YOLO-NAS-Bench: A Surrogate Benchmark with Self-Evolving Predictors for YOLO Architecture Search

Anonymous CVPR submission

Paper ID

Abstract

Neural Architecture Search (NAS) for object detection is severely bottlenecked by high evaluation cost, as fully training each candidate YOLO architecture on COCO demands days of GPU time. Meanwhile, existing NAS benchmarks largely target image classification, leaving the detection community without a comparable benchmark for NAS evaluation. To address this gap, we introduce ***YOLO-NAS-Bench****, the first surrogate benchmark tailored to YOLO-style detectors. YOLO-NAS-Bench defines a search space spanning channel width, block depth, and operator type across both backbone and neck, covering the core modules of YOLOv8 through YOLO12. We sample 1,000 architectures via random, stratified, and Latin Hypercube strategies, train them on COCO-mini, and build a LightGBM surrogate predictor. To sharpen the predictor in the high-performance regime most relevant to NAS, we propose a* Self-Evolving Mechanism *that progressively aligns the predictor's training distribution with the high-performance frontier, by using the predictor itself to discover and evaluate informative architectures in each iteration. This method grows the pool to 1,500 architectures and raises the ensemble predictor's R^2 from 0.770 to 0.815 and Sparse Kendall Tau from 0.694 to 0.752, demonstrating strong predictive accuracy and ranking consistency. Using the final predictor as the fitness function for evolutionary search, we discover architectures that surpass all official YOLOv8–YOLO12 baselines at comparable latency on COCO-mini, confirming the predictor's discriminative power for top-performing detection architectures.*

1. Introduction

Neural Architecture Search (NAS) demonstrates notable success in automating the design of high-performing image classifiers [17, 19, 28]. However, extending NAS to object detection remains expensive: detection models are much larger, datasets such as COCO [16] require far more compute per training run, and the complete search space needs to jointly consider backbone, neck, and head components. Fully training a single YOLO-family architecture on COCO can take days on a multi-GPU cluster. Evaluating thousands of candidates—as NAS algorithms typically demand—thus becomes infeasible.

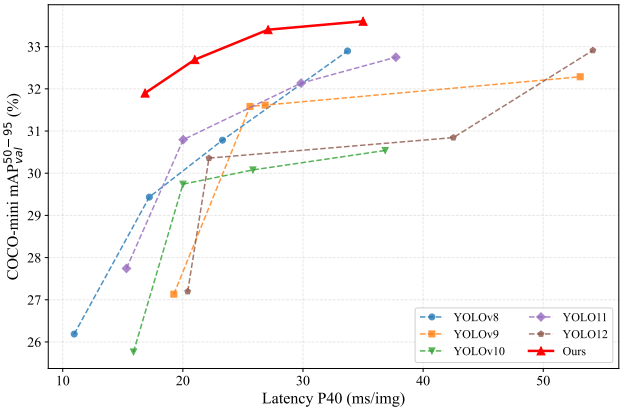

Figure 1. **Latency vs. mAP on COCO-mini.** Architectures discovered by our predictor-guided EA search consistently Pareto-dominate all official YOLO baselines (v8–v12) across the full latency spectrum, demonstrating the strong discriminative power of the YOLO-NAS-Bench surrogate predictor.

NAS benchmarks significantly accelerate research in the classification domain by providing precomputed look-up tables or surrogate predictors, thereby decoupling NAS algorithm development from the expensive architecture evaluation loop and offering a unified benchmark for fair comparison. NAS-Bench-101 [24] and NAS-Bench-201 [6] offer tabular benchmarks with exhaustive evaluation of small cell-based search spaces. NAS-Bench-301 [25] scales to the full DARTS space via a surrogate predictor. Yet all these benchmarks target image classification—their search spaces, training pipelines, and evaluation protocols cannot be directly transferred to detection architectures. Meanwhile, detection-specific NAS methods such as Det-NAS [5], OPANAS [15], and YOLO-NAS [1] each de-

fine bespoke search spaces and evaluation setups, making fair cross-method comparison difficult. The community still lacks unified benchmarks for object detection NAS.

We address this gap with **YOLO-NAS-Bench**, the first surrogate benchmark designed specifically for YOLO-style object detectors. YOLO-NAS-Bench features three tightly integrated components: (1) a comprehensive search space spanning channel width, block depth, and operator type across both backbone and neck, covering the core building blocks of YOLOv8 through YOLO12; (2) an initial ground-truth database of 1,000 architectures sampled via complementary strategies and fully trained on COCO-mini under a unified protocol; and (3) a *Self-Evolving Predictor* that iteratively enriches the training pool with high-value architectures discovered by evolutionary search, culminating in an ensemble of 10 LightGBM models that achieves sKT = 0.752 and $R^2 = 0.815$. We validate the predictor's practical utility by using its predicted mAP as the fitness function for EA-based search [19]. As shown in Fig. 1, the discovered architectures *surpass all official YOLO baselines* (v8–v12) at comparable latency.

Our contributions are summarized as follows:

- We design a YOLO-oriented search space covering backbone and neck with channel, depth, and operator dimensions that span the key modules of YOLOv8–YOLO12. Through random, stratified, and Latin Hypercube sampling, we build a benchmark database of 1,000 architectures fully trained on COCO-mini.
- We propose a Self-Evolving Predictor that bridges the distribution gap between uniformly sampled training data and the high-performance frontier critical to NAS. The predictor guides latency-bucketed evolutionary search to discover promising architectures, which are trained and fed back to retrain the predictor, forming a self-evolving loop. Over 10 rounds, the pool grows to 1,500 architectures and sKT rises from 0.694 to 0.752.
- We demonstrate the predictor's practical utility by deploying it as the fitness function for EA search and discovering architectures that surpass official YOLO baselines (v8–v12) at comparable latency on COCO-mini.

2. Related Work

Real-Time Object Detectors. The YOLO family has driven rapid progress in real-time detection. YOLOv8 [12] introduces the C2f module with streamlined cross-stage feature fusion. YOLOv9 [23] proposes GELAN and Programmable Gradient Information (PGI) for improved gradient flow. YOLOv10 [22] eliminates the NMS post-processing step via consistent dual assignments. YOLO11 [11] advances efficiency with C3k2 re-parameterized blocks and C2PSA attention modules. YOLO12 [20] adopts an attention-centric design, pushing the accuracy–speed Pareto frontier further. Beyond the YOLO lineage, RT-DETR [27] demonstrates that Transformer-based detectors can also achieve real-time speeds. Despite these continuous improvements, architecture design in the detection domain remains predominantly manual, motivating the need for automated search and, consequently, a standardized benchmark to evaluate NAS algorithms fairly.

NAS Benchmarks. Tabular benchmarks such as NAS-Bench-101 [24] (423 k architectures on CIFAR-10) and NAS-Bench-201 [6] (15,625 cell-based architectures across three datasets) are instrumental in democratizing NAS research for image classification by removing the evaluation bottleneck. NAS-Bench-301 [25] scales to the full DARTS search space by training a surrogate predictor evaluated via the Sparse Kendall Tau (sKT) metric, establishing a protocol that subsequent benchmarks adopt. NAS-Bench-360 [21] broadens the scope to diverse tasks and data modalities, yet does not include detection-specific search spaces. YOLOBench [14] characterizes a fixed set of YOLO model families but cannot be adapted for NAS algorithm evaluation. Existing benchmarks predominantly target classification or evaluate fixed model families. Consequently, surrogate benchmarks for detection-oriented NAS remain scarce, and our work addresses this gap.

NAS for Object Detection. Several works apply NAS to detection sub-problems: NAS-FPN [8] searches for feature pyramid topologies; Det-NAS [5] performs backbone architecture search with a one-shot supernet; OPANAS [15] optimizes FPN neck architectures via an accuracy predictor; SP-NAS [10] proposes a serial-to-parallel search strategy for detection backbones; and YOLO-NAS [1] employs the proprietary AutoNAC framework to design YOLO architectures. Each of these methods defines its own search space, training recipe, and evaluation setup, making direct cross-method comparison difficult. Our paper addresses this fragmentation by providing a shared search space, a precomputed architecture–performance database, and a calibrated surrogate predictor that any NAS algorithm can query at near-zero cost.

3. Method

An overview of the YOLO-NAS-Bench is illustrated in Fig. 2. Starting from a carefully designed search space over YOLO-style architectures, we sample and fully train 1,000 architectures on COCO-mini to build a ground-truth performance database. Each configuration is converted into a compact feature vector, upon which a LightGBM surrogate predictor is trained. To further strengthen the predictor in the high-performance regime that matters most for NAS, we propose a *Self-Evolving Predictor* loop that iteratively

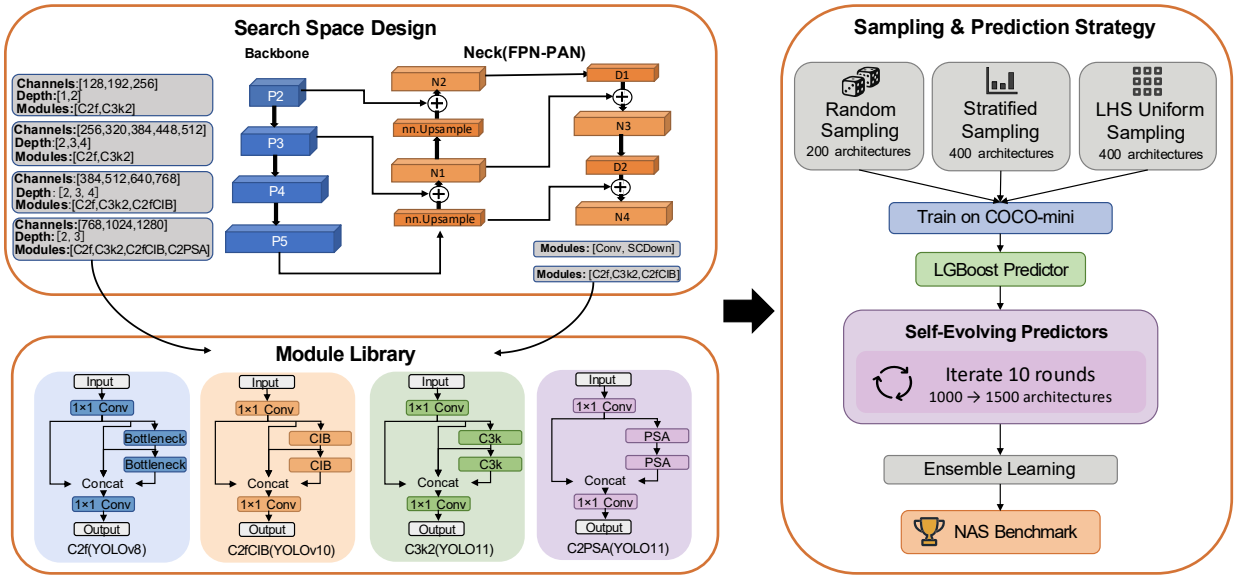

Figure 2. **Overview of the YOLO-NAS-Bench pipeline.** (1) A YOLO-style search space spanning channel, depth, and operator dimensions across both backbone and neck is defined. (2) 1,000 architectures are sampled via three complementary strategies and trained on COCO-mini. (3) A LightGBM predictor is trained on the resulting {architecture, mAP} pairs. (4) The Self-Evolving Predictor iteratively expands the pool with high-value architectures discovered by evolutionary search, and retrains the predictor over 10 rounds, yielding an ensemble of 10 LightGBM models over 1,500 architectures.

discovers, trains, and assimilates promising architectures.

3.1. Benchmark Construction

Search Space Design. We design a comprehensive search space that covers both the backbone and the neck of YOLO-style detectors, while keeping the detection head fixed. The space is parameterized along three dimensions:

- **Channel width:** Each of the four backbone stages (P2–P5) has an independently selectable channel count. The candidate sets grow with the stage depth to reflect the natural widening of feature hierarchies in modern detectors.
- **Block depth:** The number of repeated blocks within each stage is searchable, controlling the representation capacity at each resolution level.
- **Operator type:** This dimension subsumes both the *feature-extraction* module in each stage and the *downsampling* operator between stages. Feature-extraction candidates include C2f [12], C3k2 and C2PSA [11], and C2fCIB [22]—the core building blocks employed across YOLOv8 through YOLO12—spanning a spectrum from lightweight convolutions (C2f) to efficient re-parameterized blocks (C3k2, C2fCIB) and attention-augmented modules (C2PSA). Downsampling candidates include standard Conv and the stride-channel-decoupled SCDown [22].

  In the backbone, the operator palette widens progressively: P2/P3 stages choose between {C2f, C3k2}, P4 adds C2fCIB, and P5 further includes C2PSA, reflecting the increasing complexity budget at lower resolutions. In the neck, channel width and depth are fixed to avoid excessive growth of the search space; only the operator type is searchable. Feature blocks N1–N4 share a single global choice from {C2f, C3k2, C2fCIB}, and the two downsample blocks D1–D2 share a single choice from {Conv, SCDown}. Upsample layers use fixed `nn.Upsample` throughout the YOLO family and are not searchable.

Tab. 1 shows the full search space specification. The combinatorial product of all dimensions yields an architecture space on the order of millions of unique configurations, providing a rich landscape for NAS algorithm evaluation.

Architecture Sampling. To build a diverse and representative ground-truth database, we employ three complementary sampling strategies over the search space: (i) *Random sampling* (200 architectures) provides a uniform baseline coverage; (ii) *Stratified sampling* (400 architectures) bins candidates by Parameters and draws uniformly within each stratum, ensuring balanced representation across model scales; (iii) *Latin Hypercube Sampling* [18] (400 architectures) maximizes coverage across the high-dimensional discrete space by applying stratified sampling independently along each search dimension. The three strategies are complementary: random sampling covers the space broadly, stratified sampling prevents under-representation of light or

[1]P4 channels must be at least P3 to avoid degenerate configurations.

Table 1. **Search space of YOLO-NAS-Bench.** Channel, depth, and operator choices for each stage.

Dimension	Stage	Candidates	#
Channel	P2	128, 192, 256	3
	P3	256, 320, 384, 448, 512	5
	P4[1]	384, 512, 640, 768	4
	P5	768, 1024, 1280	3
Depth	P2	1, 2	2
	P3	2, 3, 4	3
	P4	2, 3, 4	3
	P5	2, 3	2
Operator (backbone)	P2/P3	C2f, C3k2	2
	P4	C2f, C3k2, C2fCIB	3
	P5	C2f, C3k2, C2fCIB, C2PSA	4
Operator (neck)	N1–N4	C2f, C3k2, C2fCIB	3
	D1–D2	Conv, SCDown	2

heavy architectures, and LHS provides near-optimal space-filling properties.

All 1,000 sampled architectures are trained from scratch on COCO-mini under an identical training protocol (detailed in Sec. 4.1). COCO-mini is a class- and size-stratified 10% subset of COCO that preserves the original category and bounding-box size distributions. The resulting database constitutes the foundation of YOLO-NAS-Bench.

Architecture Encoding and Surrogate Predictor. Each architecture configuration is converted into a 24-dimensional feature vector. Channel widths and block depths are represented as scalar values, while operator choices are one-hot encoded, allowing the tree-based predictor to split on each operator independently without imposing artificial ordering.

We train a LightGBM [13] gradient-boosted decision tree as the surrogate predictor, regressing from the 24-dim encoding to $\text{mAP}_{50\text{-}95}$. Following NAS-Bench-301 [25], we adopt two evaluation metrics:

- **Coefficient of determination** (R^2): measures overall regression quality.
- **Sparse Kendall Tau** (sKT): the Kendall τ rank correlation computed after rounding predictions to 0.1% precision, which discounts ranking changes due to negligible prediction noise. Formally, $\text{sKT} = \tau\big(\mathbf{y},\ \lfloor \hat{\mathbf{y}} \rceil_{0.001}\big)$.

3.2. Self-Evolving Predictor

A predictor trained on uniformly sampled architectures may under-represent the high-performance frontier, where ranking accuracy is most critical for NAS. To address this distribution mismatch, we propose a *Self-Evolving Mechanism*. As shown in Fig. 3, it iteratively enriches the training pool with architectures that the current predictor considers promising, thereby sharpening its discrimination in the region that matters.

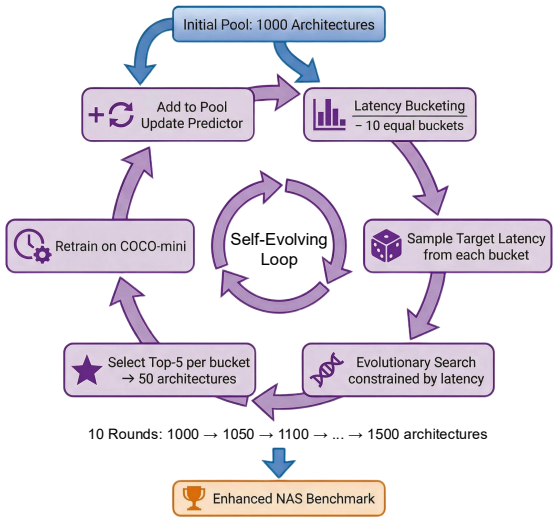

Figure 3. **Self-Evolving Predictor.** Starting from 1,000 architectures, the loop partitions latency into 10 buckets. For each bucket, EA search selects the top 5 architectures using predicted mAP as fitness and real latency as constraint. In each round, these 50 new architectures are trained on COCO-mini, merged into the pool, and the predictor is retrained. After 10 rounds the pool grows to 1,500 architectures, yielding an enhanced benchmark enriched in the high-performance regime most relevant to NAS.

Latency bucketing. The latency range observed across all 1,000 initial architectures is evenly partitioned into 10 buckets. This stratification ensures that the self-evolving loop discovers high-performing architectures at every latency operating point, not only at a single scale.

Evolutionary expansion (single round). Within each latency bucket, a target latency is sampled uniformly. An evolutionary algorithm (EA) [19] is then run with predicted mAP as the fitness function and real measured latency as the constraint. The EA uses a population of 50 and runs for 100 generations, adopts top 25% as parents, with offspring generated by uniform crossover (50%) and mutation (50%, mutation probability 0.2), and preserves the top 10% as elite. The final top 5 architectures from each bucket are retained on COCO-mini, yielding 50 new candidate architectures per round.

Iterative refinement. Each batch of 50 new architectures is fully trained on COCO-mini under the same protocol, their configuration and ground-truth mAP are recorded, and the pairs are merged into the architecture pool. The predictor is then retrained on the expanded pool. This loop repeats for 10 rounds, growing the pool from 1,000 to 1,500 architectures. Crucially, each round's newly added architectures are biased toward the high-performance regions identified by the current predictor, progressively reducing the gap be-

Table 2. **Unified training configuration** for all architectures in YOLO-NAS-Bench.

Parameter	Value
Epochs	120
Batch size	128
Image size	640×640
Learning rate (lr_0)	0.01
LR schedule	Step (decay at epoch 100)
Mosaic	1.0
MixUp	0.15
Copy-Paste	0.5
Pretrained	No
Workers	16

tween the predictor's training distribution and the distribution of architectures encountered during actual NAS search.

Ensemble prediction. After the final round, 10 LightGBM models are trained on the full 1,500-architecture pool with different random seeds. The ensemble prediction is the arithmetic mean of the 10 models, which reduces variance and further stabilizes ranking quality.

4. Experiments

4.1. Experimental Setup

Dataset. We construct COCO-mini by stratified sampling 10% of COCO [16] images, preserving the original category distribution and bounding-box size ratios. The resulting subset retains 80 classes and follows the standard train2017 / val2017 split. All architectures—both the benchmark pool and YOLO baselines—are trained and evaluated on this identical dataset, ensuring fair comparison.

Training protocol. Every architecture is trained from scratch under a unified configuration summarized in Tab. 2, including Mosaic [2], MixUp [26], and Copy-Paste [9] augmentation. The final $mAP_{50\text{-}95}$ is taken from the last epoch.

Latency measurement. All latencies are measured on a single NVIDIA P40 GPU with batch size 1, 640×640 FP32 input, 10 warmup and 50 timed forward passes. The mean inference time is reported in milliseconds.

Predictor training. The surrogate predictor is trained with RMSE as the loss function. We report R^2 and sKT on a held-out 20% validation split.

4.2. Main Results

Predictor Quality. Tab. 3 summarizes the surrogate predictor's accuracy before and after Self-Evolving. The 10-model LightGBM ensemble trained on the initial 1,000 architectures achieves a validation sKT of 0.694. After 10 rounds of Self-Evolving, both R^2 and sKT get improved: R^2 rises from 0.770 to 0.815 (+4.5%), and sKT from 0.694 to 0.752 (+5.8%). An sKT of 0.752 indicates strong ranking consistency, while the R^2 of 0.815 suggests that the predictor captures the majority of variance in architecture performance. These results show that our surrogate predictor is a high-fidelity proxy for the true performance landscape, and confirm that the Self-Evolving mechanism effectively sharpens the predictor's ranking ability. Fig. 4 further visualizes the close agreement between predicted and actual mAP across all 1,500 architectures.

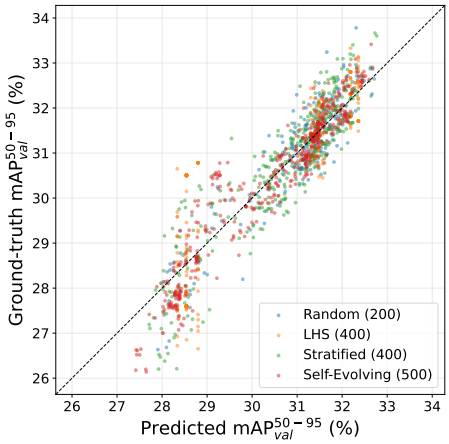

Figure 4. **Predicted vs. ground-truth mAP on the full 1,500-architecture pool.** Each point is an architecture colored by its sampling source. Points cluster closely around the $y{=}x$ diagonal, confirming strong agreement between the ensemble predictor and ground-truth performance.

Table 3. **Predictor quality before and after Self-Evolving.** Metrics are reported on the validation split (20%). Ensemble: 10 homogeneous LightGBM models.

Setting	#Archs	R^2	sKT
Before Self-Evolving	1,000	0.770	0.694
After Self-Evolving	1,500	**0.815**	**0.752**

Predictor-Guided Search Results. To validate that our surrogate predictor can reliably identify top-performing architectures, we conduct an evolutionary architecture search (EA) [19] using the *ensemble predictor's predicted mAP as the fitness function* and real measured latency as the constraint. The EA searches within the same search space defined in Sec. 3.1. Top candidate architectures are then *fully retrained* on COCO-mini under the identical protocol of Tab. 2, producing ground-truth mAP values.

As shown in Tab. 4, the four predictor-discovered architectures Pareto-dominate all official YOLO baselines (v8–v12) across the full latency spectrum, achieving consistently higher mAP at equal or lower latency. The advantage is most pronounced in the small-model regime, where `Arch-D` surpasses YOLO11s by +4.2% mAP at comparable latency, and remains substantial at the large end, where

Table 4. **Predictor-guided EA search results vs. official YOLO baselines** on COCO-mini. All models are trained from scratch under the same protocol. Latency is measured on a single P40 GPU. The top block shows architectures discovered by our predictor-guided EA; subsequent blocks show official baselines.

	Large		Medium-Large		Medium		Small	
Model	mAP (%)	Lat. (ms)	mAP (%)	Lat. (ms)	mAP (%)	Lat. (ms)	mAP (%)	Lat. (ms)
Ours	**33.6**	35.00	**33.4**	27.09	**32.7**	21.00	**31.9**	16.85
YOLO12	32.9	54.13	30.8	42.51	30.4	22.17	27.2	20.40
YOLO11	32.8	37.73	32.1	29.84	30.8	20.03	27.7	15.30
YOLOv10	30.5	36.84	30.1	25.83	29.7	20.01	25.8	15.89
YOLOv9	32.3	53.07	31.6	26.85	31.6	25.59	27.1	19.25
YOLOv8	32.9	33.71	30.8	23.30	29.4	17.21	26.2	10.95

Table 5. **Predictor type comparison** on initial 1,000 architectures (no ensemble, same data split).

Predictor	R^2	**sKT**
LightGBM	**0.768**	0.699
XGBoost	0.758	0.696
NGBoost	0.755	**0.704**
Random Forest	0.744	0.678
MLP	0.053	0.440

Table 6. **Self-Evolving vs. random pool expansion.** Same pool size (1,200); metrics on validation split (20%).

Expansion strategy	**#Archs**	R^2	**sKT**
Initial (no expansion)	1,000	0.770	0.694
Random +200	1,200	0.776	0.701
Self-Evolving +200	1,200	**0.798**	**0.738**

`Arch-A` exceeds YOLO12x in mAP while being 1.5× faster. These results confirm that the predictor not only achieves high ranking correlation (sKT) on held-out data, but also exhibits *strong discriminative power in the top-performance regime*—architectures it deems promising indeed yield superior performance after full retraining.

4.3. Ablation Studies

Predictor type comparison. Tab. 5 compares five predictor types trained on the initial 1,000-architecture pool: LightGBM [13], XGBoost [4], NGBoost [7], Random Forest [3], and MLP. Among the tree-based methods, LightGBM achieves the best balance of R^2 (0.768) and sKT (0.699), followed closely by NGBoost (sKT 0.704) and XGBoost (R^2 0.758). Random Forest lags slightly behind. The MLP baseline performs poorly. These results justify our choice of LightGBM as the base model for the predictor.

Self-Evolving vs. random expansion. To disentangle whether the predictor improvement stems from the Self-Evolving mechanism or merely from increasing the training pool size, we conduct an ablation: we train 200 randomly sampled architectures on COCO-mini, add them to the initial 1,000 architectures, and retrain the ensemble predictor. For fair comparison, we also truncate Self-Evolving when the pool reaches 1,200 architectures. Tab. 6 reports the results. Random expansion yields $R^2 = 0.776$ and sKT$= 0.701$, while Self-Evolving achieves $R^2 = 0.798$ and sKT$= 0.738$. The clear gap (+0.022 R^2, +0.037 sKT) confirms that the gain is attributable to the targeted enrichment of high-performance architectures by the Self-Evolving loop, rather than to the increase in data size alone.

4.4. Limitations and future work.

The current benchmark is built on COCO-mini (10% of COCO) and measures latency on a single GPU (NVIDIA P40). To ensure the authenticity of latency measurements across different platforms, we do not model a latency predictor in the same way as mAP; thus, when using this benchmark for NAS, one still needs to measure the latency of each architecture empirically. But this cost is acceptable compared to the error that a latency predictor would introduce on unseen hardware. Extending YOLO-NAS-Bench to the full COCO dataset, diverse hardware platforms (edge GPUs, mobile NPUs), and additional tasks such as instance segmentation and pose estimation are natural next steps.

5. Conclusion

We present YOLO-NAS-Bench, the first surrogate benchmark tailored to YOLO-style detectors. By designing a search space that spans channel width, block depth, and operator type across backbone and neck, and by training 1,000 architectures sampled by different strategies on COCO-mini, we establish an initial performance database. Based on this, our *Self-Evolving Predictor* iteratively enriches the database with high-value architectures, raising the ensemble LightGBM predictor's sKT from 0.694 to 0.752. Architectures discovered by predictor-guided evolutionary search surpass all official YOLOv8–YOLO12 baselines at comparable latency on COCO-mini, confirming the predictor's practical utility.

References

[1] Shay Aharon, Louis-Dupont, Ofri Masad, Kate Yurkova, Lotem Fridman, Lkdci, Eugene Khvedchenya, Ran Rubin, Natan Bagrov, Borys Tymchenko, Tomer Keren, Alexander Zhilko, and Eran-Deci. Super-gradients, 2021. 1, 2

[2] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. 5

[3] Leo Breiman. Random forests. *Machine Learning*, 45(1): 5–32, 2001. 6

[4] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016. 6

[5] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. DetNAS: Backbone search for object detection. 32:6642–6652, 2019. 1, 2

[6] Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020. 1, 2

[7] Tony Duan, Anand Avati, Daisy Yi Ding, Khanh K Thai, Sanjay Basu, Andrew Ng, and Alejandro Schuler. NGBoost: Natural gradient boosting for probabilistic prediction. In *International Conference on Machine Learning*, pages 2690–2700, 2020. 6

[8] 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*, pages 7036–7045, 2019. 2

[9] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D Cubuk, Quoc V Le, and Barret Zoph. Simple copy-paste is a strong data augmentation method for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2918–2928, 2021. 5

[10] Chenhan Jiang, Hang Xu, Wei Zhang, Xiaodan Liang, and Zhenguo Li. SP-NAS: Serial-to-parallel backbone search for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11863–11872, 2020. 2

[11] Glenn Jocher and Jing Qiu. Ultralytics yolo11, 2024. 2, 3

[12] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8, 2023. 2, 3

[13] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. 30:3149–3157, 2017. 4, 6

[14] Ivan Lazarevich, Matteo Grimaldi, Ravish Kumar, Saptarshi Mitra, Shahrukh Khan, and Sudhakar Sah. YOLOBench: Benchmarking efficient object detectors on embedded systems. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1169–1178, 2023. 2

[15] Tingting Liang, Yongtao Wang, Zhi Tang, Guosheng Hu, and Haibin Ling. OPANAS: One-shot path aggregation network architecture search for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10195–10203, 2021. 1, 2

[16] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision*, pages 740–755, 2014. 1, 5

[17] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. 1

[18] Michael D McKay, Richard J Beckman, and William J Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61, 2000. 3

[19] 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*, pages 4780–4789, 2019. 1, 2, 4, 5

[20] Yunjie Tian, Qixiang Ye, and David Doermann. YOLOv12: Attention-centric real-time object detectors. In *Advances in Neural Information Processing Systems*, 2025. 2

[21] Renbo Tu, Nicholas Roberts, Misha Khodak, Junhong Shen, Frederic Sala, and Ameet Talwalkar. NAS-Bench-360: Benchmarking neural architecture search on diverse tasks. 35:12380–12394, 2022. 2

[22] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. YOLOv10: Real-time end-to-end object detection. 37:107984–108011, 2024. 2, 3

[23] 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*, pages 1–21, 2024. 2

[24] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114, 2019. 1, 2

[25] Arber Zela, Julien Niklas Siems, Lucas Zimmer, Jovita Lukasik, Margret Keuper, and Frank Hutter. Surrogate NAS benchmarks: Going beyond the limited search spaces of tabular NAS benchmarks. In *International Conference on Learning Representations*, 2022. 1, 2, 4

[26] Hongyi Zhang, Moustapha Cissé, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018. 5

[27] 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*, pages 16965–16974, 2024. 2

[28] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. 1