ANYDEPTH-DETR/-YOLO: ANY-DEPTH OBJECT DETECTION WITH A SINGLE NETWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

Most object detectors are static, fixed-depth networks optimized for a single operating point, requiring separate models for different deployment scenarios. We present an any-depth detection framework that enables a single network to span a continuous range of accuracy–efficiency trade-offs by controlling depth at inference time without retraining. Our key design principle is *stage-wise modularity*: each stage's output remains compatible with downstream stages regardless of depth. To this end, each backbone and neck stage is divided into an essential path, which always executes, and a skippable refinement path, preserving the full multi-scale feature hierarchy at every depth configuration. Training such a network requires jointly optimizing many sub-networks, which introduces conflicting gradient signals. We address this via self-distillation between only the two extremes, with prediction- and feature-level alignment losses that enforce stage-wise modularity. Instantiated on RT-DETR and YOLOv12, our full-depth configurations match or surpass their respective state-of-the-art baselines with negligible parameter overhead, while the most efficient configurations achieve up to $1.60\times$ speedup at a cost of only 2.0 AP, all from a single set of weights.

1 INTRODUCTION

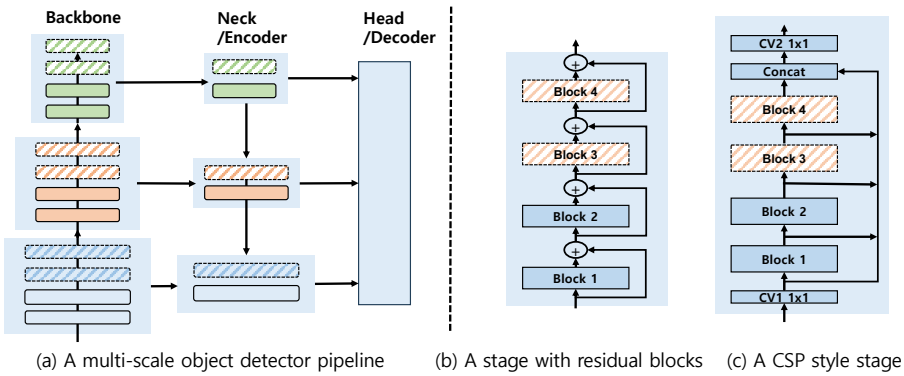
Object detection underpins applications such as autonomous driving, video surveillance, and robotics, yet modern detectors achieve high accuracy at the cost of deep hierarchical architectures that incur substantial computation and latency. To address the accuracy–efficiency trade-off, practitioners train separate models at different operating points, each static and optimized for a fixed resource budget, making it difficult for one network to satisfy heterogeneous deployment requirements.

This rigidity is particularly problematic under varying resource budgets. For example, a surveillance system may need to process many streams in parallel yet switch to high-accuracy mode when a scene of interest is detected, and a cloud service must serve clients with different latency tiers without maintaining a separate model for each. A single static model cannot adapt to such fluctuating demands, motivating detectors that dynamically balance accuracy and efficiency within one network.

Prior adaptive computation methods, including early exiting and width adaptation, reduce overhead by selectively bypassing layers or channels at inference, yielding proven efficiency gains in image classification (Huang et al., 2018; Kang & Lee, 2024; Yu et al., 2022; 2019). These designs, however, transfer poorly to object detection (Appendix A.4 gives an empirical comparison). Unlike classifiers that produce a single global output, detectors rely on a tightly coupled backbone–neck–head pipeline that depends on hierarchical multi-scale features to localize and classify objects at varying scales. Naively skipping stages or pruning channels breaks these dependencies and degrades multi-scale feature quality. To our knowledge, no prior work has achieved depth-controlled computation in object detection while preserving the full multi-scale feature hierarchy that detection requires.

In this work, we present an any-depth architecture and training framework for multi-scale object detection that enables a single model to span a continuous range of accuracy–efficiency trade-offs by controlling network depth. The central design principle is *stage-wise modularity*: at every stage boundary in the backbone and neck, the output of a shallower sub-network and that of a deeper sub-network must be interchangeable, so that downstream stages process either reliably regardless of which depth configuration is active. Realizing this property requires both architectural and training solutions that we develop jointly.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) A multi-scale object detector pipeline (b) A stage with residual blocks (c) A CSP style stage

Figure 1: (a) Overall pipeline of our any-depth object detection network, following the standard backbone–neck–head design common to modern detectors such as RT-DETR and YOLO, operating on multi-scale features. Each stage in both the backbone and neck is divided into an *essential path* (solid blocks) and a skippable *refinement path* (dashed blocks). At inference time, any combination of essential and full paths across stages yields a valid sub-network, enabling a continuous range of accuracy–efficiency trade-offs from a single set of weights. (b) A residual-block stage, as used in RT-DETR. (c) A CSP-style stage, as used in the YOLO family. In both stage types, the essential and full paths are jointly trained via self-distillation to produce aligned feature representations at each stage boundary, so that downstream stages can process the output of either path reliably.

On the architecture side, as shown in Figure 1(a), we propose dividing each stage in both the backbone and neck into an *essential path*, which always executes and learns the core feature representation at the given scale, and a *refinement path*, which is skippable and further refines those features when additional computation is available. Unlike conventional early exiting, which discards entire stages and breaks the backbone–neck–head coupling, this decomposition preserves all hierarchical stages at every depth configuration, structurally enabling stage-wise modularity. Any combination of essential and full paths (essential + refinement) across stages yields a valid sub-network, producing a rich set of depth configurations from a single architecture.

On the training side, naively training all such sub-networks jointly can introduce conflicting gradients for shared weights. We address this by training only the two extremes (the *super-net* with all full paths and the *base-net* with all essential paths) jointly via self-distillation. The super-net acts as an online teacher, and prediction-level and feature-level alignment losses enforce stage-wise modularity by aligning the outputs of the two paths at every stage boundary. Intermediate sub-networks then generalize from the trained endpoints without requiring direct supervision.

We instantiate our approach on RT-DETR (Zhao et al., 2024) and YOLOv12 (Tian et al., 2025), two complementary state-of-the-art real-time detection paradigms: RT-DETR uses a transformer decoder for end-to-end detection, while YOLOv12 follows the YOLO design with an attention-augmented backbone and dense prediction heads. Although the two architectures differ substantially, our *stage-wise modularity* design principle applies to both; architecture-specific design choices are described where they arise. The resulting models, AnyDepth-DETR (AD-DETR) and AnyDepth-YOLO (AD-YOLO), are evaluated on the COCO benchmark. In the full-depth (super-net) configuration, both models match or surpass their respective state-of-the-art baselines. At the most efficient depth configuration, AnyDepth-DETR (R-101) achieves a $1.60\times$ speedup at a cost of only 2.0 AP, and AnyDepth-YOLO (L) achieves a $1.53\times$ speedup with only 1.8 AP drop. Analysis (in Appendix A.3) further shows that the refinement path primarily improves localization precision rather than detection coverage. These results demonstrate that a single modular network can replace multiple fixed-size detectors, paving the way toward flexible real-time object detection.

2 Related Work

Real-Time Object Detectors. Modern real-time detectors rely on hierarchical multi-scale features aggregated by a dedicated neck. FPN (Lin et al., 2017) fuses semantics with spatial signals via a top-down pathway, and PANet (Liu et al., 2018) adds a bottom-up shortcut to improve localization. The

YOLO family builds on these necks with dense prediction heads (Wang et al., 2024a). YOLOv12 (Tian et al., 2025) is the latest, introducing area attention. The DETR family (Carion et al., 2020) replaces the neck with a transformer encoder and eliminates NMS; DAB-DETR (Liu et al., 2022b), DN-DETR (Li et al., 2022), DINO (Zhang et al., 2023), and RT-DETR (Zhao et al., 2024) progressively refine it with dynamic anchor queries, denoising, and a redesigned multi-scale encoder. To cover diverse resource budgets, EfficientDet (Tan et al., 2020), Scaled-YOLOv4 (Wang et al., 2021), and EAutoDet (Wang et al., 2022) each train a family of models via compound scaling, cross-stage partial scaling, and architecture search, respectively. However, every individual model remains static, requiring practitioners to maintain multiple networks.

Adaptive Networks for Object Detection. Adaptive inference is well-studied in classification: MSDNet (Huang et al., 2018) enables anytime prediction via intermediate classifiers, Slimmable Networks (Yu et al., 2019) support multiple channel widths via switchable BN, and AdaptiveDepth (Kang & Lee, 2024) introduces skippable sub-paths for depth modulation at inference. These methods were designed for classification, where a single global representation suffices, making them ill-suited for detection, where backbone, neck, and head are tightly coupled through a multi-scale feature pyramid that must remain intact across all depth configurations. (Appendix A.4 confirms this empirically for Slimmable Networks). For detection, DynamicDet (Lin et al., 2023) uses a learned routing module to decide the inference route per image, and AdaDet (Yang et al., 2024) applies bounding-box uncertainty as a per-input exit criterion. Unlike these input-dependent approaches, our method applies stage-wise conditional computation across both backbone and neck under a resource-driven strategy with no routing overhead at inference time. Work from the real-time computing community has explored scheduling multi-path neural networks under latency constraints across embedded, robotics, and cloud systems (Heo et al., 2020; Liu et al., 2020; 2022a; Kang, 2024). Our models are directly compatible with such schedulers: depth configurations are selectable at runtime, enabling a richer accuracy–efficiency trade-off space.

Knowledge Distillation for Object Detection. Knowledge Distillation (KD) in detection is harder than in classification due to foreground–background imbalance (Chen et al., 2017); early methods therefore use feature imitation over foreground regions (Dai et al., 2021; Cao et al., 2022; Yang et al., 2022; Jia et al., 2024). More recent prediction-level approaches, Localization Distillation (Zheng et al., 2022) and CrossKD (Wang et al., 2024c), show that distilling output distributions can match or exceed feature imitation. KD-DETR (Wang et al., 2024d) further identifies a DETR-specific challenge. Although several works have explored knowledge distillation for object detection, KD for state-of-the-art real-time detectors such as YOLOv12 (Tian et al., 2025) remains largely unexplored. D-FINE (Peng et al., 2025) self-distills across transformer decoder layers of a single fixed model. We instead self-distill across depth configurations of one network, between the super-net and base-net.

3 Approach

3.1 Overview

Multi-Scale Architecture of Object Detectors. As illustrated in Figure 1(a), modern object detectors consist of a backbone, a neck, and a detection head. The backbone extracts multi-scale features at progressively coarser resolutions, the neck fuses them for objects of varying sizes (Lin et al., 2017; Liu et al., 2018), and the detection head performs localization and classification. Both backbone and neck are organized as a series of stages, each operating at a specific scale. This multi-scale structure is fundamental to detection: skipping entire stages removes scale coverage and causes significant accuracy degradation.

Any-Depth Object Detection. To support depth adaptation without sacrificing multi-scale coverage, we divide each backbone and neck stage into an *essential path*, which always executes and learns the core representation at the given scale, and a *refinement path*, which is skippable and further refines those features when additional computation is available. Executing both is the *full path*. The network taking all full paths across stages is the *super-net*, the network taking only essential paths is the *base-net*, and any combination in between yields a valid intermediate sub-network, all sharing a single set of weights. For any such combination to operate correctly, the essential path output x_{ess} and the full path output x_{full} at each stage boundary must be interchangeable, a property we call *stage-wise modularity*. This must be both structurally enabled by the architecture (Section 3.2) and enforced

through training (Section 3.3), after which any sub-network can be freely selected at inference from a single set of weights. Stage-wise modularity is verified quantitatively in Appendix A.3: our any-depth detectors maintain CKA (Kornblith et al., 2019) of 0.92 or above across all stage boundaries, while non-adaptive baselines degrade to 0.09–0.66.

We instantiate our framework on two complementary real-time detectors, RT-DETR (Zhao et al., 2024) and YOLOv12 (Tian et al., 2025), yielding AnyDepth-DETR and AnyDepth-YOLO respectively.

3.2 Architecture

We design the essential and refinement paths for each stage type to structurally enable stage-wise modularity. The two dominant stage designs, or residual blocks in RT-DETR and CSP-style aggregation in YOLO, require different treatments, as described below.

3.2.1 Stages with Residual Blocks

ResNet-style stages (Figure 1(b)) are used in the backbone and neck of RT-DETR. Each block learns a residual function on top of its input, with the i-th block output defined as

$$x^{(i)} = x^{(i-1)} + f^{(i)}(x^{(i-1)}), \quad i = 1, \ldots, S, \tag{1}$$

where $x^{(0)}$ is the stage input and S the total number of blocks. Setting the split point at $m = \lceil S/2 \rceil$, the essential and full path outputs are

$$x_{\text{ess}} = x^{(m)}, \qquad x_{\text{full}} = x^{(S)} = x_{\text{ess}} + \Delta(x_{\text{ess}}), \tag{2}$$

where $\Delta(\cdot)$ is the cumulative residual contribution of the refinement blocks $\{f^{(m+1)}, \ldots, f^{(S)}\}$. Because both paths share the same tensor shape and the refinement is purely additive, no architectural conflict arises. The training objective is to encourage $\Delta(\cdot)$ to learn small refinements that do not shift the feature distribution seen by downstream stages, which is enforced by self-distillation in Section 3.3.

3.2.2 Stages with CSP-Style Aggregation

In the YOLO family (Bochkovskiy et al., 2020; Wang et al., 2023; 2024b; Tian et al., 2025), most stages follow a CSP (Cross Stage Partial) (Wang et al., 2020) style (Figure 1(c)), where block outputs are concatenated channel-wise and reduced by an aggregation layer rather than summed element-wise. This introduces an architectural conflict absent in residual stages: the concatenated tensor width depends on how many blocks were executed, so the aggregation weight cannot be shared between the two paths. A natural remedy is to reuse a column-sliced submatrix of the full aggregator as the essential path weight, but this causes conflicting gradients; the full path trains the shared slice to aggregate S blocks while the essential path trains it to aggregate only m. This conflict is more pronounced when S is large, as more parameters are subject to conflicting gradients. We therefore introduce a dedicated *switchable aggregation layer* used only when the refinement path is skipped, fully decoupling its gradient stream from the full-path aggregator $\mathbf{W}_{\text{cv2}}$. To limit parameter overhead, this dedicated weight is applied only to stages with $S \geq 4$ blocks. The full and essential path outputs are:

$$x_{\text{full}} = \mathbf{W}_{\text{cv2}} \cdot [x^{(0)}, \ldots, x^{(S)}], \qquad x_{\text{ess}} = \mathbf{W}^{\text{ess}}_{\text{cv2}} \cdot [x^{(0)}, \ldots, x^{(m)}], \tag{3}$$

where $\mathbf{W}_{\text{cv2}} \in \mathbb{R}^{c_{\text{out}} \times (S+1)c'}$ and $\mathbf{W}^{\text{ess}}_{\text{cv2}} \in \mathbb{R}^{c_{\text{out}} \times (m+1)c'}$ are $1{\times}1$ convolutions and c' is the hidden dimension per block. Stage-wise modularity between x_{ess} and x_{full} is then enforced by self-distillation in Section 3.3.

3.2.3 Switchable Batch Normalization

Refinement-path blocks see a single input distribution and use a standard BN layer. Essential-path blocks, however, serve a dual role: their output either feeds the refinement path (no-skip mode) or directly serves as the stage output (skip mode). A single shared BN would normalize the essential path identically in both modes, ignoring this dual role. Inspired by switchable batch normalization (Yu et al., 2019), we equip each essential-path block with two independent BN layers, one for *skip* and the other for *no-skip* mode, letting the essential path adjust its output distribution differently in each and promoting compatibility between x_{ess} and x_{full} at every stage boundary.

One important difference from prior switchable BN is that ours need not be N-ary to support N individually trained sub-networks (Yu et al., 2019). Because of our stage-wise modularity property, our switchable BN can instead be binary (skip/no-skip), inserted only in the essential paths, and selected independently at each stage, yielding 2^K combinatorial sub-networks for K stages.

3.3 TRAINING

The goal of training is to optimize a single shared-weight network such that every sub-network, defined by any combination of essential and full paths across stages, achieves detection performance comparable to a dedicated model trained solely at that depth configuration. The key difficulty is that parameter sharing across sub-networks of different depths introduces conflicting gradient signals: optimizing for one depth configuration can degrade others. Our training strategy addresses this through joint supervision of the super-net and base-net with a *self-distillation* objective that reduces potential gradient conflicts while promoting stage-wise modularity.

3.3.1 SELF-DISTILLATION TRAINING OBJECTIVE

In our self-distillation strategy, the super-net acts as an online teacher for the base-net within the same shared-parameter network. For each training iteration, two sequential forward–backward passes are performed on the same mini-batch.

In the first pass, the super-net executes all full paths and produces predictions $P^{\text{super}} = (P^{\text{super}}_{\text{cls}}, P^{\text{super}}_{\text{reg}})$ and intermediate feature maps $\mathcal{F}^{\text{super}}$. The super-net is supervised with the standard ground-truth detection loss:

$$\mathcal{L}^{\text{super}} = \mathcal{L}_{\text{cls}}(P^{\text{super}}_{\text{cls}}, y^{\text{cls}}) + \mathcal{L}_{\text{reg}}(P^{\text{super}}_{\text{reg}}, y^{\text{reg}}), \tag{4}$$

and its gradients are immediately back-propagated. Since the super-net is identical to the base detector, all training hyperparameters follow the respective base model without modification.

In the second pass, the base-net executes all essential paths on the same batch, producing predictions P^{base} and features $\mathcal{F}^{\text{base}}$. The base-net loss combines ground-truth supervision with self-distillation from the super-net:

$$\mathcal{L}^{\text{base}} = \alpha\, \mathcal{L}^{\text{gt}}(P^{\text{base}}, y) + (1-\alpha)\mathcal{L}^{\text{kd}}, \tag{5}$$

where $\alpha \in [0, 1]$ balances ground-truth supervision against self-distillation, and the distillation loss decomposes as:

$$\mathcal{L}^{\text{kd}} = \mathcal{L}^{\text{kd}}_{\text{cls}} + \mathcal{L}^{\text{kd}}_{\text{reg}} + \mathcal{L}^{\text{kd}}_{\text{feat}}. \tag{6}$$

Prediction distillation terms are computed over the valid distillation set for AnyDepth-YOLO (Sec. 3.3.2) and the queries from the Hungarian matcher for AnyDepth-DETR. The super-net predictions are treated as a fixed teacher during the second backward pass.

$\mathcal{L}^{\text{kd}}_{\text{cls}}$ and $\mathcal{L}^{\text{kd}}_{\text{reg}}$ implicitly promote stage-wise modularity by driving the base-net's predictions toward those of the super-net, while $\mathcal{L}^{\text{kd}}_{\text{feat}}$ explicitly enforces it by aligning essential- and full-path feature representations at every stage boundary. Since stage-wise modularity is enforced using only the two extremes, intermediate sub-networks formed by mixing essential and full paths generalize without direct supervision, eliminating the need to train all possible depth configurations explicitly.

Prediction KD. For classification, we apply KL divergence over the softened class probabilities:

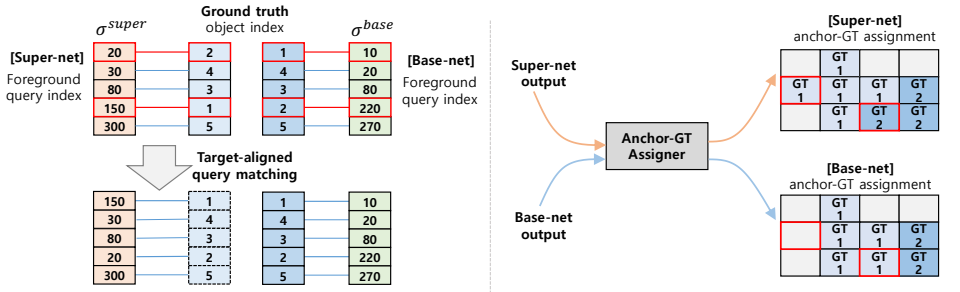
$$\mathcal{L}^{\text{kd}}_{\text{cls}} = \text{KL}(P^{\text{super}}_{\text{cls}} \| P^{\text{base}}_{\text{cls}}). \tag{7}$$

We use KL divergence rather than the super-net's classification loss (BCE for both AnyDepth-YOLO and AnyDepth-DETR) because the super-net's soft logits encode inter-class relationships, the "dark knowledge" (Hinton et al., 2015), that BCE would discard, and it cleanly separates the distillation objective from background suppression, which is handled solely by the ground-truth loss.

For regression, we use the super-net's predicted boxes as soft targets under the same IoU and edge loss (Li et al., 2023) terms as used during base model training:

$$\mathcal{L}^{\text{kd}}_{\text{reg}} = \mathcal{L}^{\text{kd}}_{\text{IoU}}(\hat{y}^{\text{super}}_{\text{box}},\ \hat{y}^{\text{base}}_{\text{box}}) + \mathcal{L}^{\text{kd}}_{\text{edge}}(\hat{y}^{\text{super}}_{\text{box}},\ \hat{y}^{\text{base}}_{\text{box}}). \tag{8}$$

A key challenge in both terms is the target assignment mismatch between the super-net and base-net, which we address in Section 3.3.2.



(a) Target-aligned query matching in AnyDepth-DETR (b) Conflicting Anchor-GT assignments in AnyDepth-YOLO

Figure 2: Conflicting target assignment in any-depth detectors. (a) In AnyDepth-DETR, the Hungarian matcher can assign different queries to the same ground-truth objects across the two execution paths. We reorder the super-net's foreground queries to align with the base-net's target-matched query ordering before applying KD. (b) In AnyDepth-YOLO, the Task-Aligned Assigner (TAL) can assign the super-net and base-net outputs to different anchors for the same ground-truth object. We exclude conflicting anchor assignments (in red boxes) from the foreground distillation set.

Feature Alignment KD. While prediction distillation implicitly encourages stage-wise modularity, we introduce an explicit feature-level alignment loss $\mathcal{L}_{\text{feat}}^{\text{kd}}$ that directly enforces $x_{\text{ess}} \approx x_{\text{full}}$ at each stage boundary, ensuring any downstream stage receives a compatible input regardless of which path the preceding stage took. However, directly matching raw spatial feature maps would over-constrain the full path, penalizing the spatial refinements that the refinement path is designed to learn (see Appendix A.5, Table 9). We therefore apply Global Average Pooling (GAP) channel-wise and align the ℓ_2-normalized descriptors across all supervised stages $\mathcal{I}$:

$$\mathcal{L}_{\text{feat}}^{\text{kd}} = \frac{1}{|\mathcal{I}|} \sum_{s \in \mathcal{I}} \left\| \hat{\mathbf{f}}_s^{\text{ess}} - \hat{\mathbf{f}}_s^{\text{full}} \right\|_2^2, \qquad \hat{\mathbf{f}}_s^{p} = \frac{\text{GAP}(x_{p,s})}{\|\text{GAP}(x_{p,s})\|_2}, \quad p \in \{\text{ess}, \text{full}\}, \tag{9}$$

where $x_{\text{ess},s}$ and $x_{\text{full},s}$ denote the essential- and full-path outputs at stage s, taken from the base-net and super-net passes, respectively. This is equivalent to $2(1 - \cos(\hat{\mathbf{f}}_s^{\text{ess}}, \hat{\mathbf{f}}_s^{\text{full}}))$, encouraging directional alignment rather than value-level imitation. The refinement path thus retains freedom to develop richer spatial representations, while the aligned channel descriptors ensure downstream stages can process the output of either path interchangeably. As we show in Appendix A.3, under this design the refinement path mainly improves localization precision rather than detection coverage.

3.3.2 Target Assignment Conflict in Self-Distillation

A fundamental challenge in self-distillation, where teacher and student share the same weights and are optimized jointly, is that their anchor or query assignments to target objects can disagree. Because the super-net and base-net produce different predictions, the Hungarian matcher in AnyDepth-DETR and TAL (Feng et al., 2021) in AnyDepth-YOLO may assign the same anchor or query to different ground-truth objects. Naively applying the super-net as a teacher on such anchors or queries would inject contradictory supervision: the base-net would be forced to mimic a prediction that its own assigner has already allocated to a different target object.

Target-Aligned Query Matching in AnyDepth-DETR. RT-DETR assigns predictions to ground-truth objects via bipartite matching (Kuhn, 1955; Carion et al., 2020), producing a permutation σ that maps each query to at most one ground-truth object. As shown in Figure 2(a), the super-net and base-net solve this matching independently, yielding permutations σ^{super} and σ^{base} that may assign the same ground-truth object to different query positions. At query position k, the base-net predicts an output for object $\sigma^{\text{base}}(k)$, while the super-net predicts an output for object $\sigma^{\text{super}}(k)$. When $\sigma^{\text{super}}(k) \neq \sigma^{\text{base}}(k)$, element-wise distillation forces the base-net to mimic the teacher's response for a *different* object, introducing a spurious supervisory signal. To avoid this, we re-order the super-net's matched predictions before computing the KD loss so that both networks respond to the same ground-truth object at every position, ensuring semantically consistent per-object guidance.

Table 1: Comparison with SOTA real-time detectors on COCO val2017. FPS is measured at batch size 1 on a single RTX 4090 GPU.

Method	Config	Params (M)	GFLOPs	FPS	AP	AP_{50}	AP_{75}
RT-DETR (Zhao et al., 2024)	R-101	76	257	78.6	54.3	72.7	58.8
	R-50	42	134	113.4	53.1	71.3	57.7
	R-34	31	92	163.9	48.9	66.8	52.6
	R-18	20	60	217.4	46.5	63.8	50.3
AnyDepth-DETR (R-101)	super-net	76(+0.065)	257	78.6	**54.6**	72.4	59.1
	base-net	–	141	125.6	52.6	70.8	57.1
AnyDepth-DETR (R-50)	super-net	42(+0.037)	134	113.4	**53.3**	71.4	57.7
	base-net	–	79	165.3	50.6	68.7	54.8
YOLOv12 (Tian et al., 2025)	X	59.4	185.9	85.5	55.4	72.6	60.3
	L	26.4	83.3	101.5	53.8	71.1	58.6
	M	20.2	60.4	159.2	52.5	70.0	57.1
AnyDepth-YOLO (X)	super-net	61.3(+1.9)	185.9	85.5	**55.5**	71.9	60.2
	base-net	–	132.9	131.8	54.0	70.4	58.6
AnyDepth-YOLO (L)	super-net	27.3(+0.9)	83.3	101.5	**53.9**	70.4	58.6
	base-net	–	59.6	155.3	52.1	68.7	56.6

Conflicting Assignment Removal in AnyDepth-YOLO. Let $\mathcal{A}^{\text{super}}$ and $\mathcal{A}^{\text{base}}$ be the sets of foreground anchor indices assigned by TAL for the super-net and base-net respectively, and let g_j^{super} and g_j^{base} denote the ground-truth index each assigns to anchor j. As shown in Figure 2(b), the two assignments can conflict. We define the *conflict set*:

$$\mathcal{C} = \left\{ j \in \mathcal{A}^{\text{super}} \cap \mathcal{A}^{\text{base}} \;\middle|\; g_j^{\text{super}} \neq g_j^{\text{base}} \right\}, \tag{10}$$

i.e., anchors that are foreground in both sub-networks but point to different objects. KD losses are then restricted to the *valid distillation set*:

$$\mathcal{V} = \left(\mathcal{A}^{\text{super}} \cap \mathcal{A}^{\text{base}}\right) \setminus \mathcal{C}, \tag{11}$$

which contains only anchors that are jointly foreground in both sub-networks and agree on the target object, providing unambiguous and stable regression targets for distillation. The remaining anchors, including background anchors, are supervised solely by the ground-truth loss in Eq. 5.

4 Experiments

4.1 Experimental Setup

All models are trained and evaluated on the COCO (Lin et al., 2014) benchmark (`train2017` / `val2017`). We adopt the original training recipe of each base detector without modification: AnyDepth-YOLO follows YOLOv12 (Tian et al., 2025) and AnyDepth-DETR follows RT-DETR (Zhao et al., 2024) in all settings, including learning rate schedule, data augmentation, batch size, and training epochs. The additional hyperparameters for the self-distillation are detailed in Appendix A.2.

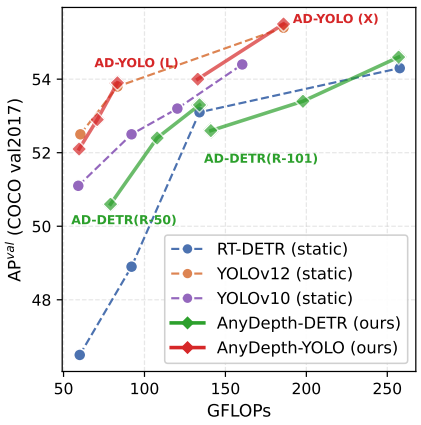


Figure 3: Pareto frontier on COCO.

4.2 Comparison with State-of-the-Art

Table 1 reports AP, GFLOPs, and FPS for each model, and Figure 3 visualizes the accuracy–efficiency Pareto frontier. Our super-net configurations match or surpass their respective baselines with negligible parameter overhead: switchable batch normalization adds less than

0.09% extra parameters, while the switchable aggregator in AnyDepth-YOLO incurs approximately 3% additional parameters. Despite this minimal overhead, the base-net configurations achieve substantially higher throughput at a modest AP cost, and the continuous range of intermediate configurations spans the accuracy–efficiency trade-off covered by multiple individually trained static models.

This flexibility stands in sharp contrast to the static baselines. For RT-DETR, four separately trained models are required to cover a range from 60 GFLOPs / 20M parameters (R-18) to 257 GFLOPs / 76M parameters (R-101). AnyDepth-DETRs (R-50 and R-101), by contrast, support up to 45% reduction in GFLOPs instantly at inference time without any retraining.

4.3 Performance Across Depth Configurations

A key advantage of any-depth detectors is that component depth can be adjusted at inference time without retraining. Table 2 sweeps each depth dimension of AnyDepth-DETR (R-50) to characterize the resulting accuracy–efficiency trade-offs. Decoder early-exiting is inherent to DETR-family models, since queries are progressively refined across layers, making any intermediate layer a valid exit point. Among the skippable components, a shallower encoder trades 0.9 AP for a 6% FPS gain, while routing all backbone stages through the essential path costs 2.0 AP for a 15% FPS gain. Notably, although only the two extremes (super-net and base-net) are trained explicitly, intermediate sub-networks generalize well, yielding smooth AP–efficiency trade-offs across the full depth spectrum, a result of the stage-wise modularity verified quantitatively in Appendix A.3.

Table 2: Component-wise ablation of AnyDepth-DETR (R-50) on COCO `val2017`. Each group varies one component while keeping all others at the super-net setting. FPS measured at batch size 1 on an RTX 4090 GPU. See Table 10 for the corresponding AnyDepth-YOLO (L) results.

Component	Setting	GFLOPs	FPS	AP
Super-net: all full paths, decoder depth 6		134.2	113.4	**53.3**
Backbone (4 stages)	Ess. path in P2	130.6	114.9	52.0
	Ess. paths in P2, P3	123.4	119.6	52.0
	Ess. paths in P2, P3, P4	112.8	127.3	51.4
	Ess. paths in P2, P3, P4, P5	109.2	130.3	51.3
Encoder (FPN + PAN)	Ess. path in PAN	128.9	116.5	53.1
	Ess. paths in FPN + PAN	107.8	120.2	52.4
Decoder (6 layers)	5 layers	132.2	118.9	53.2
	4 layers	130.4	126.9	53.1
	3 layers	128.4	136.8	52.6
	2 layers	126.4	148.0	51.3
	1 layer	124.6	160.0	48.8
Base-net: all ess. paths, decoder depth 4		79.0	165.3	**50.6**

4.4 Ablation

Table 3 ablates each proposed component, starting from a baseline that supervises both the super-net and the base-net with ground-truth only. The baseline already achieves competitive performance, demonstrating that preserving all multi-scale stages at every depth configuration through the essential and refinement path decomposition is itself a strong inductive bias for any-depth object detection.

Switchable BN and Aggregator. Switchable BN improves both models by decoupling the running statistics of the two execution paths. The gain is larger for AD-YOLO (AP_{base}: 50.8 → 51.6) than AD-DETR (50.0 → 50.1), as CSP-style concat stages produce more statistically distinct distributions than the additive residual stages of RT-DETR, making BN decoupling especially important. For AD-YOLO, the dedicated switchable aggregator $\mathbf{W}_{cv2}^{ess}$ further improves AP_{super} from 53.3 to 53.6 and AP_{base} from 51.6 to 52.0, confirming that fully decoupling the gradient streams of the two aggregation weights is essential for CSP-style stages.

Table 3: Ablation of architecture and training components on COCO `val2017`. AP_{super} and AP_{base} denote AP of the super-net and base-net, respectively.

	Component				AD-DETR (R-50)		AD-YOLO (L)	
Setting	Switch. BN	Switch. Aggr.	Target Align.	Feature Align.	AP_{super}	AP_{base}	AP_{super}	AP_{base}
Baseline (Naive Joint-Train)					52.8	50.0	50.2	50.8
+ Switchable BN	✓				53.0	50.1	53.3	51.6
+ Switchable Aggregator	✓	✓			–	–	53.6	52.0
Random 2-Path Sampling	✓	✓			52.7	49.9	53.4	51.7
+ Self-Distillation (Naive)	✓	✓			52.5	50.1	53.7	52.0
+ Target Alignment	✓	✓	✓		52.9	50.3	53.9	52.1
+ Feature Alignment	✓	✓	✓	✓	53.3	50.6	53.9	52.1

Random 2-Path Sampling per Iteration. By default, we train on two extreme endpoints (super-net and base-net). As a control, we instead sample two random sub-networks per iteration and supervise each with ground truth only. This compute-matched variant underperforms training with two fixed endpoints in both AD-DETR and AD-YOLO, showing that our gains do not stem simply from an increased volume of training signal.

Self-Distillation and Target Alignment. Naive self-distillation alone does not improve performance, indicating that uncorrected target assignment conflicts inject noisy gradients when distillation is applied without alignment. Applying target-aligned query reordering (AD-DETR) and conflicting assignment removal (AD-YOLO) resolves this: AD-DETR AP_{super} recovers to 52.9 and AP_{base} improves to 50.3, while AD-YOLO gains $+0.2$ AP and $+0.1$ AP in super-net and base-net, respectively. The smaller gain in AD-YOLO is expected: anchor-target misassignment in dense prediction is rare ($<1.0\%$ of anchors), whereas RT-DETR assigns predictions from only 300 queries, making conflicting matches across sub-networks proportionally more disruptive.

Feature Alignment. Finally, adding $\mathcal{L}_{\text{feat}}^{\text{kd}}$ (Eq. 9) to enforce stage-wise modularity at every stage boundary further improves AD-DETR to AP_{super} 53.3 and AP_{base} 50.6, while AD-YOLO remains unchanged. The difference is explained by the stage architecture. In CSP-style stages, the switchable aggregator $\mathbf{W}_{\text{cv2}}^{\text{ess}}$ already provides structural decoupling between the two paths, making $\mathcal{L}_{\text{feat}}^{\text{kd}}$ redundant: the loss remains negligibly small (≈ 0.005) even without the explicit alignment term, indicating that stage-wise modularity is already achieved by the architecture alone. In residual stages, by contrast, the refinement in skippable blocks is purely additive and no such structural decoupling exists, so the explicit alignment loss is necessary to enforce stage-wise modularity through training.

5 Conclusion

We presented an any-depth object detection framework that enables a single network to span a continuous range of accuracy–efficiency trade-offs by controlling depth at inference time. This is enabled by stage-wise modularity, achieved jointly through architectural decomposition and self-distillation, which keeps each stage's output compatible regardless of the paths taken. Experiments on COCO show that a single model can cover the operating range of multiple static detectors, significantly reducing development and deployment costs.

Limitations. (1) The self-distillation training procedure requires two sequential forward–backward passes per iteration, which effectively doubles the training time compared to standard models. (2) The current framework selects depth configurations based on a resource budget rather than adapting to input difficulty. Integrating a lightweight routing mechanism for input-adaptive depth selection remains an interesting direction for future work.

AI USE STATEMENT

We used generative AI tools for writing assistance, i.e., editing and polishing the manuscript text, and to help write some of the analysis code (e.g., scripts used for the CKA and precision/recall analyses in Appendix A.3). The research idea and all core model and training code were conceived and implemented by the authors without AI assistance. We have reviewed and tested all AI-assisted code and text for correctness, and take responsibility for the final content of this work.

ETHICS STATEMENT

This work evaluates standard detectors (RT-DETR, YOLOv12) on the public COCO benchmark and involves no human subjects, crowdsourcing, or private data; all assets are properly cited and used under their licenses. Our contribution improves detection efficiency rather than introducing new capabilities. As with any efficiency gain in object detection, it could aid beneficial applications (e.g., robotics, autonomous driving) as well as less desirable ones (e.g., cheaper surveillance), but we identify no risks beyond those already associated with general-purpose object detectors.

REPRODUCIBILITY STATEMENT

Our architecture is detailed in Section 3.2, with per-stage configurations in Table 4 (Appendix A.1). The self-distillation objective and target-assignment conflict resolution are described in Section 3.3, with all hyperparameters in Table 5 (Appendix A.2). Training environment and evaluation protocol are given in Appendix A.2. We include full source code and training scripts for both AnyDepth-DETR and AnyDepth-YOLO as supplementary material, sufficient to reproduce all main results; both build on the open-source RT-DETR and YOLOv12 codebases with their original training recipes, evaluated on the public COCO benchmark.

REFERENCES

Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

Weihan Cao, Yifan Zhang, Jianfei Gao, Anda Cheng, Ke Cheng, and Jian Cheng. PKD: General distillation framework for object detectors via Pearson correlation coefficient. In *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=Q9dj3MzY1o7.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I*, pp. 213–229, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-58451-1. doi: 10.1007/978-3-030-58452-8_13. URL https://doi.org/10.1007/978-3-030-58452-8_13.

Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. Learning efficient object detection models with knowledge distillation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 742–751, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Xing Dai, Zeren Jiang, Zhao Wu, Yiping Bao, Zhicheng Wang, Si Liu, and Erjin Zhou. General instance distillation for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7842–7851, June 2021.

Chengjian Feng, Yujie Zhong, Yu Gao, Matthew R Scott, and Weilin Huang. TOOD: Task-aligned one-stage object detection. In *ICCV*, 2021.

Seonyeong Heo, Sungjun Cho, Youngsok Kim, and Hanjun Kim. Real-time object detection system with multi-path neural networks. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 174–187, 2020. doi: 10.1109/RTAS48715.2020.000-8.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations (ICLR)*, 2018.

Zihao Jia, Shengkun Sun, Guangcan Liu, and Bo Liu. MSSD: multi-scale self-distillation for object detection. *Visual Intelligence*, 2(8), March 2024. doi: 10.1007/s44267-024-00040-3. URL https://link.springer.com/article/10.1007/s44267-024-00040-3.

Woochul Kang. QoS-aware inference acceleration using adaptive depth neural networks. *IEEE Access*, 12:49329–49340, 2024. doi: 10.1109/ACCESS.2024.3384233.

Woochul Kang and Hyungseop Lee. Adaptive depth networks with skippable sub-paths. In *Advances in Neural Information Processing Systems*, volume 37, pp. 33213–33231, 2024.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3519–3529. PMLR, 2019.

H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

Feng Li, Hao Zhang, Shilong Liu, Jian Guo, Lionel M. Ni, and Lei Zhang. DN-DETR: Accelerate DETR training by introducing query denoising. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13619–13627, June 2022.

Xiang Li, Chengqi Lv, Wenhai Wang, Gang Li, Lingfeng Yang, and Jian Yang. Generalized focal loss: Towards efficient representation learning for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3139–3153, 2023. doi: 10.1109/TPAMI.2022.3180392.

Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common objects in context, 2014. URL https://arxiv.org/abs/1405.0312.

Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Zhihao Lin, Yongtao Wang, Jinhe Zhang, and Xiaojie Chu. DynamicDet: A unified dynamic architecture for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Shengzhong Liu, Shuochao Yao, Xinzhe Fu, Rohan Tabish, Simon Yu, Ayoosh Bansal, Heechul Yun, Lui Sha, and Tarek Abdelzaher. On removing algorithmic priority inversion from mission-critical machine inference pipelines. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pp. 319–332, 2020. doi: 10.1109/RTSS49844.2020.00037.

Shengzhong Liu, Xinzhe Fu, Maggie Wigness, Philip David, Shuochao Yao, Lui Sha, and Tarek Abdelzaher. Self-cueing real-time attention scheduling in criticality-aware visual machine perception. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 173–186, 2022a. doi: 10.1109/RTAS54340.2022.00022.

Shilong Liu, Feng Li, Hao Zhang, Xiao Yang, Xianbiao Qi, Hang Su, Jun Zhu, and Lei Zhang. DAB-DETR: Dynamic anchor boxes are better queries for DETR. In *International Conference on Learning Representations*, 2022b. URL https://openreview.net/forum?id=oMI9PjOb9Jl.

Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

PaddlePaddle. Knowledge distillation in PaddleClas. https://paddleclas.readthedocs.io/en/latest/advanced_tutorials/distillation/distillation_en.html, 2024. Accessed: 2026-05-04.

Yansong Peng, Hebei Li, Peixi Wu, Yueyi Zhang, Xiaoyan Sun, and Feng Wu. D-FINE: Redefine regression task in DETRs as fine-grained distribution refinement. In *International Conference on Learning Representations (ICLR)*, 2025.

Mingxing Tan, Ruoming Pang, and Quoc V. Le. EfficientDet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Yunjie Tian, Qixiang Ye, and David Doermann. YOLOv12: Attention-Centric Real-Time Object Detectors. In *Advances in Neural Information Processing Systems*, pp. 78433–78457, 2025.

Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. YOLOv10: real-time end-to-end object detection. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, 2024a.

Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. CSPNet: A new backbone that can enhance learning capability of CNN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 390–391, 2020.

Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-YOLOv4: Scaling cross stage partial network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13029–13038, June 2021.

Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7464–7475, 2023.

Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. YOLOv9: Learning what you want to learn using programmable gradient information. *arXiv preprint arXiv:2402.13616*, 2024b.

Jiabao Wang, Yuming Chen, Zhaohui Zheng, Xiang Li, Ming-Ming Cheng, and Qibin Hou. CrossKD: Cross-head knowledge distillation for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16520–16530, June 2024c.

Xiaoxing Wang, Jiale Lin, Juanping Zhao, Xiaokang Yang, and Junchi Yan. EAutoDet: Efficient architecture search for object detection. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XX*, pp. 668–684, Berlin, Heidelberg, 2022. Springer-Verlag. ISBN 978-3-031-20043-4.

Yu Wang, Xin Li, Shengzhao Weng, Gang Zhang, Haixiao Yue, Haocheng Feng, Junyu Han, and Errui Ding. KD-DETR: Knowledge distillation for detection transformer with consistent distillation points sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16016–16025, June 2024d.

I. Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, and Dhruv Mahajan. Billion-scale semi-supervised learning for image classification. *arXiv preprint arXiv:1905.00546*, 2019. URL https://arxiv.org/abs/1905.00546.

Le Yang, Ziwei Zheng, Jian Wang, Shiji Song, Gao Huang, and Fan Li. AdaDet: An adaptive object detection system based on early-exit neural networks. *IEEE Transactions on Cognitive and Developmental Systems*, 16(1):332–345, 2024. doi: 10.1109/TCDS.2023.3274214.

Zhendong Yang, Zhe Li, Xiaohu Jiang, Yuan Gong, Zehuan Yuan, Danpei Zhao, and Chun Yuan. Focal and global knowledge distillation for detectors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4643–4652, 2022.

Fang Yu, Kun Huang, Meng Wang, Yuan Cheng, Wei Chu, and Li Cui. Width & depth pruning for vision transformers. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(3): 3143–3151, Jun. 2022.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel Ni, and Heung-Yeung Shum. DINO: DETR with improved denoising anchor boxes for end-to-end object detection. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=3mRwyG5one.

Yian Zhao, Wenyu Lv, Shangliang Xu, Jinman Wei, Guanzhong Wang, Qingqing Dang, Yi Liu, and Jie Chen. DETRs Beat YOLOs on real-time object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16965–16974, June 2024.

Zhaohui Zheng, Rongguang Ye, Ping Wang, Dongwei Ren, Wangmeng Zuo, Qibin Hou, and Ming-Ming Cheng. Localization distillation for dense object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9407–9416, June 2022.

A Appendix: Detailed Settings, Analysis, and Evaluation Results

A.1 Implementation Details

Architecture Configuration. Table 4 summarizes the per-stage architecture of AnyDepth-DETR (R-50) and AnyDepth-YOLO (L), showing the total number of blocks per stage S, the essential-path split point $m = \lceil S/2 \rceil$, and the number of skippable refinement blocks $S - m$. For AnyDepth-DETR, the backbone follows ResNet-50 with Bottleneck blocks; for AnyDepth-YOLO, the backbone and neck follow YOLOv12-L with C3k2 and A2C2f blocks. In both models, the essential path always executes and the refinement path is skippable at inference time. One exception to the split point formula is encoder stages in AnyDepth-DETR with $S = 3$ blocks, where we use $m = 1$ instead of $m = 2$, as this configuration yields better empirical performance.

Table 4: Per-stage architecture configuration of AnyDepth-DETR (R-50) and AnyDepth-YOLO (L). S: total blocks per stage; m: essential-path blocks; $S{-}m$: skippable blocks. *Stages with $S \geq 4$ use a dedicated switchable aggregation layer $\mathbf{W}_{\text{cv2}}^{\text{ess}}$.

(a) AnyDepth-DETR (R-50)

Component	Stage	Block type	S	m	$S{-}m$
Backbone	P2	Bottleneck	3	2	1
	P3	Bottleneck	4	2	2
	P4	Bottleneck	6	3	3
	P5	Bottleneck	3	2	1
Encoder	P4 (FPN)	RepVggBlock	3	1	2
	P3 (FPN)	RepVggBlock	3	1	2
	P4 (PAN)	RepVggBlock	3	1	2
	P5 (PAN)	RepVggBlock	3	1	2
Decoder	6 layers	Transformer	6	Any layer is a valid exit	

(b) AnyDepth-YOLO (L)

Component	Stage	Block type	S	m	$S{-}m$
Backbone	P2	C3k2	2	1	1
	P3	C3k2	2	1	1
	P4	A2C2f*	4	2	2
	P5	A2C2f*	4	2	2
Neck	P4 (FPN)	A2C2f	2	1	1
	P3 (FPN)	A2C2f	2	1	1
	P4 (PAN)	A2C2f	2	1	1
	P5 (PAN)	C3k2	2	1	1
Detector head	No depth adaptation				

Switchable Aggregation in AnyDepth-YOLO. CSP-style stages with $S \geq 4$ blocks (marked * in Table 4(b)) contain a dedicated switchable aggregation layer $\mathbf{W}_{\text{cv2}}^{\text{ess}}$ (Eq. 3) that is used only when the refinement path is skipped, fully decoupling its gradient stream from the full-path aggregator $\mathbf{W}_{\text{cv2}}$. Stages with $S < 4$ instead reuse the column-sliced submatrix of $\mathbf{W}_{\text{cv2}}$ corresponding to the first $m{+}1$ block outputs, which limits parameter overhead where gradient conflicts are mild.

A.2 Training Details

Training Environment. All models are trained using PyTorch with Distributed Data Parallel (DDP) across 4 RTX 4090 GPUs or 4 RTX 5090 GPUs. All experiments use CUDA 12.8 with automatic mixed precision (AMP) to reduce memory usage and accelerate training. FPS is measured separately on a single NVIDIA RTX 4090 GPU at batch size 1, with 200 warm-up iterations prior to timing.

Table 5: Self-distillation hyperparameters for AnyDepth-DETR (R-50) and AnyDepth-YOLO (L).

Hyperparameter	**AnyDepth-DETR**	**AnyDepth-YOLO**
α (Eq. 5)	0.0	0.2
Classification KD ($\mathcal{L}_{cls}^{kd}$)		
Weight	3.75	0.4
KL divergence temperature	1.0	2.0
Regression KD ($\mathcal{L}_{reg}^{kd}$)		
IoU loss weight ($\mathcal{L}_{\text{IoU}}^{\text{kd}}$)	2.0	1.2
Edge loss weight ($\mathcal{L}_{\text{edge}}^{\text{kd}}$)	5.0	0.8
Edge loss type	L1	DFL
KL divergence temperature for DFL	–	1.0
Feature Alignment KD ($\mathcal{L}_{feat}^{kd}$)		
Backbone stages weight	0.5	0.4
Encoder stages weight	0.2	0.4

Hyperparameters. Both AnyDepth-YOLO and AnyDepth-DETR adopt the original training recipe of their respective base detectors without modification, following YOLOv12 (Tian et al., 2025) and RT-DETR (Zhao et al., 2024) respectively. For self-distillation, the super-net is supervised by ground-truth labels following each base detector's training recipe, while the base-net is jointly supervised by both ground-truth labels and the super-net's predictions (Eq. 5). For AnyDepth-DETR, ResNet backbones are pre-trained before fine-tuning on COCO, following RT-DETR training recipe (PaddlePaddle, 2024). The super-net path of the backbone is supervised by a ResNeXt-101 teacher (Yalniz et al., 2019), and the base-net path is trained to match its representations via self-distillation.

Table 5 lists the hyperparameters specific to self-distillation. One hyperparameter that differs substantially between AnyDepth-YOLO and AnyDepth-DETR is α, the ratio of GT-to-KD supervision for the base-net. In AnyDepth-YOLO, dense prediction over 8400 ($= 80\times80 + 40\times40 + 20\times20$) anchors can cause severe foreground/background class imbalance problems for pure KD. To address this, our self-distillation restricts KD supervision to non-conflicting foreground anchors, leaving the remaining background anchors supervised by GT alone. We therefore use a non-zero α for AnyDepth-YOLO, combining GT and KD supervision during self-distillation. For example, setting $\alpha = 0$ drops super-net and base-net AP by 0.5 and 0.7, respectively, confirming that GT supervision is necessary to compensate for the imbalance. AnyDepth-DETR, in contrast, does not face this problem. Its transformer decoder assigns only 300 queries to target objects and background via Hungarian matching, so there is no dense per-anchor imbalance to correct. Moreover, its auxiliary and denoising outputs are already matched directly against ground truth for both super-net and base-net, regardless of α. The base-net therefore already receives substantial GT supervision through these branches, and adding more via α is redundant with what the architecture already provides. For example, setting $\alpha = 0.2$ for AnyDepth-DETR reduces AP by about 0.3 for both super-net and base-net rather than improving it. We therefore intentionally set $\alpha = 0$ for AnyDepth-DETR.

A.3 Analysis

To better understand how our any-depth detectors achieve their accuracy–efficiency trade-offs, we analyze two complementary aspects of the trained models: how the refinement path contributes to detection quality, and whether stage-wise modularity is empirically achieved across all stage boundaries.

Precision and Recall Analysis. Table 6 breaks down the super-net and base-net results of AnyDepth-DETR (R-50) on COCO `val2017` by IoU threshold and object scale. The overall AP gain from the refinement path is $+2.7$, and the strict-IoU gain ($AP_{0.75}$: $+2.9$) slightly exceeds the loose-IoU gain ($AP_{0.50}$: $+2.7$), indicating that the refinement path sharpens localization *precision* more than detection coverage. Consistent with this, the AR gain is smaller than the AP gain across all

Table 6: Detailed COCO `val2017` evaluation results for AnyDepth-DETR (R-50), comparing the base-net and super-net. Δ denotes the gain of the super-net over the base-net. The AP gain consistently exceeds the AR gain across all scales and IoU thresholds, suggesting that the refinement path primarily improves localization precision rather than detection coverage.

Avg. precision	**Base-net**	**Super-net**	**+Δ**
All (0.50:0.95)	50.6	53.3	+2.7
All (0.50)	68.7	71.4	+2.7
All (0.75)	54.8	57.7	+2.9
AP_{small}	32.6	34.9	+2.3
AP_{medium}	55.0	57.6	+2.6
AP_{large}	66.8	70.4	+3.6

Avg. recall	**Base-net**	**Super-net**	**+Δ**
AR_1	38.1	39.2	+1.1
AR_{10}	64.2	65.9	+1.7
AR_{100}	70.8	72.7	+1.9
AR_{small}	53.9	56.0	+2.1
AR_{medium}	74.9	76.9	+2.0
AR_{large}	87.1	88.3	+1.2

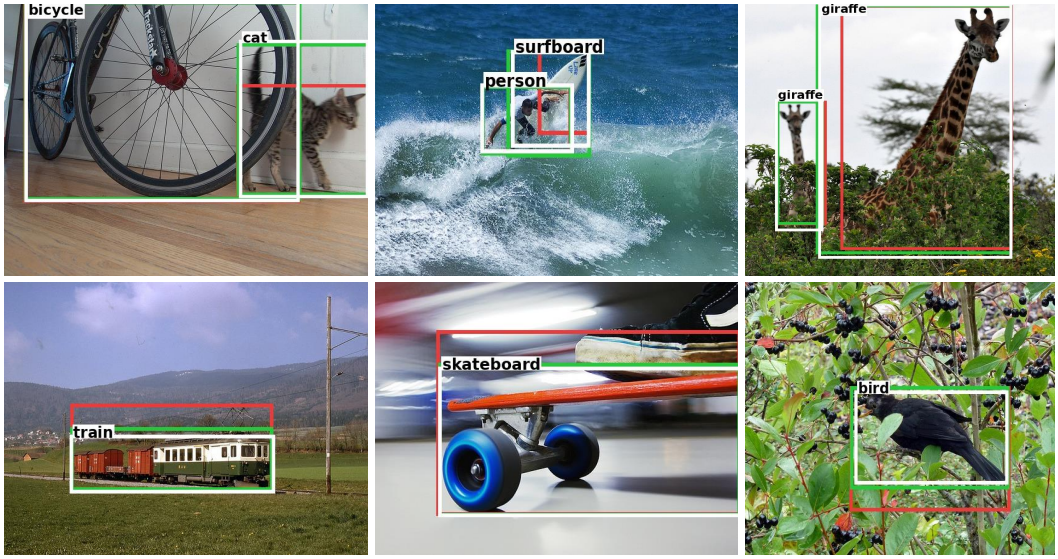


Figure 4: AnyDepth-DETR (R-50) localization examples on COCO `val2017`. White: ground truth, Green: super-net, Red: base-net. Both sub-networks detect the same objects, but the super-net tends to produce tighter bounding boxes on hard images. When scenes contain occlusion or clutter (e.g., cat and surfboard), the skippable refinement path further refines the basic representation from the base-net, recovering finer spatial detail that strict IoU thresholds reward.

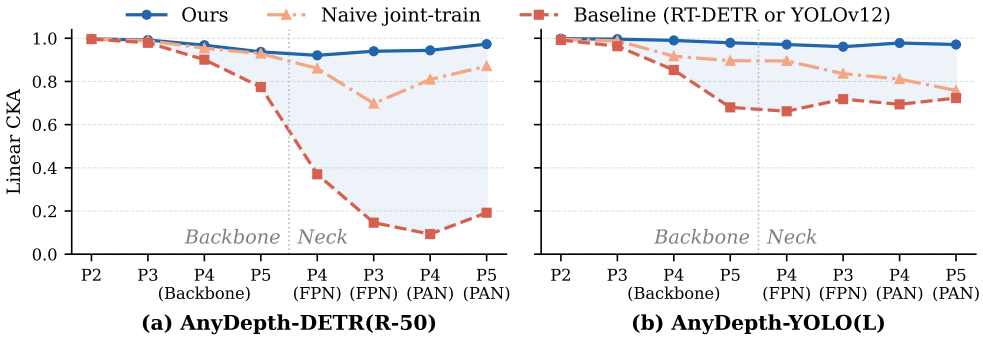
scales (AR_{100}: +1.9 vs. AP: +2.7), suggesting that the base-net already recalls most objects reliably, meaning the semantic recognition signal is largely intact without the refinement path, but localizes them less precisely. The refinement path therefore primarily contributes spatial boundary refinement rather than semantic recognition: it further processes the base-net's representation to recover finer spatial detail that strict IoU thresholds reward.

Figure 4 illustrates this qualitatively: the super-net (green) consistently produces tighter bounding boxes than the base-net (red), particularly for hard images containing occlusion and clutter, while both networks detect the same objects.

Stage-Wise Modularity. To quantitatively verify stage-wise modularity, we measure Centered Kernel Alignment (CKA) (Kornblith et al., 2019) between the essential and full path outputs at each stage boundary on COCO `val2017`. CKA measures representational similarity independently of rotation or scaling (1.0 = identical); high CKA at every stage boundary suggests that downstream stages receive compatible inputs regardless of which path the preceding stage took. As references, we measure (1) the naive joint-train baseline from Table 3 (our architecture, $\alpha = 1$, no switchable components or distillation), and (2) standard RT-DETR and YOLOv12, applying the same split point post-hoc in both cases.

Table 7: Linear CKA between base-net and super-net feature representations at each backbone and neck stage, measured on COCO `val2017`. Each feature map of shape (B, C, H, W) is spatially reduced via 4×4 adaptive average pooling and then flattened to $(B{\cdot}16,\ C)$, treating each pooled spatial cell as an independent sample. Features are accumulated up to 20,000 spatial samples per stage across 500 batches (95% CIs $\leq$0.012, bootstrapped with n=500). *Naive joint*: our architecture without switchable BN/aggregator or self-distillation. *Baseline*: standard models (RT-DETR or YOLOv12).

Comp.	**Stage**	**AnyDepth-DETR (R-50)**			**AnyDepth-YOLO (L)**		
		Ours	Naive joint	RT-DETR	Ours	Naive joint	YOLOv12
Backbone	P2	0.997	0.997	0.996	0.998	0.995	0.992
	P3	0.992	0.989	0.979	0.996	0.988	0.963
	P4	0.968	0.954	0.901	0.990	0.917	0.853
	P5	0.937	0.929	0.774	0.979	0.896	0.680
Neck	P4 (FPN)	0.921	0.860	0.370	0.971	0.895	0.662
	P3 (FPN)	0.940	0.698	0.146	0.961	0.836	0.718
	P4 (PAN)	0.944	0.809	0.093	0.978	0.811	0.694
	P5 (PAN)	0.973	0.871	0.192	0.971	0.758	0.723



(a) AnyDepth-DETR(R-50) **(b) AnyDepth-YOLO(L)**

Figure 5: Linear CKA between essential and full path outputs on COCO `val2017`.

Table 7 and Figure 5 show that our any-depth models maintain consistently high alignment (CKA: $\geq$ 0.92), whereas standard baselines exhibit substantially lower similarity (CKA: 0.09–0.99). The naive joint-train, which incorporates only the architectural decomposition, falls between these two extremes, showing moderate alignment (CKA: 0.70–0.99). This result confirms that stage-wise modularity requires both the architectural decomposition, which provides a structural inductive bias through additive refinement in AnyDepth-DETR and a switchable aggregation layer in AnyDepth-YOLO, and self-distillation, which prevents $\Delta(x_{\text{ess}})$ in Eq. 2 from drifting.

Figure 6 provides qualitative support for stage-wise modularity by visualizing backbone activation heatmaps at stages P4 and P5. The super-net and base-net of AnyDepth-DETR (R-50) produce similar activation patterns at both stages, attending to the same semantic regions of the image, despite the base-net executing only the essential path.

A.4 Comparison with Adaptive-Computation Baselines

A natural question is whether adaptive-computation techniques for classification, such as width-switching, transfer to detection directly or with simple modifications, and how much accuracy weight-sharing costs relative to our stage-wise skip design. Slimmable Networks (Yu et al., 2019) apply width switching to two-stage detectors through the backbone only; no existing baseline applies it to modern one-stage real-time detectors, so we construct this baseline ourselves: *Slimmable-YOLO* adapts Slimmable Networks' width-switching technique to YOLOv12, following its original recipe

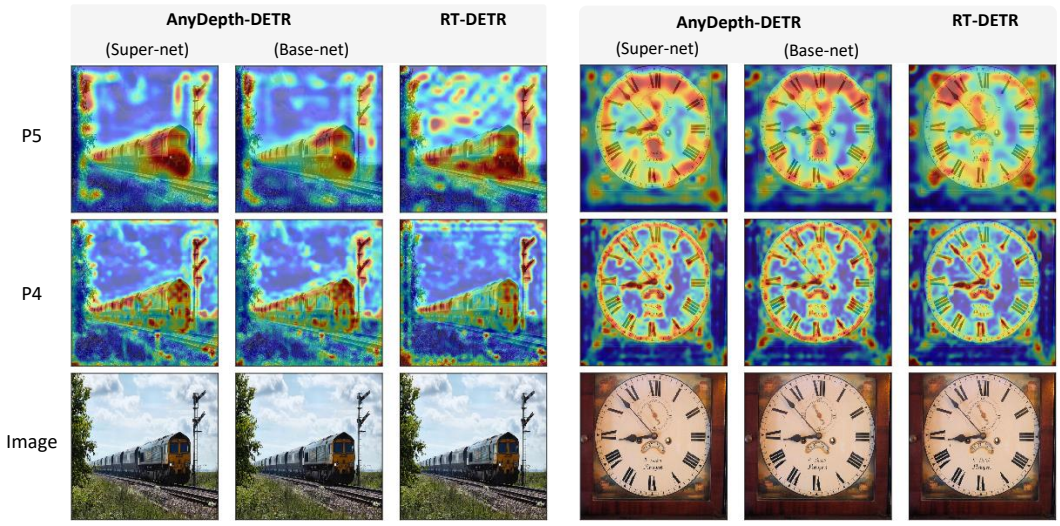


Figure 6: Activation heatmaps at the P4 and P5 backbone stages.

as closely as the architecture allows. We test two variants: (a) a direct port, using switchable batch normalization, sandwich-rule two-pass training, global width-switching, and ground-truth loss only; and (b) the same baseline with a simple modification, naive self-distillation, where the narrow width learns from the wide width's dense per-location predictions with no foreground masking and no conflict exclusion, unlike our method.

Slimmable-YOLO supports two switching modes, wide (83.28 GFLOPs) and narrow (48.07 GFLOPs). Its sandwich-rule training performs the same two forward/backward passes per step as our self-distillation, so per-step training compute closely matches our method. We train both variants for 600 epochs, matching AnyDepth-YOLO's schedule, on identical data and optimizer settings. Table 8 reports the results.

Table 8: Comparison with Slimmable-YOLO, an adaptation of Yu et al. (2019)'s width-switching technique to YOLOv12, trained for 600 epochs on identical data and optimizer settings. (a) is a direct port; (b) adds naive self-distillation. Latency is measured at batch size 1 on a single RTX 5090 GPU, averaged over 100 forward passes after a 20-iteration warmup.

Config	AP	AP50	GFLOPs	ms/img (bs=1)
AnyDepth-YOLO(L) super-net	53.9	70.4	83.27	6.98
Slimmable-YOLO (a) wide	53.1	69.6	83.28	6.97
Slimmable-YOLO (b) wide	53.3	70.0	83.28	6.97
AnyDepth-YOLO(L) base-net	52.1	68.7	59.63	4.50
Slimmable-YOLO (a) narrow	51.5	67.9	48.07	8.41
Slimmable-YOLO (b) narrow	49.7	66.0	48.07	8.41

Accuracy. At matched compute (83.27 vs. 83.28 GFLOPs), our super-net already outperforms both adaptations, by 0.8 AP over Slimmable-YOLO (a) and 0.6 AP over (b). Since both methods use switchable batch normalization, we attribute this gap to the difference between width switching and depth skipping: width switching forces every layer's shared weights to serve both widths, whereas our design keeps the essential path at full width and confines the interaction between the two sub-networks to stage boundaries. At the reduced-compute point, compute is not exactly matched: our base-net uses about 24% more GFLOPs than Slimmable-YOLO's narrow config, since the width-granularity constraint discussed below prevents it from reaching a width closer to our base-net's compute. The resulting gap there (0.6 AP for (a), 2.4 AP for (b)) is therefore expected rather than further evidence of our design's advantage, and we report it for completeness. What is not expected is that this lower compute brings Slimmable-YOLO no speed-up at all, see the latency discussion

below. Separately, (b)'s naive dense distillation hurts rather than helps relative to GT-only training (a): it distills every spatial location uniformly, including background and locations where the wide and narrow sub-networks disagree, rather than only the assignment-consistent foreground positions our conflict-excluded design uses.

Latency. The result is surprising. The adapted baseline provides no real speed benefit at reduced width: Slimmable-YOLO's narrow configuration runs at 0.83 times the speed of wide, meaning it is actually slower, while our base-net runs 1.55 times faster than our super-net. At batch size 1, GPU compute units are underutilized, so latency tracks kernel launches and memory traffic rather than FLOPs. Global width-switching keeps the same number of layers and kernel launches at every width, only shrinking each convolution's size, and it must additionally slice a shared full-width weight tensor at every call, resulting in irregular, non-contiguous memory access. Our method instead skips whole blocks outright, removing kernel launches and memory traffic entirely.

Sub-network granularity. YOLOv12's channel-divisibility constraints permit only 2–4 usable global width points, versus the 2^8 combinatorial sub-networks our stage-wise skip design provides from training just two endpoints. This is a structural, not just accuracy-based, distinction that reflects the stage-wise modularity we claim as a core contribution. Adding more width points is also not free for width-adaptation networks: finer granularity means proportionally more batch-normalization parameters across the whole network, and the sandwich rule must forward and backward through every one of them. As Section 3.2.3 describes, our stage-wise modularity lets each stage boundary use binary (skip/no-skip) switchable BN, rather than the N-ary switching method used in Slimmable Networks (Yu et al., 2019), so this binary choice yields 2^8 sub-networks combinatorially.

A.5 Further Evaluation Results

Feature Alignment Granularity. Section 3.3.1 motivates the GAP-based alignment by arguing that per-location spatial matching would over-constrain the full path. We verify this empirically by comparing three alignment granularities: *pixel-wise* (direct unnormalized feature matching), *spatial* (ℓ_2-normalized per-location matching), and *semantic* (GAP-based channel descriptor alignment, ours). The spatial variant is defined as:

$$\mathcal{L}^{\text{kd}}_{\text{feat,spatial}} = \frac{1}{|\mathcal{I}|}\sum_{s\in\mathcal{I}}\frac{1}{H_sW_s}\sum_{u=1}^{H_sW_s}\left\|\hat{\mathbf{x}}^{\text{ess}}_{s,u}-\hat{\mathbf{x}}^{\text{full}}_{s,u}\right\|_2^2, \qquad \hat{\mathbf{x}}_{s,u}=\frac{\mathbf{x}_{s,u}}{\|\mathbf{x}_{s,u}\|_2}. \tag{12}$$

As shown in Table 9, performance degrades as alignment granularity increases: pixel-wise matching performs worst by imposing value-level constraints that prevent the refinement path from learning any distributional shift, and spatial matching still penalizes spatial refinements by enforcing per-location directional agreement. The GAP-based semantic alignment avoids both issues by aligning only channel-level descriptors, preserving the refinement path's freedom to develop richer spatial representations.

Table 9: Comparison of feature alignment granularity for AnyDepth-DETR (R-50) on COCO `val2017`. Pixel-wise: direct unnormalized feature matching; Spatial: per-location ℓ_2-normalized matching; Semantic (Ours): GAP-based channel descriptor alignment (Eq. 9).

Alignment	AP_{super}	AP_{base}
Pixel-wise	52.7	50.4
Spatial	53.1	50.3
Semantic (Ours)	**53.3**	**50.6**

Performance Across Depth Configurations in AnyDepth-YOLO. Table 10 reports the full component-wise depth sweeps for AnyDepth-YOLO (L). For AnyDepth-YOLO (L), applying essential paths progressively to backbone stages reduces GFLOPs from 83.3 to 70.8 (15%) with only 1.0 AP cost. Applying essential paths to both FPN and PAN neck stages reduces GFLOPs to 72.1 at a 1.1 AP cost, though with a smaller FPS gain than the backbone, reflecting the neck's smaller computational share. Overall, the base-net achieves a $1.53\times$ speedup over the super-net at only 1.8 AP cost, all from a single set of weights.

Table 10: Component-wise ablation of AnyDepth-YOLO (L) on COCO `val2017`.

Component	Setting	GFLOPs	FPS	AP
Super-net: All full paths in backbone and neck		83.3	101.5	**53.9**
Backbone	Ess. path in P2	81.6	112.6	53.7
	Ess. paths in P2, P3	74.7	130.2	53.3
	Ess. paths in P2, P3, P4	72.8	134.2	53.0
	Ess. paths in P2, P3, P4, P5	70.8	134.1	52.9
Neck (FPN+PAN)	Ess. paths in PAN	79.2	103.3	53.6
	Ess. paths in FPN+PAN	72.1	113.9	52.8
Base-net: All essential paths in backbone and neck		59.6	155.3	**52.1**