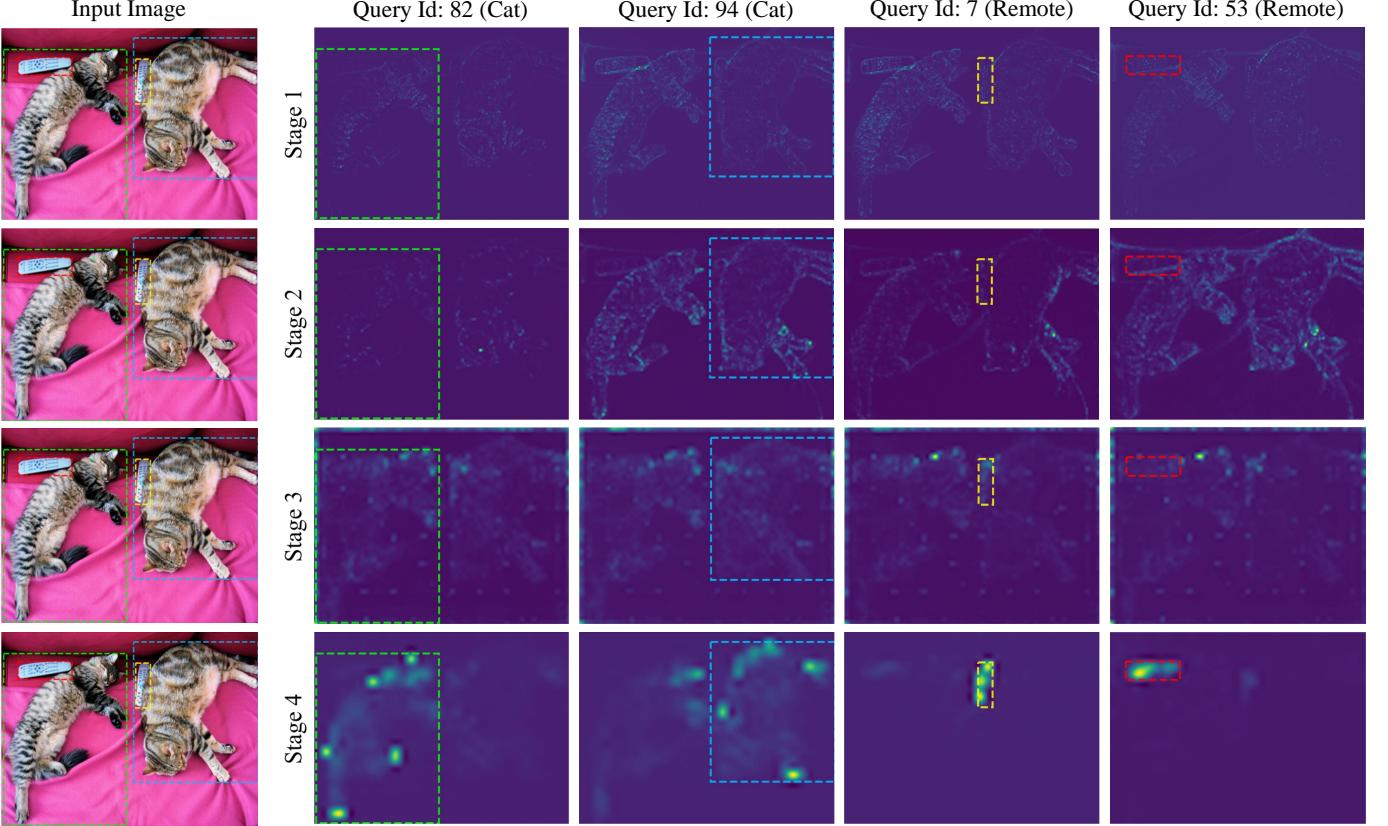
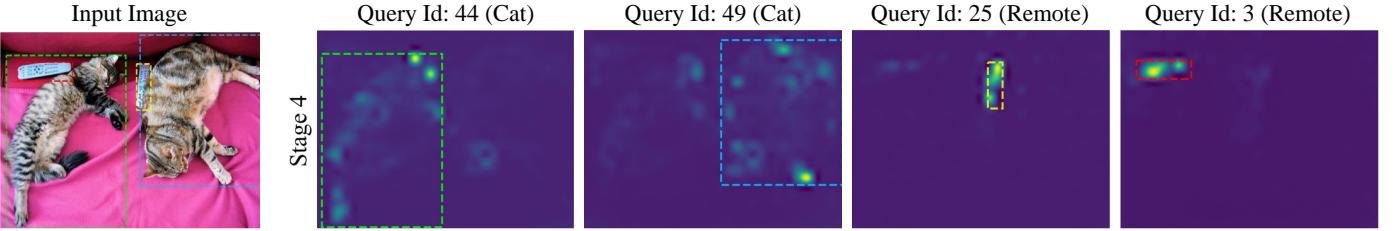
(a) Cross-attention at **all** stages {1, 2, 3, 4}.(b) Cross-attention at the **last** stage {4}.

Fig. 1. Visualization of the attention map for cross-attention with ViDT (Swin-nano).

Typically, the dimension of each head is divided by the total number of heads.

Feed-Forward Networks (FFNs). The output of the multi-head attention is fed to the point-wise FFNs, which performs the linear transformation for each position separately and identically to allow the model focusing on the contents of different representation subspaces. Here, the residual connection and layer normalization are applied before and after the FFNs. The final output is generated by

$$H = \text{LayerNorm}(\text{Dropout}(H') + H''),$$

where $H' = \text{FFN}(H'')$ and (7)

$$H'' = \text{LayerNorm}(\text{Dropout}(\text{Multi-Head}(Z)) + Z).$$

Multi-Layer Transformers. The output of a previous layer is fed directly to the input of the next layer. Regarding the positional encoding, the same value is added to the input of each attention module for all layers.