

RF-DETR: NEURAL ARCHITECTURE SEARCH FOR REAL-TIME DETECTION TRANSFORMERS

Isaac Robinson¹, Peter Robicheaux¹, Matvei Popov¹, Deva Ramanan², Neehar Peri²

¹Roboflow, ²Carnegie Mellon University

ABSTRACT

Open-vocabulary detectors achieve impressive performance on COCO, but often fail to generalize to real-world datasets with out-of-distribution classes not typically found in their pre-training. Rather than simply fine-tuning a heavy-weight vision-language model (VLM) for new domains, we introduce RF-DETR, a light-weight specialist detection transformer that discovers accuracy-latency Pareto curves for any target dataset with weight-sharing neural architecture search (NAS). Our approach fine-tunes a pre-trained base network on a target dataset and evaluates thousands of network configurations with different accuracy-latency tradeoffs *without re-training*. Further, we revisit the “tunable knobs” for NAS to improve the transferability of DETRs to diverse target domains. Notably, RF-DETR significantly improves over prior state-of-the-art real-time methods on COCO and Roboflow100-VL. RF-DETR (nano) achieves 48.0 AP on COCO, beating D-FINE (nano) by 5.3 AP at similar latency, and RF-DETR (2x-large) outperforms GroundingDINO (tiny) by 1.2 AP on Roboflow100-VL while running 20 \times as fast. To the best of our knowledge, RF-DETR (2x-large) is the first real-time detector to surpass 60 AP on COCO. Our code is available on [GitHub](#).

1 INTRODUCTION

Object detection is a fundamental problem in computer vision that has matured in recent years (Felzenszwalb et al., 2009; Lin et al., 2014; Ren et al., 2015). Open-vocabulary detectors like GroundingDINO (Liu et al., 2023) and YOLO-World (Cheng et al., 2024) achieve remarkable zero-shot performance on common categories like `car`, `truck`, and `pedestrian`. However, state-of-the-art vision-language models (VLMs) still struggle to generalize to out-of-distribution classes, tasks and imaging modalities not typically found in their pre-training (Robicheaux et al., 2025). Fine-tuning VLMs on a target dataset significantly improves in-domain performance at the cost of runtime efficiency (due to heavy-weight text encoders) and open-vocabulary generalization. In contrast, specialist (i.e., closed-vocabulary) object detectors like D-FINE (Peng et al., 2024) and RT-DETR (Zhao et al., 2024) achieve real-time inference, but underperform fined-tuned VLMs like GroundingDINO. In this paper, we modernize specialist detectors by combining internet-scale pre-training with real-time architectures to achieve state-of-the-art performance *and* fast inference.

Are Specialist Detectors Over-Optimized for COCO? Sustained progress in object detection can be largely attributed to standardized benchmarks like PASCAL VOC (Everingham et al., 2015) and COCO (Lin et al., 2014). However, we find that recent specialist detectors implicitly overfit to COCO at the cost of real-world performance using bespoke model architectures, learning rate schedulers, and augmentation schedulers. Notably, state-of-the-art object detectors like YOLOv8 (Jocher et al., 2023) generalize poorly to real-world datasets with significantly different data distributions from COCO (e.g., number of objects per image, number of classes, and dataset size). To address these limitations, we present RF-DETR, a scheduler-free approach that leverages internet-scale pre-training to generalize to real-world data distributions. To better specialize our model for diverse hardware platforms and dataset characteristics, we revisit neural architecture search (NAS) in the context of end-to-end object detection and segmentation.

Rethinking Neural Architecture Search (NAS) for DETRs. NAS discovers accuracy-latency tradeoffs by exploring architectural variants within a pre-defined search space. NAS has been previously studied in the context of image classification (Tan & Le, 2019; Cai et al., 2019) and for model

sub-components like detector backbones [Tan et al. \(2020\)](#) and FPNs [Ghiasi et al. \(2019\)](#). Unlike prior work, we explore *end-to-end* weight-sharing NAS for object detection and segmentation. Our key insight, inspired by OFA ([Cai et al., 2019](#)), is that we can vary model inputs like image resolution, and architectural components like patch size during training. Further, weight-sharing NAS allows us to modify inference configurations like the number of decoder layers and query tokens to specialize our strong base model *without fine-tuning*. We evaluate all model configurations with grid search on a validation set. Importantly, our approach does not evaluate the search space until the base model has been fully-trained on the target dataset. As a result, all possible sub-nets (i.e., model configurations within the search space) achieve strong performance without further fine-tuning, significantly reducing the computational cost of optimizing for new hardware. Interestingly, we find that sub-nets not explicitly seen during training still achieve high performance ([Appendix F](#)), suggesting that RF-DETR can generalize to unseen architectures. Extending RF-DETR for segmentation is also relatively straightforward and only requires adding a lightweight instance segmentation head. We denote this model as RF-DETR-Seg. Notably, this allows us to also leverage end-to-end weight-sharing NAS to discover Pareto optimal architectures for real-time instance segmentation.

Standardizing Latency Evaluation. We evaluate our approach on COCO ([Lin et al., 2014](#)) and Roboflow100-VL (RF100-VL) ([Robicheaux et al., 2025](#)) and achieve state-of-the-art performance among real-time detectors. RF-DETR (nano) outperforms D-FINE (nano) by 5% AP on COCO at comparable run-times, and RF-DETR (2x-large) beats GroundingDINO (tiny) on RF100-VL at a fraction of the runtime. RF-DETR-Seg (nano) outperforms YOLOv11-Seg (x-large) on COCO while running $4 \times$ as fast. However, comparing RF-DETR’s latency with prior work remains challenging because reported latency evaluation varies significantly between papers. Notably, each new model re-benchmarks the latency of prior work for fair comparison on their hardware. For example, D-FINE’s reported latency evaluation of LW-DETR ([Chen et al., 2024a](#)) is 25% faster than originally reported. We identify that this lack of reproducibility can be primarily attributed to GPU power throttling during inference. We find that buffering between forward passes limits power over-draw and standardizes latency evaluation ([Table 1](#)).

Contributions. We present three major contributions. First, we introduce RF-DETR, a family of scheduler-free NAS-based detection and segmentation models that outperform prior state-of-the-art on RF100-VL ([Robicheaux et al., 2025](#)) and real-time methods with latencies ≤ 40 ms on COCO ([Lin et al., 2014](#)) ([Figure 1](#)). To the best of our knowledge, RF-DETR is the first real-time detector to exceed 60 mAP on COCO. Next, we explore the “tunable-knobs” for weight-sharing NAS to improve accuracy-latency tradeoffs for end-to-end object detection ([Figure 3](#)). Notably, our use of a weight-sharing NAS allows us to leverage large-scale pre-training and effectively transfer to small datasets ([Table 4](#)). Lastly, we revisit current benchmarking protocols for measuring latency and propose a simple standardized procedure to improve reproducibility.

2 RELATED WORKS

Neural Architecture Search (NAS) automatically identifies families of model architectures with different accuracy-latency tradeoffs ([Zoph & Le, 2016](#); [Zoph et al., 2018](#); [Real et al., 2019](#); [Cai et al., 2018a](#)). Early NAS approaches ([Zoph & Le, 2016](#); [Real et al., 2019](#)) focused primarily on maximizing accuracy, with little consideration for efficiency. As a result, discovered architectures (e.g., NASNet and AmoebaNet) were often computationally expensive. More recent hardware-aware NAS methods ([Cai et al., 2018b](#); [Tan et al., 2019](#); [Wu et al., 2019](#)) address this limitation by incorporating hardware feedback directly into the search process. However, these methods must repeat the search and training process for each new hardware platform. In contrast, OFA ([Cai et al., 2019](#)) proposes a weight-sharing NAS that decouples training and search by simultaneously optimizing thousands of sub-nets with different accuracy-latency tradeoffs. Contemporary methods typically evaluate NAS for object detection by simply replacing standard backbones with NAS backbones in existing detection frameworks. Unlike prior work, we directly optimize end-to-end object detection accuracy to find Pareto optimal accuracy-latency tradeoffs for any target dataset.

Real-Time Object Detectors are of significant interest for safety-critical and interactive applications. Historically, two-stage detectors like Mask-RCNN ([He et al., 2017](#)) and Hybrid Task Cascade ([Chen et al., 2019](#)) achieved state-of-the-art performance at the cost of latency, while single-stage detectors like YOLO ([Redmon et al., 2016](#)) and SSD ([Liu et al., 2016](#)) traded accuracy for state-

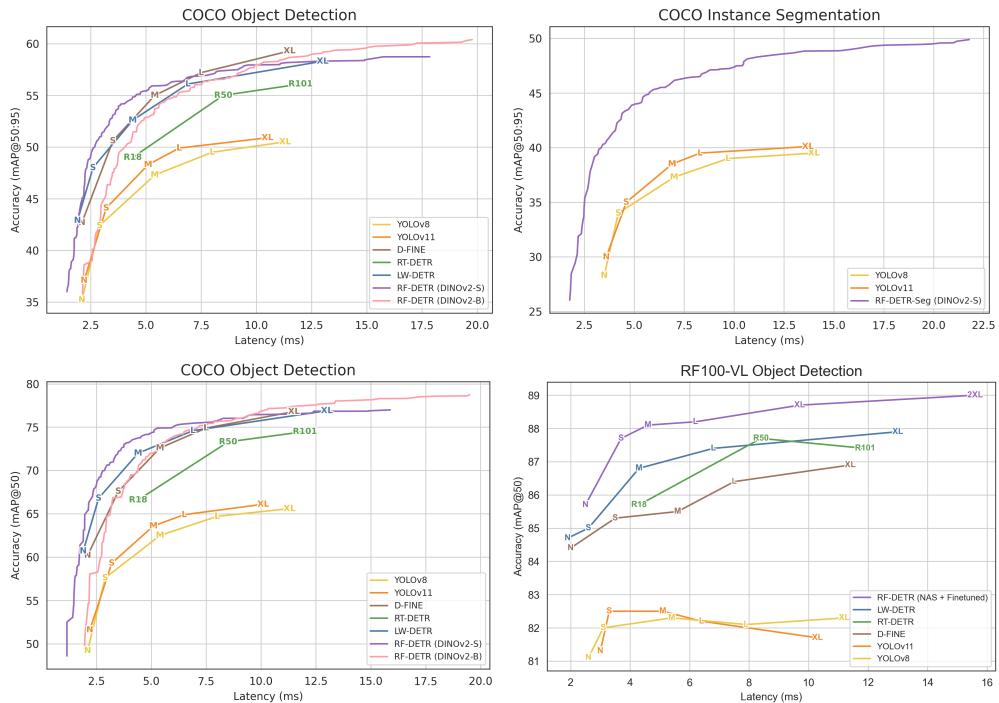


Figure 1: **Accuracy-Latency Pareto Curve.** We plot the Pareto accuracy-latency frontier for real-time detectors on the COCO detection val-set (top left, bottom left), COCO segmentation val-set (top right), and RF100-VL test-set (bottom right). Since RF100-VL contains 100 distinct datasets, we select target latencies for the N, S, M, L, XL, 2XL configurations, search for RF-DETR models with latencies within 10% of the target and report their average performance after fine-tuning to convergence. Importantly, all points along RF-DETR’s continuous Pareto curves for COCO are derived from a single training run.

of-the-art runtime. However, modern detectors (Zhao et al., 2024) reexamine this accuracy-latency tradeoff, simultaneously improving on both axes. Recent YOLO variants innovate on architecture, data augmentation, and training techniques (Redmon et al., 2016; Wang et al., 2023; 2024; Jocher et al., 2023; 2024) to improve performance while maintaining fast inference. Despite their efficiency, most YOLO models rely on non-maximum suppression (NMS), which introduces additional latency. In contrast, DETR (Carion et al., 2020) removes hand-crafted components like NMS and anchor boxes. However, early DETR variants (Zhu et al., 2020; Zhang et al., 2022a; Meng et al., 2021; Liu et al., 2022) achieved strong accuracy at the cost of runtime, limiting their use in real-time applications. Recent works such as RT-DETR (Zhao et al., 2024) and LW-DETR (Chen et al., 2024a) have successfully adapted high performance DETRs for real-time applications. Building on LW-DETR, RF-DETR is the first real-time detector to achieve more than 60 AP on COCO.

Vision-Language Models are trained on large-scale, weakly supervised image-text pairs from the web. Such internet-scale pre-training is a key enabler for open-vocabulary object detection (Liu et al., 2023; Cheng et al., 2024). GLIP (Li et al., 2022) frames detection as phrase grounding with a single text query, while Detic (Zhou et al., 2022) boosts long-tail detection using ImageNet-level supervision (Russakovsky et al., 2015). MQ-Det (Xu et al., 2024) extends GLIP with a learnable module that enables multi-modal prompting. Recent VLMs demonstrate strong zero-shot performance and are often applied as black-box models in diverse downstream tasks (Ma et al., 2023; Peri et al., 2023; Khurana et al., 2024; Osep et al., 2024; Takmaz et al., 2025). However, Robicheaux et al. (2025) find that such models perform poorly when evaluated on categories not typically found in their pre-training, requiring further fine-tuning. In addition, many vision-language models are prohibitively slow, making them difficult to use for real-time tasks. In contrast, RF-DETR combines the fast inference of real-time detectors with the internet-scale priors of VLMs to achieve state-of-the-art performance on RF100-VL and at all latencies ≤ 40 ms on COCO.

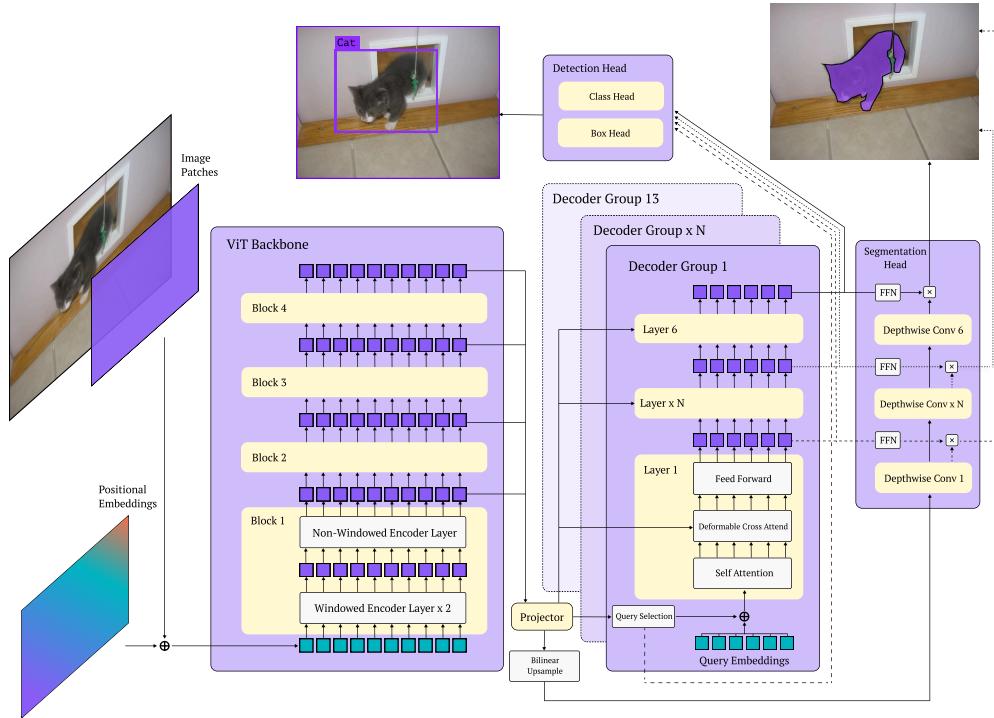


Figure 2: **RF-DETR Architecture.** RF-DETR uses a pre-trained ViT backbone to extract multi-scale features of the input image. We interleave windowed and non-windowed attention blocks to balance accuracy and latency. Notably, the deformable cross-attention layer and segmentation head both bilinearly interpolate the output of the projector, allowing for consistent spatial organization of features. Lastly, we apply detection and segmentation losses at all decoder layers to facilitate decoder drop out at inference.

3 RF-DETR: WEIGHT-SHARING NAS WITH FOUNDATION MODELS

In this section, we describe the architecture of our base model (Figure 2) and present the “tunable knobs” of our weight-sharing NAS (Figure 3). Further, we highlight the limitations of hand-designed learning-rate and augmentation schedulers, and advocate for a scheduler-free approach.

Incorporating Internet-Scale Priors. RF-DETR modernizes LW-DETR (Chen et al., 2024a) by simplifying its architecture and training procedure to improve generalization to diverse target domains. First, we replace LW-DETR’s CAEv2 (Zhang et al., 2022b) backbone with DINOv2 (Oquab et al., 2023). We find that initializing our backbone with DINOv2’s pre-trained weights significantly improves detection accuracy on small datasets. Notably, CAEv2’s encoder has 10 layers with a patch size of 16, while DINOv2’s encoder has 12 layers. Our DINOv2 backbone has more layers and is slower than CAEv2, but we make up for this latency using NAS (discussed next). Lastly, we facilitate training on consumer-grade GPUs via gradient accumulation by using layer norm instead of batch norm in the multi-scale projector.

Real-Time Instance Segmentation. Inspired by Li et al. (2023), we add a lightweight instance segmentation head to jointly predict high quality segmentation masks. Our segmentation head bilinearly interpolates the output of the encoder and learns a lightweight projector to generate a pixel embedding map. Specifically, we upsample the same low-resolution feature map for the detection and segmentation heads to ensure that it contains relevant spatial information. Unlike MaskDINO (Li et al., 2023), we do not incorporate multi-scale backbone features in our segmentation head to minimize latency. Lastly, we compute the dot product of all projected query token embeddings (at the output of each decoder layer transformed by a FFN) with the pixel embedding map to generate segmentation masks. Interestingly, we can interpret these pixel embeddings as segmentation

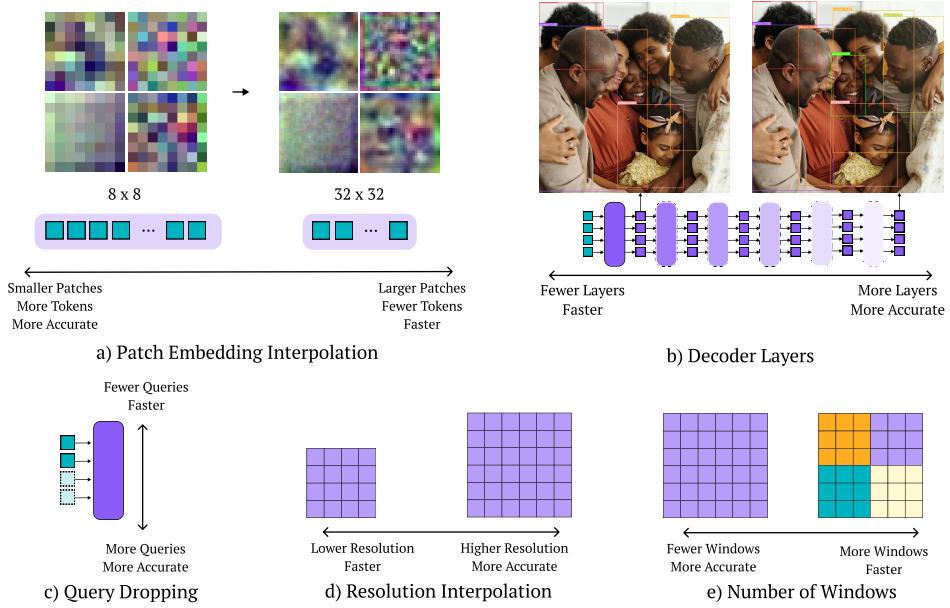


Figure 3: NAS Search Space. We vary (a) patch size, (b) number of decoder layers, (c) number of queries, (d) image resolution, and (e) number of windows per attention block when evaluating different operating points along RF-DETR’s Pareto curve. In addition to training thousands of network configurations in parallel, we find that this “architecture augmentation” serves as a regularizer and improves generalization.

prototypes (Bolya et al., 2019). Motivated by LW-DETR’s observation that pre-training improves DETRs, we pre-train RF-DETR-Seg on Objects-365 (Shao et al., 2019) psuedo-labeled with SAM2 (Ravi et al., 2024) instance masks.

End-to-End Neural Architecture Search. Our weight-sharing NAS evaluates thousands of model configurations with different input image resolutions, patch sizes, window attention blocks, decoder layers, and query tokens. At every training iteration, we uniformly sample a random model configuration and perform a gradient update (Appendix A). This allows our model to efficiently train thousands of sub-nets in parallel, similar to ensemble learning with dropout (Srivastava et al., 2014). We find that this weight-sharing NAS approach also serves as a regularizer during training, effectively performing “architecture augmentation”. To the best of our knowledge, RF-DETR is the first end-to-end weight-sharing NAS applied to object detection and segmentation. We describe each component below.

- *Patch Size.* Smaller patches lead to higher accuracy at greater computational cost. We adopt a FlexiVIT-style (Beyer et al., 2023) transformation to interpolate between patch sizes during training.
- *Number of Decoder Layers.* Similar to recent DETRs (Peng et al., 2024; Zhao et al., 2024), we apply a regression loss to the output of all decoder layers during training. Therefore, we can drop any (or all) decoder blocks during inference. Interestingly, removing the entire decoder during inference effectively turns RF-DETR into a single-stage detector. Notably, truncating the decoder also shrinks the size of the segmentation branch, allowing for greater control over segmentation latency.
- *Number of Query Tokens.* Query tokens learn spatial priors for bounding box regression and segmentation. We drop query tokens (ordered by the maximum sigmoid of the corresponding class logit per token at the output of the encoder, see Appendix B) at test time to vary the maximum number of detections and reduce inference latency. The Pareto optimal number of query tokens implicitly encodes dataset statistics about the average number of objects per image in a target dataset.

- *Image Resolution.* Higher resolution improves small object detection performance, while lower resolution improves runtime. We pre-allocate N positional embeddings corresponding to the largest image resolution divided by the smallest patch size and interpolate these embeddings for smaller resolutions or larger patch sizes.
- *Number of Windows per Windowed Attention Block.* Window attention restricts self-attention to only process a fixed number of neighboring tokens. We can add or remove windows per block to balance accuracy, global information mixing, and computational efficiency.

At inference time, we pick a specific model configuration to select an operating point on the accuracy-latency Pareto curve. Importantly, different model configurations may have similar parameter counts but significantly different latencies. Similar to [Cai et al. \(2019\)](#), we see little benefit from fine-tuning the NAS-mined models on COCO (Appendix G), but note modest improvements from fine-tuning NAS-mined models on RF100-VL. This additional fine-tuning is optional, and is often unnecessary for practical deployment. We posit that RF-DETR benefits from additional fine-tuning on RF100-VL because the “architecture augmentation” regularization requires more than 100 epochs to converge on small datasets. Notably, prior weight-sharing NAS methods ([Cai et al., 2019](#)) train in stages and use a different learning-rate scheduler per-stage. However, such schedulers make strict assumptions about model convergence, which may not hold across diverse datasets.

Training Schedulers and Augmentations Bias Model Performance. State-of-the-art detectors often require careful hyper-parameter tuning to maximize performance on standard benchmarks. However, such bespoke training procedures implicitly bias the model towards certain dataset characteristics (e.g. number of images). Concurrent with DINOv3 ([Siméoni et al., 2025](#)), we observe that cosine schedules assume a known (fixed) optimization horizon, which is impractical for diverse target datasets like those in RF100-VL. Data augmentations introduce similar biases by presuming prior knowledge of dataset properties. For example, prior work leverages aggressive data augmentation (e.g., VerticalFlip, RandomFlip, RandomResize, RandomCrop, YOLOHSVRandomAug, and CachedMixUp) to increase effective dataset size. However, certain augmentations like VerticalFlip may negatively bias model predictions in safety-critical domains. For example, a person detector in a self-driving vehicle should not be trained with VerticalFlip to avoid false positive detections from reflections in puddles. Therefore, we limit augmentations to horizontal flips and random crops. Lastly, LW-DETR applies a per-image random resize augmentation, where each image is padded to match the largest image in the batch. As a result, most images have significant padding, which introduces window artifacts, and wastes computation on padded regions. In contrast, we resize images at the batch level to minimize the number of padded pixels per-batch and to ensure that all positional encoding resolutions are equally likely to be seen at train time.

4 EXPERIMENTS

We evaluate RF-DETR on COCO and RF100-VL and demonstrate that our approach achieves state-of-the-art accuracy among all real-time methods. In addition, we identify inconsistencies in standard benchmarking protocols and present a simple standardized procedure to improve reproducibility. Following LW-DETR ([Chen et al., 2024a](#)), we group models of similar latency into the same size bucket rather than grouping based on parameter count.

Datasets and Metrics. We evaluate RF-DETR on COCO for fair comparison with prior work and on RF100-VL to evaluate generalization to real-world datasets with significantly different data distributions. Due to the diversity of RF100-VL’s 100 datasets, we posit that overall performance on this benchmark is a proxy for transferability to any target domain. We use pycocotools to report standard metrics like mean average precision (mAP) and provide breakdown analysis for AP_{50} , AP_{75} , AP_{Small} , AP_{Medium} , and AP_{Large} . Further, we evaluate efficiency by measuring GFLOPs, number of parameters, and inference latency on an NVIDIA T4 GPU with Tensor-RT 10.4 and CUDA 12.4.

Standardizing Latency Benchmarking. Despite its maturity, benchmarking object detectors remains inconsistent across prior work. For example, YOLO-based models often omit non-maximal suppression (NMS) when computing latency, leading to unfair comparisons with end-to-end detec-

Table 1: **Standardizing Latency Evaluation.** Variance in latency measurements can be largely attributed to power throttling and GPU overheating. We mitigate this issue by buffering for 200ms between forward passes. Notably, this benchmarking approach is not designed to measure sustained throughput, but rather ensures reproducible latency measurements. We are unable to reproduce YOLOv8 and YOLOv11’s mAP results in TensorRT, likely because these models evaluate with multi-class NMS but only use single-class NMS in inference. We use the standard NMS-tuned confidence threshold of 0.01. YOLOv8 and YOLOv11 performance degrades further when quantized from FP32 to FP16, reaffirming that all models should report latency and accuracy using the same model artifact. Notably, naively quantizing D-FINE to FP16 reduces performance to 0.5 AP. We fix this issue by changing the authors’ export code to use ONNX opset 17 (Appendix A).

Method	Reported		Buffering (FP-32)		Buffering (FP-16)	
	AP _{50:95}	Latency (ms)	AP _{50:95}	Latency (ms)	AP _{50:95}	Latency (ms)
YOLOv8 (M)	50.2	5.86	49.3	14.8	47.3	5.4
YOLOv11 (M)	51.5	4.7	49.7	18.7	48.3	5.2
RT-DETR (R18)	49.0	4.61	49.0	12.2	49.0	4.4
LW-DETR (M)	52.5	5.6	52.6	26.8	52.6	4.4
D-FINE (M)	55.1	5.62	55.1	13.9	55.0 (0.5*)	5.4
RF-DETR (M)	-	-	54.8	20.5	54.7	4.4

Table 2: **COCO Detection Evaluation.** We compare RF-DETR with popular real-time and open-vocabulary object detectors below. We find that RF-DETR (nano) outperforms D-FINE (nano) and LW-DETR (tiny) by more than 5 AP. RF-DETR significantly outperforms YOLOv8 and YOLOv11, while RF-DETR’s nano size achieves performance parity with YOLOv8 and YOLOv11’s medium size model. We denote models that do not support TensorRT execution with a star, and instead report PyTorch latency results. See Appendix E for L, XL, and Max variants of RF-DETR on COCO.

Model	Size	# Params.	GFLOPS	Latency (ms)	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
Real-Time Object Detection w/ NMS										
YOLOv8 (Jocher et al., 2023)	N	3.2M	8.7	2.1	35.2	49.2	38.3	15.8	38.8	51.3
YOLOv11 (Jocher et al., 2024)	N	2.6M	6.5	2.2	37.1	51.6	40.4	17.3	40.7	55.6
YOLOv8 (Jocher et al., 2023)	S	11.2M	28.6	2.9	42.4	57.6	46.0	22.2	47.1	59.6
YOLOv11 (Jocher et al., 2024)	S	9.4M	21.5	3.2	44.1	59.3	47.9	26.1	48.5	62.6
YOLOv8 (Jocher et al., 2023)	M	25.9M	78.9	5.4	47.3	62.5	51.5	27.5	52.9	65.1
YOLOv11 (Jocher et al., 2024)	M	20.1M	68.0	5.1	48.3	63.6	52.5	29.1	53.8	66.3
Open-Vocabulary Object Detection (Fully-Supervised Fine-Tuning)										
GroundingDINO (Liu et al., 2023)	T	173.0M	1008.3	427.6*	58.2	-	-	-	-	-
End-to-End Real-Time Object Detection										
LW-DETR (Chen et al., 2024a)	T	12.1M	21.4	1.9	42.9	60.7	45.9	22.7	47.3	60.0
D-FINE (Peng et al., 2024)	N	3.8M	7.3	2.1	42.7	60.2	45.4	22.9	46.6	62.1
RF-DETR (Ours)	N	30.5M	31.9	2.3	48.0	67.0	51.4	25.2	53.5	70.0
LW-DETR (Chen et al., 2024a)	S	14.6M	31.8	2.6	48.0	66.8	51.6	26.7	52.5	65.6
D-FINE (Peng et al., 2024)	S	10.2M	25.2	3.5	50.6	67.6	55.0	32.6	54.6	66.6
RF-DETR (Ours)	S	32.1M	59.8	3.5	52.9	71.9	57.0	32.0	58.3	73.0
RT-DETR (Zhao et al., 2024)	R18	36.0M	100.0	4.4	49.0	66.6	53.3	32.8	52.1	65.0
LW-DETR (Chen et al., 2024a)	M	28.2M	83.9	4.4	52.6	72.0	56.6	32.5	57.6	70.5
D-FINE (Peng et al., 2024)	M	19.2M	56.6	5.4	55.0	72.6	59.7	37.6	59.4	71.7
RF-DETR (Ours)	M	33.7M	78.8	4.4	54.7	73.5	59.2	36.1	59.7	73.8
RF-DETR (Ours)	2XL	126.9M	438.4	17.2	60.1	78.5	65.5	43.2	64.9	76.2

tors. Additionally, YOLO-based segmentation models measure the latency of generating prototype predictions instead of directly usable per-object masks (Jocher et al., 2024), leading to biased runtime measurements. Further, D-FINE’s reported latency evaluation of LW-DETR is 25% faster than reported by Chen et al. (2024b). We observe that such differences can be attributed to detectable power throttling events, particularly when the GPU overheats (Table 1). In contrast, simply pausing for 200ms between consecutive forward passes largely mitigates power throttling, yielding more stable latency measurements (Appendix K). Lastly, we find that prior work often reports latency using FP16 quantized models, but evaluates accuracy with FP32 models. However, naive quantization can significantly degrade performance (in some cases dropping performance to near 0 AP). To ensure fair comparison, we advocate for reporting accuracy and latency with the same model artifact. We release our stand-alone benchmarking tool on [GitHub](https://github.com/RF-DETR/RF-DETR).

Evaluating RF-DETR and RF-DETR-Seg on COCO. COCO (Lin et al., 2014) is a flagship benchmark for object detection and instance segmentation. In Table 2, we compare RF-DETR with leading real-time and open-vocabulary detectors. RF-DETR (nano) beats both D-FINE (nano) and LW-DETR (nano) by more than 5 AP. We see similar trends for small and medium sizes as well. No-

Table 3: **COCO Instance Segmentation Evaluation.** We compare RF-DETR with popular real-time instance segmentation methods on COCO. Notably, RF-DETR (nano) outperforms all reported YOLOv8 and YOLOv11 model sizes. Further RF-DETR (nano) outperforms FastInst by 5.4%, while running nearly ten times faster. RF-DETR (medium) approaches the performance on MaskDINO at a fraction of the runtime. We denote models that do not support TensorRT execution with a star, and instead report PyTorch latency results. Our latencies for YOLOs also include the conversion of protos into masks, which are not typically included in prior benchmarks but nonetheless contribute meaningfully to practical latency. See Appendix E for L, XL, and Max variants of RF-DETR-Seg on COCO.

Model	Size	# Params.	GFLOPS	Latency (ms)	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
Real-Time Instance Segmentation w/ NMS										
YOLOv8 (Jocher et al., 2023)	N	3.4M	12.6	3.5	28.3	45.6	29.8	9.3	31.3	44.3
YOLOv11 (Jocher et al., 2024)	N	2.9M	10.4	3.6	30.0	47.8	31.5	10.0	33.4	47.7
YOLOv8 (Jocher et al., 2023)	S	11.8M	42.6	4.2	34.0	53.8	36.0	13.6	38.5	52.2
YOLOv11 (Jocher et al., 2024)	S	10.1M	35.5	4.6	35.0	55.4	37.1	15.3	39.7	53.9
YOLOv8 (Jocher et al., 2023)	M	27.3M	110.2	7.0	37.3	58.2	39.9	16.7	43.0	56.1
YOLOv11 (Jocher et al., 2024)	M	22.4M	123.3	6.9	38.5	60.0	40.9	18.0	44.3	57.6
End-to-End Instance Segmentation										
RF-DETR-Seg. (Ours)	N	33.6M	50.0	3.4	40.3	63.0	42.6	16.3	45.3	63.6
RF-DETR-Seg. (Ours)	S	33.7M	70.6	4.4	43.1	66.2	45.9	21.9	48.5	64.1
FastInst (He et al., 2023)	R50	29.7M	99.7	39.6*	34.9	56.0	36.2	13.3	38.0	56.8
MaskDINO (Li et al., 2023)	R50	52.1M	586	242*	46.3	69.0	50.7	26.1	49.3	66.1
RF-DETR-Seg. (Ours)	M	35.7M	102.0	5.9	45.3	68.4	48.8	25.5	50.4	65.3
RF-DETR (Ours)	2XL	38.6M	435.3	21.8	49.9	73.1	54.5	33.9	54.1	65.7

table, RF-DETR also significantly outperforms YOLOv8 and YOLOv11. RF-DETR (nano) matches the performance of YOLOv8 and YOLOv11 (medium). We use mm detection’s implementation of GroundingDINO and include their reported AP since they do not release a model artifact for GroundingDINO fine-tuned on COCO. We benchmark mmGroundingDINO’s parameter count, GFLOPS, and latency using the released open-vocabulary model. In Table 3, we compare RF-DETR-Seg with real-time instance segmentation models. RF-DETR-Seg (nano) outperforms YOLOv8 and YOLOv11 at all sizes. Furthermore, RF-DETR-Seg (nano) beats FastInst by 5.4% while running almost ten times faster. Similarly, RF-DETR (x-large) surpasses GroundingDINO (tiny), and RF-DETR-Seg (large) outperforms MaskDINO (R50), at a fraction of their runtime.

Evaluating RF-DETR on RF100-VL. RF100-VL is a challenging detection benchmark composed of 100 diverse datasets. We report latencies, FLOPs, and accuracy averaged over all 100 datasets in Table 4. Our results show that RF-DETR (2x-large) outperforms GroundingDINO and LLMDet while requiring only a fraction of their runtime. Interestingly, RT-DETR outperforms D-FINE (which is built on RT-DETR) at AP₅₀, indicating that D-FINE’s hyperparameters are potentially overfit to COCO. We also note that RF-DETR benefits from scaling to larger backbone sizes (Appendix E). In contrast, YOLOv8 and YOLOv11 consistently underperform DETR-based detectors, and scaling these model families to larger sizes does not improve their performance on RF100-VL (Figure 1).

Impact of Neural Architecture Search. We ablate the impact of weight-sharing NAS in Table 3. We find that adopting a gentler set of hyperparameters compared to LW-DETR (e.g. larger batch size, lower learning rate, and replacing batch normalization with layer normalization) reduces performance over LW-DETR by 1.0%. Notably, replacing batch normalization with layer normalization hurts performance, but is necessary to train on consumer hardware. However, replacing LW-DETR’s CAEv2 backbone with DINoV2 improves performance by 2%. The lower learning rate, in particular, helps preserve DINoV2’s pre-trained knowledge, while additional epochs of Objects-365 pre-training further compensates for the slower optimization. Our final model with weight-sharing NAS improves over LW-DETR by 2% without increasing latency.

Impact of Backbone Architecture and Pre-Training. We study the impact of different backbone architectures in RF-DETR. We find that DINoV2 achieves the best performance, outperforming CAEv2 by 2%. Interestingly, despite having fewer parameters than SigLIPV2, SAM2’s Hiera-S backbone is considerably slower. This is in contrast with Hiera’s claim that it is meaningfully faster than equivalently performant ViTs. However, Hiera does not explore latency in the context of Flash Attention kernels, which are highly optimized in compilers such as TensorRT. Additionally, existing

Table 4: RF100-VL Evaluation. We compare RF-DETR with real-time and open-vocabulary object detectors on RF100-VL. Interestingly, RF-DETR (2x-large) outperforms GroundingDINO (tiny), and LLMDet (tiny) at a fraction of their runtime. We report the average latency and FLOPs over all 100 datasets. We note that YOLOv8 and YOLOv11’s latency measurements may be suboptimal because the default tuned NMS threshold of 0.01 may not work well for all datasets in RF100-VL. We denote models that do not support TensorRT execution with a star, and instead report PyTorch latency results. See Appendix E for L, XL, and Max variants of RF-DETR on RF100-VL.

Model	Size	# Params.	GFLOPS	Latency (ms)	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
Real-Time Object Detectors w/ NMS										
YOLOv8 (Jocher et al., 2023)	N	3.2M	8.7	2.6	55.0	81.1	59.5	4.8	44.1	48.0
YOLOv11 (Jocher et al., 2024)	N	2.6M	6.5	3.0	55.5	81.3	60.3	4.7	44.4	49.2
YOLOv8 (Jocher et al., 2023)	S	11.2M	28.6	3.1	56.3	82.0	60.9	6.1	45.6	48.6
YOLOv11 (Jocher et al., 2024)	S	9.4M	21.5	3.3	56.4	82.5	61.3	6.5	45.5	48.5
YOLOv8 (Jocher et al., 2023)	M	25.9M	78.9	5.4	56.5	82.3	60.9	6.4	45.7	48.6
YOLOv11 (Jocher et al., 2024)	M	20.1M	68.0	5.1	57.0	82.5	61.9	7.3	46.1	48.6
Open-Vocabulary Object-Detectors (Fully-Supervised Fine-Tuning)										
GroundingDINO (Liu et al., 2023)	T	173.0M	1008.3	309.9*	62.3	88.8	67.8	39.2	57.7	69.5
LLMDet (Fu et al., 2025)	T	173.0M	1008.3	308.4*	62.3	88.3	67.8	39.1	57.6	70.3
End-to-End Real-Time Object Detectors										
LW-DETR (Chen et al., 2024a)	N	12.1M	21.4	1.9	57.1	84.7	61.5	31.2	51.8	65.8
D-FINE (Peng et al., 2024)	N	3.8M	7.3	2.0	58.2	84.4	62.5	32.4	52.9	65.8
RF-DETR (Ours)	N	31.2M	34.5	2.5	57.8	85.1	62.5	30.1	52.2	67.2
RF-DETR w/ Fine-Tuning (Ours)	N	31.2M	34.5	2.5	58.6	85.7	63.0	31.0	53.2	67.6
LW-DETR (Chen et al., 2024a)	S	14.6M	31.8	2.6	57.4	85.0	62.0	32.1	52.1	65.8
D-FINE (Peng et al., 2024)	S	10.2M	25.2	3.5	60.3	85.3	65.4	36.6	56.0	68.4
RF-DETR (Ours)	S	33.5M	62.4	3.7	60.9	87.5	66.1	34.2	55.7	69.6
RF-DETR w/ Fine-Tuning (Ours)	S	33.5M	62.4	3.7	61.2	87.7	66.1	34.9	55.6	69.5
RT-DETR (Zhao et al., 2024)	M	36.0M	100.0	4.3	59.6	85.7	64.6	36.4	54.6	67.3
LW-DETR (Chen et al., 2024a)	M	28.2M	83.9	4.3	59.8	86.8	64.9	34.0	54.4	68.9
D-FINE (Peng et al., 2024)	M	19.2M	56.6	5.6	60.6	85.5	65.8	36.0	56.6	67.5
RF-DETR (Ours)	M	33.5M	86.7	4.6	61.7	88.0	66.9	35.8	56.5	70.0
RF-DETR w/ Fine-Tuning (Ours)	M	33.5M	86.7	4.6	62.0	88.1	67.1	36.2	56.4	70.2
RF-DETR (Ours)	2XL	123.5M	410.2	15.6	63.3	88.9	69.0	38.7	58.2	71.6
RF-DETR (Ours) w/ Fine-Tuning	2XL	123.5M	410.2	15.6	63.5	89.0	69.2	38.9	58.3	71.7

Table 5: Ablation on Neural Architecture Search. We ablate the impact of each “tunable knob” on accuracy and latency below. Using a gentler set of hyperparameters compared to LW-DETR (e.g. smaller batch size, lower learning rate, replacing batch norm with layer norm) reduces performance by 1%. However, we regain this lost performance by replacing LW-DETR’s CAEV2 backbone with DINOV2. Importantly, the lower learning rate and layer-norm allow us to better preserve DINOV2’s foundational knowledge and allows us to train with larger batch sizes, making weight-sharing NAS more effective. Counterintuitively, introducing weight sharing NAS to the training scheme improves the performance of the base configuration even though patch size 14 isn’t in the NAS search space.

Model	# Params.	GFLOPS	Latency (ms)	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
LW-DETR (M)	28.2M	83.7	4.4	52.6	72.0	56.6	32.5	57.6	70.5
+ Gentler Hyperparameters	28.2M	83.7	4.4	51.6	71.1	55.5	31.7	56.4	69.4
+ DINOV2 Backbone	32.3M	78.2	4.7	53.6	72.7	58.0	34.3	58.3	72.4
+ Additional O365 Pre-Training	32.3M	78.2	4.7	54.3	73.4	58.8	35.8	59.2	72.3
+ Weight Sharing NAS	32.3M	78.2	4.7	54.6	73.4	59.3	36.3	59.3	72.1
+ Patch Size 14 → 16, Res 560 → 640	32.3M	78.5	4.7	54.4	73.2	59.1	35.9	59.2	72.1
+ Image Resolution 640 → 576	32.2M	64.2	4.0	53.6	72.4	58.2	34.8	58.6	72.0
+ # Windows per Block 4 → 2	32.2M	63.7	4.3	54.3	73.3	58.8	35.6	59.4	73.2
+ # Decoder Layers 3 → 4	33.7M	64.8	4.4	54.6	73.5	59.1	36.0	59.8	73.7
+ # Query Tokens 300 → 300	33.7M	64.8	4.4	54.6	73.5	59.1	36.0	59.8	73.7

foundation model families typically do not release lightweight ViT variants such as ViT-S or ViT-T, making it difficult to repurpose such models for real-time applications.

Rethinking Standard Accuracy Benchmarking Practices. Following prior work, we report all COCO results on the validation set. However, relying solely on the validation for both model selection and evaluation can lead to overfitting. For example, D-FINE (which builds on RT-DETR) conducts an extensive hyperparameter sweep on COCO’s validation set and reports its best model. However, evaluating this configuration on RF100-VL shows that D-FINE underperforms RT-DETR on the test set. In contrast, our method achieves state-of-the-art performance among all real-time detectors on both RF100-VL and COCO, demonstrating the robustness of our weight-sharing NAS. In

Table 6: Ablation on Backbone. We ablate the impact of using different backbone architectures for RF-DETR below. We find that DINOv2 achieves the highest performance, outperforming CAEv2 by 2.4%. All models are pretrained with 60 epochs of Objects-365 and the “Gentler Hyperparameters” setting. Note that SAM2 and SigLIPv2 perform poorly when evaluated in FP16. Therefore, we report FP16 TensorRT latency with FP32 ONNX accuracy for these two models as an upper bound on their performance if optimized for FP16.

LW-DETR (M) + Gentler Hyperparameters	# Params.	GFLOPS	Latency (ms)	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
w/ CAEv2 ViT/S-16-Truncated Backbone	28.3M	83.7	4.4	52.3	71.4	56.3	32.3	56.4	70.0
w/ DINOv2 ViT/S-14 Backbone	32.3M	78.2	4.7	54.3	73.4	58.8	35.8	59.2	72.3
w/ SigLIPv2 ViT/B-32 Backbone*	105.1M	81.6	4.8	50.4	70.4	53.7	28.0	55.3	73.0
w/ SAM2 Hierarchical Backbone*	44.0M	109.1	11.2	53.6	72.4	57.9	33.3	58.3	71.0

addition to evaluating on COCO, we advocate that future detectors should also evaluate on datasets with public validation and test splits like RF100-VL.

Limitations. Despite controlling for power throttling and GPU overheating during inference, our latency measurements still have a variance of up to 0.1ms due to the non-deterministic behavior of TensorRT during compilation. Specifically, TensorRT can introduce power throttling, which in turn affects the resulting engine and leads to random fluctuations in latency. Although the measurement of a given TensorRT engine is generally consistent, recompiling the same ONNX artifact can produce slightly different latency results. Therefore, we only report latencies with one digit of precision after the decimal place.

5 CONCLUSION

In this paper, we introduce RF-DETR, a state-of-the-art NAS-based method for fine-tuning specialist end-to-end object detectors for diverse target datasets and hardware platforms. Our approach outperforms prior state-of-the-art real-time methods on COCO and RF100-VL, improving upon D-FINE (nano) by 5% AP on COCO. Moreover, we highlight that current architectures, learning rate schedulers and augmentation schedulers are tailored to maximize performance on COCO, suggesting that the community should benchmark models on diverse, large-scale datasets to prevent implicit overfitting. Lastly, we highlight the high variance in latency benchmarking due to power throttling and propose a standardized protocol to improve reproducibility.

ACKNOWLEDGEMENTS

This work was supported in part by compute provided by NVIDIA DGX. We’d like to thank Brad Dwyer for reviewing early versions of our manuscript.

REFERENCES

- Lucas Beyer, Pavel Izmailov, Alexander Kolesnikov, Mathilde Caron, Simon Kornblith, Xiaohua Zhai, Matthias Minderer, Michael Tschannen, Ibrahim Alabdulmohsin, and Filip Pavetic. Flexivit: One model for all patch sizes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14496–14506, 2023.
- Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9157–9166, 2019.
- Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018a.
- Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018b.
- Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.
- Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pp. 213–229. Springer, 2020.
- Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, et al. Hybrid task cascade for instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4974–4983, 2019.
- Qiang Chen, Xiangbo Su, Xinyu Zhang, Jian Wang, Jiahui Chen, Yunpeng Shen, Chuchu Han, Ziliang Chen, Weixiang Xu, Fanrong Li, et al. Lw-detr: A transformer replacement to yolo for real-time detection. *arXiv preprint arXiv:2406.03459*, 2024a.
- Qiang Chen, Xiangbo Su, Xinyu Zhang, Jian Wang, Jiahui Chen, Yunpeng Shen, Chuchu Han, Ziliang Chen, Weixiang Xu, Fanrong Li, et al. Lw-detr: a transformer replacement to yolo for real-time detection. *arXiv preprint arXiv:2406.03459*, 2024b.
- Tianheng Cheng, Lin Song, Yixiao Ge, Wenyu Liu, Xinggang Wang, and Ying Shan. Yolo-world: Real-time open-vocabulary object detection. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2024.
- M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111, 2015.
- Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2009.
- Shenghao Fu, Qize Yang, Qijie Mo, Junkai Yan, Xihan Wei, Jingke Meng, Xiaohua Xie, and Wei-Shi Zheng. Llmdet: Learning strong open-vocabulary object detectors under the supervision of large language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 14987–14997, 2025.
- Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7036–7045, 2019.
- Junjie He, Pengyu Li, Yifeng Geng, and Xuansong Xie. Fastinst: A simple query-based model for real-time instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 23663–23672, 2023.

- Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.
- Glenn Jocher, Jing Qiu, and Ayush Chaurasia. Ultralytics YOLO, January 2023. URL <https://docs.ultralytics.com/models/yolov8>.
- Glenn Jocher, Jing Qiu, and Ayush Chaurasia. Ultralytics YOLO, January 2024. URL docs.ultralytics.com/models/yolol1.
- Mehar Khurana, Neehar Peri, Deva Ramanan, and James Hays. Shelf-supervised multi-modal pre-training for 3d object detection. *arXiv preprint arXiv:2406.10115*, 2024.
- Feng Li, Hao Zhang, Huazhe Xu, Shilong Liu, Lei Zhang, Lionel M Ni, and Heung-Yeung Shum. Mask dino: Towards a unified transformer-based framework for object detection and segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3041–3050, 2023.
- Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Li-juan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, et al. Grounded language-image pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10965–10975, 2022.
- Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *ECCV*, 2014.
- 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. *arXiv preprint arXiv:2201.12329*, 2022.
- Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.
- Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016.
- Yechi Ma, Neehar Peri, Shuoquan Wei, Wei Hua, Deva Ramanan, Yanan Li, and Shu Kong. Long-tailed 3d detection via 2d late fusion. *arXiv preprint arXiv:2312.10986*, 2023.
- Depu Meng, Xiaokang Chen, Zejia Fan, Gang Zeng, Houqiang Li, Yuhui Yuan, Lei Sun, and Jing-dong Wang. Conditional detr for fast training convergence. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 3651–3660, 2021.
- Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.
- Aljosa Osep, Tim Meinhardt, Francesco Ferroni, Neehar Peri, Deva Ramanan, and Laura Leal-Taixe. Better call sal: Towards learning to segment anything in lidar. 2024.
- 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. *arXiv preprint arXiv:2410.13842*, 2024.
- Neehar Peri, Achal Dave, Deva Ramanan, and Shu Kong. Towards long-tailed 3d detection. 2023.
- Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Valsudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024. URL <https://arxiv.org/abs/2408.00714>.
- Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

- Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.
- Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, 2015.
- Peter Robicheaux, Matvei Popov, Anish Madan, Isaac Robinson, Joseph Nelson, Deva Ramanan, and Neehar Peri. Roboflow100-vl: A multi-domain object detection benchmark for vision-language models. *arXiv preprint arXiv:2505.20612*, 2025.
- Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.
- Shuai Shao, Zeming Li, Tianyuan Zhang, Chao Peng, Gang Yu, Xiangyu Zhang, Jing Li, and Jian Sun. Objects365: A large-scale, high-quality dataset for object detection. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 8429–8438, 2019. doi: 10.1109/ICCV.2019.00852.
- Oriane Siméoni, Huy V Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, et al. Dinov3. *arXiv preprint arXiv:2508.10104*, 2025.
- Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1): 1929–1958, 2014.
- Ayca Takmaz, Cristiano Saltori, Neehar Peri, Tim Meinhardt, Riccardo de Lutio, Laura Leal-Taixe, and Aljosa Osep. Towards Learning to Complete Anything in Lidar. In *International Conference on Machine Learning (ICML)*, 2025.
- Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pp. 6105–6114. PMLR, 2019.
- Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2820–2828, 2019.
- 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*, pp. 10781–10790, 2020.
- 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. In *European conference on computer vision*, pp. 1–21. Springer, 2024.
- Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10734–10742, 2019.
- Yifan Xu, Mengdan Zhang, Chaoyou Fu, Peixian Chen, Xiaoshan Yang, Ke Li, and Changsheng Xu. Multi-modal queried object detection in the wild. *Advances in Neural Information Processing Systems*, 36, 2024.
- Xiaoju Ye. calfflops: a flops and params calculate tool for neural networks in pytorch framework, 2023. URL <https://github.com/MrYxJ/calculate-flops.pytorch>.

- Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M Ni, and Heung-Yeung Shum. Dino: Detr with improved denoising anchor boxes for end-to-end object detection. *arXiv preprint arXiv:2203.03605*, 2022a.
- Xinyu Zhang, Jiahui Chen, Junkun Yuan, Qiang Chen, Jian Wang, Xiaodi Wang, Shumin Han, Xiaokang Chen, Jimin Pi, Kun Yao, et al. Cae v2: Context autoencoder with clip target. *arXiv preprint arXiv:2211.09799*, 2022b.
- 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*, pp. 16965–16974, 2024.
- Xingyi Zhou, Rohit Girdhar, Armand Joulin, Philipp Krähenbühl, and Ishan Misra. Detecting twenty-thousand classes using image-level supervision. In *European Conference on Computer Vision*, pp. 350–368. Springer, 2022.
- Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020.
- Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.
- Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

A IMPLEMENTATION DETAILS

Training Hyperparameters. RF-DETR extends LW-DETR (Chen et al., 2024a) for Neural Architecture Search. We highlight key differences in our training procedure below. First, we pseudo-label Objects-365 (Shao et al., 2019) with SAM2 (Ravi et al., 2024) to allow us to pre-train the segmentation and detection heads on the same data. We use a learning rate of 1e-4 (LW-DETR uses 4e-4), and a batch size of 128 (LW-DETR uses the same). Similar to DINOv3 (Siméoni et al., 2025), we use an EMA scheduler since this is necessary for EMA to function. However, unlike DINOv3, we omit learning-rate warm-up. We clip all gradients greater than 0.1 and apply a per-layer multiplicative decay of 0.8 to preserve information (especially the earlier layers) in the DINOv2 backbone. We place our window attention blocks between layers {0, 1, 3, 4, 6, 7, 9, 10}, while LW-DETR places their window attention blocks between layers {0, 1, 3, 6, 7, 9}. Although we have the same number of windows, contiguous windowed blocks don’t require an additional reshape operation, making our implementation slightly more efficient. Further, we train with more multi-scale resolutions (0.5 to 1.5 scale) than LW-DETR (0.7 to 1.4 scale) to ensure that the augmentation is symmetric around the default scale. Notably, we add resolution as a “tunable knob” in our NAS search space, while LW-DETR uses it as a form of data augmentation. Our model training and inference code is available on [GitHub](#).

Latency Evaluation. We ensure fair evaluation between models by measuring detection accuracy and latency using the same artifact. To further standardize inference, we employ CUDA graphs in TensorRT, which pre-queue all kernels rather than requiring the CPU to launch them serially during execution. This optimization can accelerate some networks depending on the number and type of kernels used by the model. We observe that RT-DETR, LW-DETR, and RF-DETR benefit from this optimization. Further, CUDA graphs place LW-DETR on the same latency-accuracy curve as D-FINE, since CUDA graphs speed up LW-DETR but do not benefit D-FINE. We release our stand-alone latency benchmarking tool on [GitHub](#).

Pareto-Optimal Model Configurations in COCO. We present the Pareto-Optimal RF-DETR and RF-DETR-Seg configurations in Tables 7 and 8. We highlight notable trends about RF-DETR’s Pareto-Optimal architectures in Appendix L.

Table 7: RF-DETR COCO Detection Model Config.

Model Size	Resolution	Patch Size	Windows	Decoder Layers	Queries	Backbone
N	384	16	2	2	300	DINOv2-S
S	512	16	2	3	300	DINOv2-S
M	576	16	2	4	300	DINOv2-S
L	704	16	2	4	300	DINOv2-S
XL	700	20	1	5	300	DINOv2-B
2XL	880	20	2	5	300	DINOv2-B
Max	828	12	1	6	300	DINOv2-B

Table 8: RF-DETR-Seg COCO Segmentation Model Config.

Model Size	Resolution	Patch Size	Windows	Decoder Layers	Queries	Backbone
N	312	12	1	4	100	DINOv2-S
S	384	12	2	4	100	DINOv2-S
M	432	12	2	5	200	DINOv2-S
L	504	12	2	5	300	DINOv2-S
XL	624	12	2	6	300	DINOv2-S
2XL	768	12	2	6	300	DINOv2-S
Max	890	10	1	6	300	DINOv2-S

Parameter Sampling Grid. Lastly, we present our sampling grid for training and inference below. Importantly, we only drop decoder layers and queries during inference. We uniformly sample configurations during training and perform grid search over all configurations during inference to find Pareto-Optimal model configurations. RF-DETRs total training time is roughly two to four times as long as a non-NAS baseline, depending on the target dataset. However, RF-DETR can generate all size configurations from this single training run, while other non-NAS baselines must be re-trained for each new model size. We evaluate 6,468 network configurations (11 resolutions * 7 patch sizes * 7 decoder layers * 3 windows * 4 query settings) during architecture search. We estimate that this search takes approximately 10,000 GPU Hours (200 T4 GPUs * 48 hours).

Training Configurations

- Image Resolutions: 320, 384, 448, 512, 576, 640, 704, 768, 832, 896, 960
- Patch Sizes: 8, 10, 12, 16, 20, 24, 32
- Number of Decoder Layers: 6
- Number of Windows: 1, 2, 4
- Number of Queries: 300

Inference Configurations

- Image Resolutions: 320, 384, 448, 512, 576, 640, 704, 768, 832, 896, 960
- Patch Sizes: 8, 10, 12, 14, 16, 20, 24, 32
- Number of Decoder Layers: 0, 1, 2, 3, 4, 5, 6
- Number of Windows: 1, 2, 4
- Number of Queries: 50, 100, 200, 300

B ABLATION ON QUERY TOKENS AND DECODER LAYERS

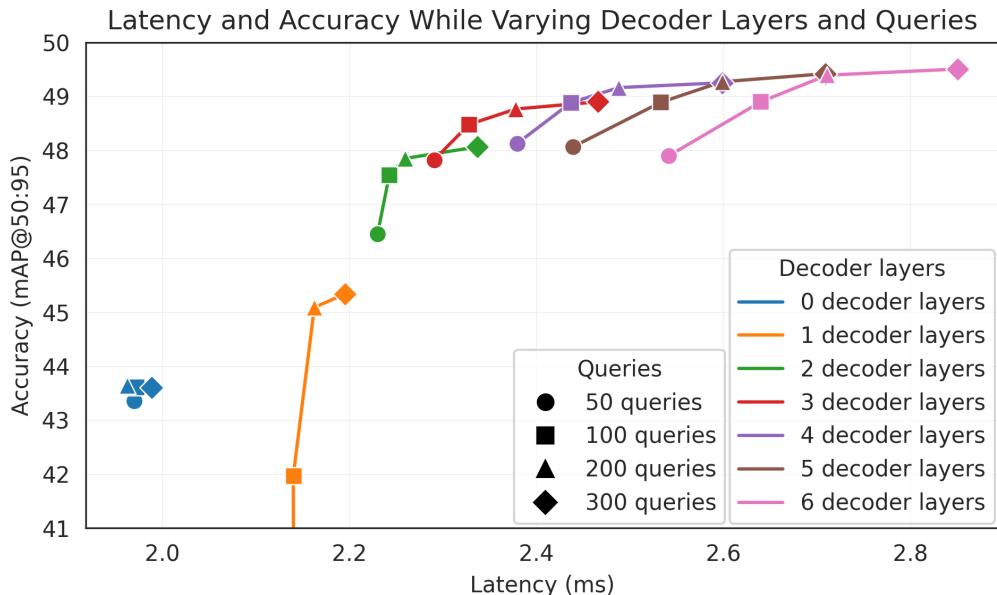


Figure 4: **Impact of Decoder Layers vs. Query Tokens.** We evaluate the impact of inference-time query dropping for trading-off accuracy and latency in RF-DETR (nano). Interestingly, we find that dropping the 100 lowest confidence queries does not significantly reduce performance, but modestly improves latency for all decoder layers.

We train RF-DETR (nano) with 300 object queries, following standard practices for real-time DETR-based object detectors. However, many datasets contain fewer than 300 objects per image. Therefore, processing all 300 queries can be computationally wasteful. LW-DETR (tiny) demonstrates that training with fewer queries can improve the latency-accuracy tradeoff. Rather than deciding on the optimal number of queries apriori, we find that we can drop queries at test time *without retraining* by discarding the lowest-confidence queries ordered by the confidence of the corresponding token at the output of the encoder. As shown in Figure 4, this yields meaningful latency-accuracy tradeoffs. In addition, prior work (Zhao et al., 2024) demonstrates that decoder layers can be pruned at test time, since each layer is supervised independently during training. We

find that it is possible to remove *all* decoder layers, relying solely on the initial query proposals from the two-stage DETR pipeline. In this case, there is no cross-attention to the encoder states or self-attention between queries, leading to a substantial runtime reduction. The resulting model resembles a single-stage YOLO-style architecture without NMS. Eliminating the final decoder layer reduces latency by 10% with only a 2 mAP drop in performance.

C BENCHMARKING FLOPs

We benchmark FLOPs for RF-DETR, GroundingDINO, and LLMDet with PyTorch’s `FlopCounterMode`. We find that `FlopCounterMode` closely reproduces FLOP counts obtained with custom benchmarking tools for YOLOv11, D-FINE, and LW-DETR. In practice, we also find that it provides more reliable results than CalFLOPs (Ye, 2023). Notably, LW-DETR’s FLOPs count is roughly twice that of the originally reported result (cf. Table 9). We posit that this discrepancy can be attributed to LW-DETR reporting MACs instead of FLOPs. We rely on the officially reported FLOPs counts from YOLOv11, YOLOv8, D-FINE, and RT-DETR.

Table 9: **FLOPs Benchmarking Comparison.** We compare FLOPs reported with custom benchmarking tools, CalFLOPs, and PyTorch’s `FlopCounterMode`. Notably, we find that `FlopCounterMode` closely matches the results reported with custom benchmarking code, suggesting that it is more reliable than prior generic benchmarking tools.

Model	Size	Reported	CalFLOPs	FlopCounterMode
D-FINE	S	25.2 M	25.2 M	25.5 M
LW-DETR	S	16.6 M	22.9 M	31.8 M
YOLO11	S	21.5 M	23.9 M	21.6 M

D IMPACT OF CLASS-NAMES ON OPEN- VOCABULARY DETECTORS

We evaluate the impact of fine-tuning open-vocabulary detectors like GroundingDINO with class names on RF100-VL in Table 10. Intuitively, GroundingDINO’s vision-language pre-training should be more useful when we prompt with class names (e.g. car, truck, bus) instead of class indices (e.g. 0, 1, 2). Using class names when fine-tuning provides more information to the VLM about the underlying data than is available to non-VLM detectors, potentially leading to better downstream performance. However, we find that fine-tuning GroundingDINO on RF100-VL yields nearly identical performance in both cases, suggesting that naively fine-tuning the end-to-end model mitigates the benefits of open-vocabulary pre-training. Future work should investigate ways of effectively fine-tuning VLMs to preserve foundational pre-training.

Table 10: **Evaluating the Impact of Class Names.** We evaluate the impact of using class-names when fine-tuning VLMs like GroundingDINO. We find that class-names do not provide significant benefit over prompting with class indices, suggesting that fine-tuning has diminished the impact of internet-scale pre-training.

Model	Size	# Params.	GFLOPS	Latency (ms)	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
RF100-VL										
GroundingDINO (Liu et al., 2023) w/ Standard Class Names	T	173.0M	1008.3	309.9*	62.3	88.8	67.8	39.2	57.7	69.5
GroundingDINO (Liu et al., 2023) w/ Class Index Names	T	173.0M	1008.3	309.9*	62.5	88.2	68.3	40.0	58.4	70.3

E BENCHMARKING LARGER MODEL VARIANTS

Detectors like LW-DETR (Chen et al., 2024a) and D-FINE (Peng et al., 2024) hand-design larger variants to scale up a model family. In contrast, NAS-based architectures like RF-DETR automatically discover scaling strategies through grid-based search. We analyze two families of RF-DETR models derived from distinct scaling strategies: one based on a DINOv2-S backbone and another based on a DINOv2-B backbone. To evaluate how well each family scales, we compare their NAS-generated Pareto curves against those of D-FINE. Specifically, at each D-FINE size, we identify the RF-DETR variant with the same backbone that maximizes performance at a comparable latency. For example, when comparing to D-FINE (small), we select the RF-DETR model that offers the

Table 11: **mAP@50:95 Gap of RF-DETR vs D-FINE at Similar Latencies** We compare how different RF-DETR model families scale relative to D-FINE. D-FINE (nano) is excluded since it was not pretrained on Objects-365 and is therefore not expected to follow similar scaling trends. For each RF-DETR backbone, we select the highest accuracy Pareto-optimal NAS-mined model with latency up to that of the corresponding D-FINE variant. Notably, RF-DETR (DINOv2-B) achieves better scalability than RF-DETR (DINOv2-S) and D-FINE. Note that none of the RF-DETR models for COCO are finetuned.

Method (Backbone)	S	M	L	XL
D-FINE (Peng et al., 2024)	50.6	55.4	57.2	59.3
RF-DETR (DINOv2-S)	+2.3	+0.9	-0.4	-1.1
RF-DETR (DINOv2-B)	-3.1	-1.3	-1.2	-0.7

Table 12: **COCO Detection Evaluation for Larger Model Variants.** We present RF-DETR’s performance for L, XL, and 2XL sizes on COCO below. Notably, D-FINE (x-large) outperforms RF-DETR (x-large) on mAP 50:95. However, RF-DETR (2x-large) beats D-FINE by 0.8 AP, and is the first real-time detector to surpass 60 AP on COCO.

Model	Size	# Params.	GFLOPS	Latency (ms)	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
Real-Time Object Detection w/ NMS										
YOLOv8 (Jocher et al., 2023)	L	43.7M	165.2	8.0	49.5	64.7	54.0	30.2	55.1	68.5
YOLOv11 (Jocher et al., 2024)	L	25.3M	86.9	6.5	49.9	64.9	54.5	30.4	55.9	68.1
YOLOv8 (Jocher et al., 2023)	XL	68.2M	257.8	11.3	50.5	65.6	55.1	30.0	56.2	69.5
YOLOv11 (Jocher et al., 2024)	XL	56.9M	194.9	10.5	50.9	66.1	55.4	31.5	56.6	68.7
End-to-End Real-Time Object Detection										
RT-DETR (Zhao et al., 2024)	R50	42M	136	8.5	55.0	73.3	59.8	37.9	59.7	71.6
LW-DETR (Chen et al., 2024a)	L	46.8M	137.5	6.9	56.1	74.6	61.0	37.1	60.4	73.0
D-FINE (Peng et al., 2024)	L	31M	91	7.5	57.2	74.9	62.2	40.6	61.4	73.7
RF-DETR (Ours)	L	33.9M	125.6	6.8	56.5	75.1	61.3	39.0	61.0	73.9
RT-DETR (Zhao et al., 2024)	R101	76M	259	12.0	56.1	74.5	61.1	38.1	60.4	73.4
LW-DETR (Chen et al., 2024a)	XL	118.0M	342.5	13.0	58.3	76.9	63.3	40.2	63.3	74.7
D-FINE (Peng et al., 2024)	XL	62M	202	11.5	59.3	76.8	64.6	42.1	64.2	76.3
RF-DETR (Ours)	XL	126.4M	299.3	11.5	58.6	77.4	63.8	40.3	63.9	76.2
RF-DETR (Ours)	2XL	126.9M	438.4	17.2	60.1	78.5	65.5	43.2	64.9	76.2
RF-DETR (Ours)	Max	132.4M	1742.5	98.0	61.8	79.7	67.7	47.5	66.1	76.0

best accuracy without exceeding D-FINE (small)’s latency. Note that these RF-DETR models are different than those reported in Tables 2 and 12.

As shown in Table 11, the DINOv2-S backbone family initially surpasses D-FINE in mAP@50:95 but fails to maintain this advantage at larger model sizes, suggesting that its scaling strategy is less effective than D-FINE’s manual design. In contrast, the DINOv2-B backbone family shows the opposite trend, where the performance gap between D-FINE and RF-DETR narrows as latency increases. This implies that at higher latencies, the DINOv2-B based RF-DETR models could surpass D-FINE (and indeed RF-DETR (2x-large) outperforms D-FINE on mAP 50:95). Importantly, expanding the D-FINE model family would require substantial additional engineering effort, whereas extending the RF-DETR model family is straightforward; higher-latency variants can be sampled directly from the same NAS search without re-training. We present the COCO and RF100-VL results of our larger variants in Tables 12, 13, and 14. We also include an RF-DETR Max variant on each dataset to show our method’s maximum performance with latency less than 100ms, a scale other model families don’t reach.

F PER-KNOB SENSITIVITY ANALYSIS

We evaluate the impact of varying resolution and patch size in Figure 5. Both curves follow a clear Pareto frontier, and are consistent with findings from prior work like FlexiViT (Beyer et al., 2023). RF-DETR is able to gracefully interpolate between seen (blue circles) and unseen (red stars) configurations during inference. Importantly, unseen configurations closely track the trend established by the seen configurations, demonstrating that RF-DETR generalizes beyond the model configurations encountered during training.

Table 13: **COCO Segmentation Evaluation for Larger Model Variants.** We present RF-DETR’s performance for L, XL, and 2XL sizes on the COCO segmentation benchmark below. We find that scaling up RF-DETR yields considerable performance improvements. In contrast, YOLOv8 and YOLOv11 do not significantly improve with scale.

Model	Size	# Params.	GFLOPS	Latency (ms)	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
Real-Time Instance Segmentation w/ NMS										
YOLOv8 (Jocher et al., 2023)	L	46.0M	220.5	9.7	39.0	60.5	41.7	18.0	44.7	57.8
YOLOv11 (Jocher et al., 2024)	L	27.6M	132.2	8.3	39.5	61.5	42.1	18.6	45.5	59.4
YOLOv8 (Jocher et al., 2023)	XL	71.8M	344.1	14.0	39.5	61.3	42.1	18.9	45.6	58.8
YOLOv11 (Jocher et al., 2024)	XL	62.1M	296.4	13.7	40.1	62.4	42.6	18.8	46.4	60.1
End-to-End Real-Time Instance Segmentation										
RF-DETR (Ours)	L	36.2M	151.1	8.8	47.1	70.5	50.9	28.4	52.1	65.6
RF-DETR (Ours)	XL	38.1M	260.0	13.5	48.8	72.2	53.1	30.6	53.3	65.9
RF-DETR (Ours)	2XL	38.6M	435.3	21.8	49.9	73.1	54.5	33.9	54.1	65.7
RF-DETR (Ours)	Max	40.1M	1668.2	95.6	50.5	74.0	55.4	34.6	54.2	65.4

Table 14: **RF100-VL Detection Evaluation for Larger Model Variants.** We present RF-DETR’s performance for L, XL, and 2XL sizes on RF100-VL below. Notably, RF-DETR (x-large) beats D-FINE by 0.5 AP. Fine-tuning RF-DETR (x-large) improves performance by an additional 0.4 AP.

Model	Size	# Params.	GFLOPS	Latency (ms)	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
Real-Time Object Detection w/ NMS										
YOLOv8 (Jocher et al., 2023)	L	43.7M	165.2	7.9	56.5	82.1	61.1	7.1	46.0	48.9
YOLOv11 (Jocher et al., 2024)	L	25.3M	86.9	6.4	56.5	82.2	61.0	6.4	45.5	49.0
YOLOv8 (Jocher et al., 2023)	XL	68.2M	257.8	11.2	56.5	82.3	61.0	6.6	45.7	47.9
YOLOv11 (Jocher et al., 2024)	XL	56.9M	194.9	10.3	56.2	81.7	60.8	6.1	45.9	48.1
End-to-End Real-Time Object Detection										
RT-DETR (Zhao et al., 2024)	R50	42M	136	8.4	61.7	87.7	66.9	38.1	57.1	69.4
LW-DETR (Chen et al., 2024a)	L	46.8M	137.5	6.8	61.5	87.4	67.0	37.1	56.4	69.0
D-FINE (Peng et al., 2024)	L	31M	91	7.5	61.6	86.4	67.2	37.8	56.5	70.1
RF-DETR (Ours)	L	34.1M	119.1	6.2	62.0	88.1	67.3	36.9	57.1	70.2
RF-DETR (Ours) w/ Fine-Tuning	L	34.1M	119.1	6.2	62.3	88.2	67.4	37.1	57.2	70.3
RT-DETR (Zhao et al., 2024)	R101	76M	259	11.9	61.0	87.4	66.2	36.6	56.3	68.2
LW-DETR (Chen et al., 2024a)	XL	118.0M	342.5	13.0	62.1	87.9	67.6	37.4	57.1	70.2
D-FINE (Peng et al., 2024)	XL	59.3	76.8	11.4	62.2	86.9	68.0	37.6	57.4	69.7
RF-DETR (Ours)	XL	35.0M	199.0	9.7	62.6	88.5	67.9	39.0	57.8	70.4
RF-DETR (Ours) w/ Fine-Tuning	XL	35.0M	199.0	9.7	63.0	88.7	68.2	38.8	58.2	70.6
RF-DETR (Ours)	2XL	123.5M	410.2	15.6	63.3	88.9	69.0	38.7	58.2	71.6
RF-DETR (Ours) w/ Fine-Tuning	2XL	123.5M	410.2	15.6	63.5	89.0	69.2	38.9	58.3	71.7

G IMPACT OF NAS FINE-TUNING ON COCO

We find that fine-tuning after NAS provides limited benefit for COCO. We posit that the NAS “architecture augmentation” acts as a strong regularizer, and additional training without this regularization leads to degraded performance. Specifically, when models are pre-trained with strong regularization, removing the regularization during fine-tuning leads to overfitting. As shown in Tables 15 and 16, this trend is consistent across both detection and segmentation tasks. Interestingly, models trained on RF100-VL benefit more from fine-tuning, likely because they require more than 100 epochs to

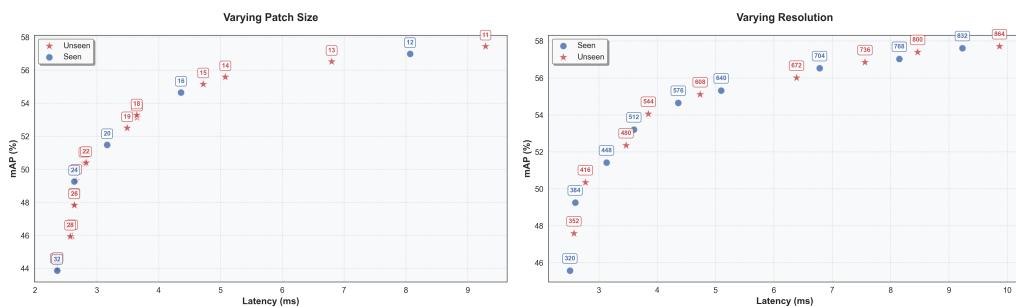


Figure 5: **Per Knob Sensitivity Analysis.** Despite never seeing certain resolutions and patch sizes, RF-DETR is able to gracefully interpolate to novel model configurations.

converge. In such cases, we posit that reducing the total number of NAS configurations during training, or training for more than 100 epochs with weight-sharing NAS can improve performance.

Table 15: **COCO Detection Fine-Tuning Evaluation.** We find that fine-tuning after NAS provides limited benefit for COCO detection, particularly for larger model sizes.

Model	Size	# Params.	GFLOPS	Latency (ms)	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
End-to-End Real-Time Object Detectors										
RF-DETR (Ours)	N	30.5M	31.9	2.3	48.0	67.0	51.4	25.2	53.5	70.0
RF-DETR (Ours) w/ Fine-Tuning	N	30.5M	31.9	2.3	+0.4	+0.6	+0.3	+0.1	+0.1	+1.3
RF-DETR (Ours)	S	32.1M	59.8	3.5	52.9	71.9	57.0	32.0	58.3	73.0
RF-DETR (Ours) w/ Fine-Tuning	S	32.1M	59.8	3.5	+0.1	+0.2	+0.2	-0.2	+0.2	+0.1
RF-DETR (Ours)	M	33.7M	78.8	4.4	54.7	73.5	59.2	36.1	59.7	73.8
RF-DETR (Ours) w/ Fine-Tuning	M	33.7M	78.8	4.4	+0.0	+0.1	+0.0	-0.1	+0.1	-0.1
RF-DETR (Ours)	L	33.9M	125.6	6.8	56.5	75.1	61.3	39.0	61.0	73.9
RF-DETR (Ours) w/ Fine-Tuning	L	33.9M	125.6	6.8	+0.0	+0.0	+0.0	-0.1	+0.1	+0.1
RF-DETR (Ours)	XL	126.4M	299.3	11.5	58.6	77.4	63.8	40.3	63.9	76.2
RF-DETR (Ours) w/ Fine-Tuning	XL	126.4M	299.3	11.5	+0.3	+0.1	+0.2	+0.5	+0.4	+0.1
RF-DETR (Ours)	2XL	126.9M	438.4	17.2	60.1	78.5	65.5	43.2	64.9	76.2
RF-DETR (Ours) w/ Fine-Tuning	2XL	126.9M	438.4	17.2	+0.1	+0.0	+0.3	+0.5	+0.2	+0.1

Table 16: **COCO Segmentation Fine-Tuning Evaluation.** We find that fine-tuning after NAS provides limited benefit for COCO segmentation, particularly for larger model sizes.

Model	Size	# Params.	GFLOPS	Latency (ms)	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
End-to-End Real-Time Object Detectors										
RF-DETR-Seg. (Ours)	N	33.6M	50.0	3.4	40.3	63.0	42.6	16.3	45.3	63.6
RF-DETR-Seg. w/ Fine-Tuning (Ours)	N	33.6M	50.0	3.4	+0.1	+0.4	+0.0	-0.5	+0.2	+0.7
RF-DETR-Seg. (Ours)	S	33.7M	70.6	4.4	43.1	66.2	45.9	21.9	48.5	64.1
RF-DETR w/ Fine-Tuning (Ours)	S	Did	Not	Improve	-	-	-	-	-	-
RF-DETR-Seg. (Ours)	M	35.7M	102.0	5.9	45.3	68.4	48.8	25.5	50.4	65.3
RF-DETR w/ Fine-Tuning (Ours)	M	Did	Not	Improve	-	-	-	-	-	-
RF-DETR (Ours)	L	36.2M	151.1	8.8	47.1	70.5	50.9	28.4	52.1	65.6
RF-DETR (Ours) w/ Fine-Tuning	L	Did	Not	Improve	-	-	-	-	-	-
RF-DETR (Ours)	XL	38.1M	260.0	13.5	48.8	72.2	53.1	30.6	53.3	65.9
RF-DETR (Ours) w/ Fine-Tuning	XL	Did	Not	Improve	-	-	-	-	-	-
RF-DETR (Ours)	2XL	38.6M	435.3	21.8	49.9	73.1	54.5	33.9	54.1	65.7
RF-DETR (Ours) w/ Fine-Tuning	2XL	Did	Not	Improve	-	-	-	-	-	-

H IMPACT OF DATASET CHARACTERISTICS ON TUNABLE KNOBS

We evaluate the impact of different dataset characteristics on optimal network configurations for RF-DETR (medium) on RF100-VL in Table 17 and Figure 6. We compare different combinations of object size, number of spatial locations, number of decoder layers, number of windows, number of classes, number of annotations, objects per image, and number of queries below. We do not expect these relationships to be linear, but expect that they will be monotonic (e.g. non-zero slope). For example, we find strong correlations between the number of classes and number of decoder layers, objects per image and number of queries, spatial locations and number of windows. Notably, we do not find strong correlations between object size and number of decoder layers, number of annotations and number of decoder layers, and objects per image and number of decoder layers

Table 17: **Regression Analysis.** We evaluate the linear relationships between various dataset characteristics and model parameters. Although these correlations are non-linear, the line-of-best-fit helps explain general trends. P-values indicate the significance of the correlation.

Relationship	Slope	Intercept	R-squared	P-value	Std. Error
Average Object Size vs Number of Spatial Locations	-0.009	0.384	0.148	0.000	0.002
Average Object Size vs Number of Decoder Layers	-0.000	0.044	0.000	0.971	0.006
Average Object Size vs Number of Windows	-0.008	0.065	0.019	0.170	0.006
Number of Classes vs Number of Decoder Layers	0.837	2.125	0.026	0.106	0.513
Number of Annotations vs Number of Decoder Layers	698.763	6654.795	0.001	0.706	1848.733
Average Objects per Image vs Number Decoder Layers	0.163	7.618	0.000	0.857	0.904
Average Objects per Image vs Number of Queries	0.020	4.457	0.019	0.171	0.015
Number of Spatial Locations vs Number of Windows	1.722	32.984	0.492	0.000	0.177

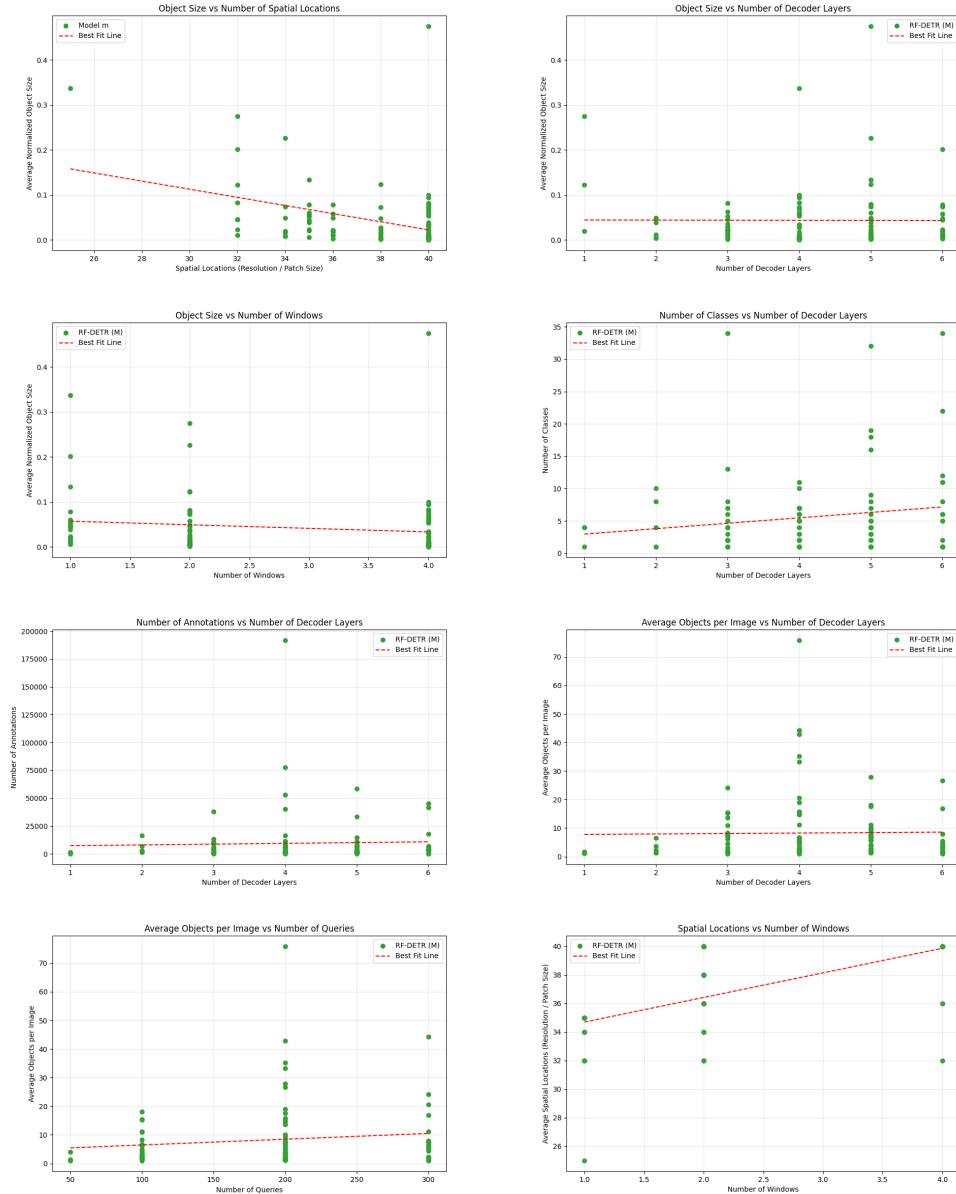


Figure 6: Impact of Dataset Characteristics on Tunable Knobs. We visualize the correlation between key dataset characteristics and several tunable architectural knobs above. Across all subplots, individual points represent different datasets within RF100-VL while dashed lines show corresponding linear trends. Overall, the results indicate that object-centric properties of datasets such as average object size, number of classes, object density, and total annotations tend to have only modest influence on architectural choices like the number of decoder layers, number of windows, number of queries, and spatial resolution. Slight positive or negative trends appear in some cases (e.g., more classes or more objects per image loosely correlating with deeper decoders or higher query counts), but the scatter remains wide, suggesting no strong deterministic relationship. These findings highlight that while dataset characteristics offer some intuition for selecting model hyperparameters, optimal configurations ultimately depend on a combination of factors rather than any single dataset attribute.

I IMPACT OF FIXED ARCHITECTURE ON RF100-VL

We evaluate the impact of transferring a NAS architecture optimized for COCO to RF100-VL in Table 18 and Figure 7. We find that these fixed architecture models perform remarkably well without further dataset-specific NAS. Specifically, RF-DETR (large) model with a fixed architecture achieves the best performance among all prior real-time models on COCO. However, dataset-specific NAS yields significant improvements. Notably, the performance delta from LW-DETR to the fixed architecture is comparable to the improvement from the fixed architecture to the NAS-optimized model on the target dataset for nano, small, and medium scale models.

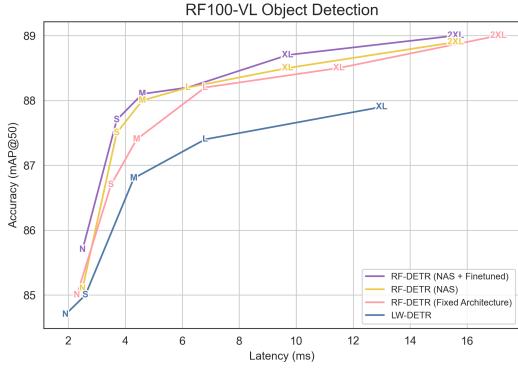


Figure 7: Ablating Fixed Architecture RF100-VL. We evaluate the benefit of dataset-specific NAS by transferring the COCO-optimized RF-DETR architecture to RF100-VL. Although the fixed architecture was not tuned for RF100-VL, it still outperforms LW-DETR. Running NAS directly on RF100-VL further improves performance over the fixed architecture. Additional fine-tuning provides consistent gains across all model sizes, with particularly strong improvements for smaller models. This is consistent with our observations on COCO object detection.

Table 18: RF100-VL Fixed Architecture Evaluation. We evaluate the transfer of architectures optimized for COCO to RF100-VL. Fixed architecture models perform well without additional dataset-specific NAS, with the RF-DETR (large) model achieving the best performance among prior real-time models. However, dataset-specific NAS provides significant further gains.

Model	Size	# Params.	GFLOPS	Latency (ms)	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
End-to-End Real-Time Object Detectors										
RF-DETR (Ours) Fixed Architecture	N	30.5M	31.9	2.3	57.7	85.0	61.9	30.8	51.5	67.4
RF-DETR (Ours)	N	31.2M	34.5	2.5	57.8	85.1	62.5	30.1	52.2	67.2
RF-DETR w/ Fine-Tuning (Ours)	N	31.2M	34.5	2.5	58.6	85.7	63.0	31.0	53.2	67.6
RF-DETR (Ours) Fixed Architecture	S	32.1M	59.8	3.5	60.2	86.7	65.0	34.2	54.4	68.9
RF-DETR (Ours)	S	33.5M	62.4	3.7	60.9	87.5	66.1	34.2	55.7	69.6
RF-DETR w/ Fine-Tuning (Ours)	S	33.5M	62.4	3.7	61.2	87.7	66.1	34.9	55.6	69.5
RF-DETR (Ours) Fixed Architecture	M	33.7M	78.8	4.4	61.2	87.4	66.4	35.8	56.1	69.8
RF-DETR (Ours)	M	33.6M	91.0	4.6	61.7	88.0	66.9	35.8	56.5	70.0
RF-DETR w/ Fine-Tuning (Ours)	M	33.6M	91.0	4.6	62.0	88.1	67.1	36.2	56.4	70.2
RF-DETR (Ours) w/ Fixed Architecture	L	33.9M	125.6	6.8	62.2	88.2	67.8	37.7	57.0	70.5
RF-DETR (Ours)	L	34.1M	119.1	6.2	62.0	88.1	67.3	36.9	57.1	70.2
RF-DETR (Ours) w/ Fine-Tuning	L	34.1M	119.1	6.2	62.3	88.2	67.4	37.1	57.2	70.3
RF-DETR (Ours, DINov2-Base) w/ Fixed Architecture	XL	126.4M	299.3	11.5	62.9	88.5	68.6	37.0	57.5	71.3
RF-DETR (Ours)	XL	35.0M	199.0	9.7	62.6	88.5	67.9	39.0	57.8	70.4
RF-DETR (Ours) w/ Fine-Tuning	XL	35.0M	199.0	9.7	63.0	88.7	68.2	38.8	58.2	70.6
RF-DETR (Ours, DINov2-Base) w/ Fixed Architecture	2XL	126.9M	438.4	17.1	63.2	89.0	69.3	38.4	58.4	71.5
RF-DETR (Ours, DINov2-Base)	2XL	123.5M	410.2	15.6	63.3	88.9	69.0	38.7	58.2	71.6
RF-DETR (Ours, DINov2-Base) w/ Fine-Tuning	2XL	123.5M	410.2	15.6	63.5	89.0	69.2	38.9	58.3	71.7

J ABLATION ON BACKBONE ARCHITECTURE WITH RF20-VL

In Table 19, we reproduce our ablation on the impact of backbone architecture on downstream model performance (Table 6) on RF20-VL. All trends from the main paper hold.

Table 19: Ablation on Backbone RF20-VL.

LW-DETR (M) + Gentler Hyperparameters	# Params.	GFLOPS	Latency (ms)	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
w/ CAEV2 ViT/S-16-Truncated Backbone	28.3M	83.7	4.3	64.4	92.2	71.2	33.8	56.8	71.9
w/ DINov2 ViT/S-14 Backbone	32.3M	78.2	4.6	65.2	92.5	72.8	37.8	58.2	72.6
w/ SigIPv2 ViT/B-32 Backbone*	105.1M	81.6	4.5	62.2	91.0	67.8	28.9	54.1	70.3
w/ SAM2 Hiera-S Backbone*	44.0M	109.1	11.2	65.2	92.5	72.7	37.8	58.2	72.1

K ANALYSIS ON BUFFERING

We further analyze the impact of buffering on the relative ordering of inference speed in Table 20. Notably, we find that buffering beyond 200ms does not change latency measurements. However, we acknowledge that adding a 200ms buffer after every forward pass considerably increases overall inference time. Future work should consider alternatives to buffering to address power throttling.

Table 20: **Analysis on Buffering.** We evaluate models with different amounts of buffering between consecutive forward passes. We find that buffering beyond 200ms does not provide any additional stability to latency measurements.

Model	mAP	0 ms	200 ms	400 ms	800 ms
YOLOv8 (M)	47.3	5.5 ms	5.4 ms	5.4 ms	5.4 ms
YOLOv11 (M)	48.4	5.0 ms	5.0 ms	5.1 ms	5.1 ms
RT-DETR (R18)	49.0	4.4 ms	4.4 ms	4.4 ms	4.4 ms
LW-DETR (M)	52.6	4.5 ms	4.3 ms	4.3 ms	4.3 ms
D-FINE (M)	54.9	5.7 ms	5.4 ms	5.4 ms	5.4 ms
RF-DETR (M)	54.7	4.7 ms	4.4 ms	4.4 ms	4.4 ms

L DISCUSSION ON NOTABLE DISCOVERED ARCHITECTURES

Several trends emerge from our weight-sharing NAS. First, we note that all tunable “knobs” are used when defining Pareto-optimal model families, validating our search space. This suggests that expanding the search space may further improve downstream performance.

Across Pareto-optimal models, patch size is consistent within model families. For example, the optimal patch size for RF-DETRs with a DINOv2-S backbone is 16, RF-DETRs with a DINOv2-B backbone is 20, and RF-DETR-Segs with a DINOv2-S backbone is 12. Pareto-optimal models also jointly scale encoder and decoder compute: patch size, number of windows, and resolution impact the encoder, while decoder depth, and number of queries affect the decoder. For RF-DETR-Seg, scaling resolution impacts the segmentation head. We find that using 2 windows in the encoder is typically optimal and resolution scales within a model family as we increase latency. On COCO, RF-DETR scales decoder depth while keeping the number of queries fixed, while RF-DETR-Seg simultaneously scales both axes. This likely reflects a minimum viable segmentation head depth; to offset its latency, RF-DETR-Seg reduces its total number of queries, yielding a thin, deep decoder, in contrast to RF-DETRs wide, shallow decoder.

Next, we find that RF-DETR’s performance is more correlated with the total number of spatial locations (e.g. resolution divided by patch size) rather than resolution or patch size alone. Scaling resolution with a fixed patch size yields similar results to scaling patch size with a fixed resolution, since vision transformers are agnostic to absolute input resolution after the patchify-and-project operation. To verify this, we constructed an alternative model family with fixed resolution (640) and varied patch sizes to preserve the total number of spatial locations. Specifically, we evaluate RF-DETR (nano) with a patch size of 27, RF-DETR (small) with a patch size of 21, and RF-DETR (medium) with a patch size of 18. Surprisingly, all model results are nearly identical to the Pareto-optimal family. Notably, patch sizes of 27 and 18 were unseen during training, demonstrating RF-DETR’s strong generalization to novel patch sizes (Beyer et al., 2023). However, we find that this trend does not hold for RF-DETR-Seg because segmentation features are always upsampled to $\frac{1}{4}$ of the input image resolution. As a result, scaling RF-DETR-Seg’s input resolution affects both the number of spatial locations and the segmentation feature resolution. Specifically, RF-DETR-Seg (nano, small, medium) uses input resolutions of 312, 384, and 432 with patch size 12, yielding segmentation feature resolutions of 78, 96, and 108 and 26, 32, and 36 spatial locations, respectively. In contrast, increasing patch size alone (e.g., patch size 16 at input resolution 576) preserves spatial locations while increasing segmentation feature resolution. As a result, although RF-DETR (medium) and RF-DETR-Seg (medium) both use 36 spatial locations, RF-DETR-Seg operates at lower input resolution, demonstrating that coupling segmentation feature resolution with input resolution shifts the Pareto-optimal operating point.

Further, we find that most Pareto-optimal RF-DETR models perform best with 2 windows, whereas LW-DETR achieves the best performance with 4 windows. We attribute this difference to how each architecture handles class tokens. LW-DETRs CAEv2 backbone omits the class token, while RF-

DETR’s DINOv2 backbone relies on it as a key part of pre-training. To make windowed attention compatible with class tokens, we duplicate the class token for each window. During global attention, window-level class tokens attend to one another, while all other tokens attend to all class tokens. In practice, RF-DETR (nano), RF-DETR (small), and RF-DETR (medium) all use 2 windows, since duplicating class tokens for additional windows reduces runtime efficiency. As a result, unlike LW-DETR, RF-DETR does not benefit from scaling to 4 windows.

Lastly, we note that dataset characteristics influence optimal discovered architectures. We find that the optimal low latency models on RF100-VL tend to use fewer queries than the COCO models of equivalent latency. We attribute this to RF100-VL datasets having fewer objects per image than COCO.

M VISUALIZING MODEL PREDICTIONS

We visualize model predictions from RF-DETR (nano) and relevant detection and segmentation baselines in Figure 8. We find that RF-DETR (nano) predicts fewer false positives (e.g. mistaking sign post for person). Similarly, RF-DETR-Seg. (nano) predicts more precise object boundaries.

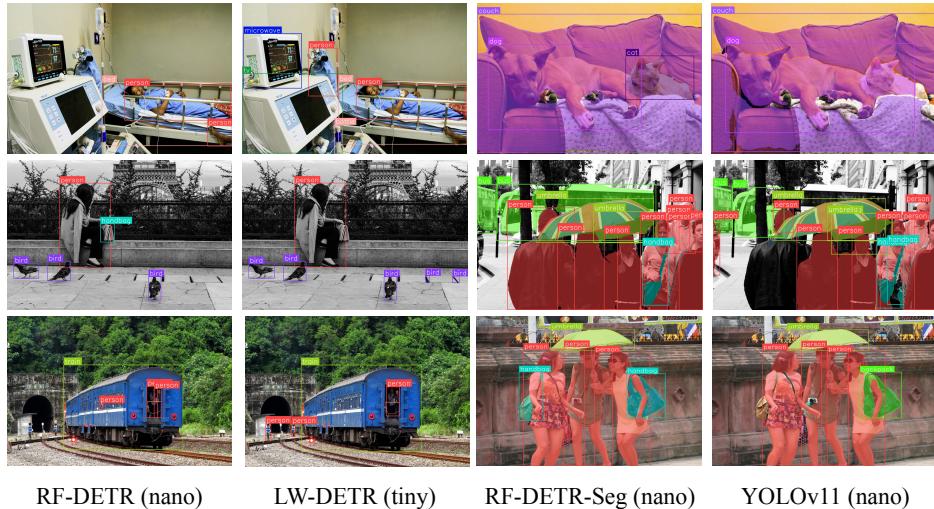


Figure 8: **Visualizing Model Predictions.** On the left, we compare detections from RF-DETR (nano) and LW-DETR (tiny). On the right, we compare instance segmentation masks from RF-DETR-Seg (nano) and YOLOv11 (nano)