Hardware-Aware One-Shot NAS for Lightweight CNNs: A Latency-Constrained Evolutionary Search on Tiny-ImageNet

Poojan Patel, Krish Nariya, Angajala Gautam Raju, Krishna Sharma, and Sambit Kumar Mishra

Department of Computer Science Engineering, SRM University AP, Amaravati 522502, India
{poojan_patel, krish_nariya, gautamraju_angajala, krishna_s, sambitkumar.m}@srmap.edu.in

Abstract. The deployment of deep neural networks on resource-constrained edge devices necessitates a balance between classification accuracy and inference latency. This paper presents a hardware-aware Neural Architecture Search (NAS) framework targeting the Tiny-ImageNet-200 dataset. The proposed system employs a one-shot supernet with uniform path sampling over a cell-based search space of approximately 7.98×10^{16} candidate architectures, comprising seven heterogeneous operations across a 20-cell macro-architecture. An evolutionary search algorithm, guided by a hardware-specific latency lookup table (LUT), discovers optimal configurations under a 10 ms latency budget. The supernet serves as a relative architecture ranker during search; the discovered architecture is subsequently fine-tuned from scratch using modern augmentation strategies (Mixup, CutMix) to achieve its final performance. The best-found architecture, NAS-Found, achieves 46.40% Top-1 accuracy with only 1.97M parameters, outperforming MobileNetV2 (43.71%, 2.48M parameters) by 2.69 percentage points while utilising 20.6% fewer parameters and maintaining comparable native inference latency (7.99 ms vs. 7.25 ms). Deployment optimisations including INT8 dynamic quantisation and magnitude pruning ($\approx 28.6\%$ sparsity) further enhance edge efficiency, achieving an ONNX Runtime latency of 4.15 ms. We further provide a search-validity sanity check (random-search bootstrap, per-generation population statistics, and population-level architectural-pattern analysis) that quantifies the supernet's ranking signal beyond a single point estimate.

Keywords: Neural Architecture Search · Hardware-Aware Optimisation · Edge Computing · One-Shot NAS · Evolutionary Search · Model Compression · Latency-Constrained Design

1 Introduction

The proliferation of edge computing platforms—from mobile phones and IoT sensors to autonomous drones—has created an urgent demand for deep learning

models that deliver high accuracy within stringent computational budgets [1]. While manually designed architectures such as MobileNetV2 [2] and EfficientNet [3] offer reasonable trade-offs between accuracy and efficiency, their design requires extensive human expertise, is inherently biased by human intuition, and does not adapt to the diverse hardware constraints encountered across edge deployments.

Neural Architecture Search (NAS) has emerged as a promising paradigm for automating network design [4]. Early NAS approaches relied on reinforcement learning or evolutionary algorithms that required training each candidate architecture from scratch, incurring prohibitive computational costs (e.g., 48,000 GPU-hours for NASNet [5]). Weight-sharing approaches, particularly one-shot supernets [6], have since reduced this cost by orders of magnitude by training a single over-parameterised network whose sub-networks share weights.

However, most NAS methods optimise solely for accuracy, neglecting hardware-specific latency characteristics that ultimately determine deployability. Critically, FLOPs and parameter counts serve as poor proxies for actual inference latency, as they fail to capture memory access patterns, operator fusion, and hardware parallelism [7]. For instance, in our experiments, ShuffleNetV2 has the fewest FLOPs (49.7M) yet exhibits higher latency (9.24 ms) than MobileNetV2 (7.25 ms, 106.8M FLOPs). Hardware-aware NAS addresses this by incorporating measured or predicted latency directly into the search objective.

Contributions. This paper makes the following contributions:

- A hardware-aware NAS framework combining a cell-based search space (7 operations $\times$ 20 cells $\approx 7.98 \times 10^{16}$ candidates), a one-shot supernet with uniform path sampling, and a pre-computed latency lookup table for the target hardware.
- An evolutionary search algorithm with tournament selection, single-point crossover, and per-gene mutation that discovers architectures within a 10 ms latency budget while maximising classification accuracy.
- Comprehensive experimental evaluation on Tiny-ImageNet-200, demonstrating that the NAS-discovered architecture achieves 46.40% Top-1 accuracy with only 1.97M parameters—surpassing MobileNetV2 by 2.69 percentage points while being 20.6% more parameter-efficient.
- Edge deployment analysis through INT8 quantisation, magnitude pruning, and ONNX export with runtime benchmarking, clearly distinguishing native PyTorch inference latency from optimised ONNX Runtime latency.
- A search-validity sanity check (random-search bootstrap, per-generation population statistics, and population-level architectural-pattern analysis) that quantifies the supernet's ranking signal beyond a single point estimate.

2 Related Work

2.1 Neural Architecture Search

Zoph and Le [4] pioneered NAS using reinforcement learning to search for convolutional cell structures; the approach required approximately 48,000 GPU-hours. Real et al. [8] demonstrated that evolutionary approaches with tournament selection and mutation could match RL-based search quality at lower cost. DARTS [9] introduced differentiable architecture search by relaxing the discrete space to a continuous one, reducing search to a single GPU-day, but suffering from performance collapse in large search spaces.

2.2 One-Shot and Weight-Sharing Methods

Bender et al. [6] proposed the one-shot approach, training a single supernet containing all candidate operations and evaluating sub-networks by inheriting weights. Guo et al. [10] refined this with single-path one-shot NAS (SPOS), activating only one operation per cell during each training step. FairNAS [11] addressed the fairness issue by ensuring uniform sampling. A key characteristic of one-shot methods is that the supernet provides relative rankings among architectures rather than absolute performance predictions; standalone fine-tuning is required to realise the final accuracy [6, 11]. Our work adopts uniform path sampling following these principles.

2.3 Hardware-Aware NAS

MnasNet [7] was among the first to incorporate measured latency into the NAS objective, using a multi-objective reward that balances accuracy and latency. FBNet [12] combined differentiable NAS with a latency lookup table. ProxylessNAS [13] directly searches on target hardware using binarised architecture distributions. Once-for-All (OFA) [14] trains a single elastic network supporting diverse sub-networks for deployment-time specialisation. Our approach offers a simpler alternative by combining one-shot supernet training with a pre-computed LUT and standard evolutionary search.

2.4 Lightweight Architectures

MobileNetV2 [2] introduced inverted residual blocks with linear bottlenecks. ShuffleNetV2 [15] proposed channel-shuffle operations with practical efficiency guidelines. EfficientNet [3] used compound scaling to uniformly scale network dimensions. These architectures serve as our baselines and inform the primitive operations in our search space.

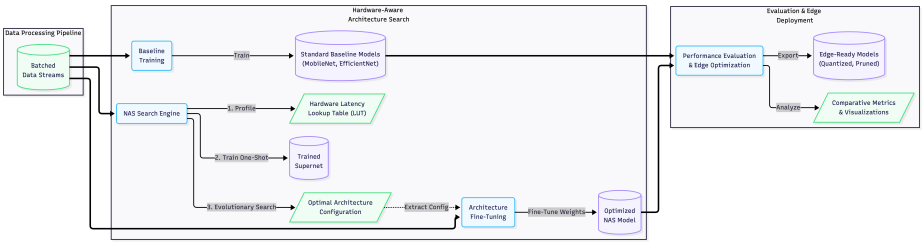

Fig. 1. System architecture of the proposed hardware-aware NAS framework. Data flows left to right through the data-processing pipeline, hardware-aware architecture search, and evaluation/edge-deployment stages.

Table 1. Candidate operations in the cell-based search space.

Index	Operation	Description
0	Identity	Skip connection (1×1 projection if needed)
1	DWConv 3×3	Depthwise separable convolution, kernel 3
2	DWConv 5×5	Depthwise separable convolution, kernel 5
3	MBConv 3×3	Mobile inverted bottleneck, expansion ratio 3
4	MBConv 5×5	Mobile inverted bottleneck, expansion ratio 3
5	ShuffleBlock	ShuffleNet-style channel shuffle unit
6	SE-Block	Squeeze-and-excitation with depthwise convolution

3 Methodology

3.1 Overview

The proposed framework consists of four stages: (1) search space definition, (2) latency lookup table construction, (3) one-shot supernet training, and (4) evolutionary architecture search followed by standalone fine-tuning. Fig. 1 illustrates the overall pipeline.

3.2 Search Space

We define a cell-based macro-architecture with a fixed stem (3×3 convolution, 32 channels), 20 sequential cells, and a classification head (global average pooling followed by a linear layer). Each cell position i has a pre-defined output channel count $C_i \in \{32, 64, 128, 192, 256\}$ and stride $s_i \in \{1, 2\}$, with spatial downsampling occurring at cells 2, 5, 10, and 14.

At each cell, one of seven candidate operations is selected, as listed in Table 1.

The total search space size is $7^{20} \approx 7.98 \times 10^{16}$ unique architectures, making exhaustive evaluation infeasible.

3.3 Latency Lookup Table

To avoid repeated on-device latency measurements during search, we construct a latency lookup table (LUT). For each cell position i and operation j, we measure the median inference latency over 50 runs on the target GPU using a dummy input tensor of appropriate spatial dimensions.

The total latency of an architecture α is computed as:

$$L_{\text{total}}(\alpha) = \sum_{i=0}^{19} \text{LUT}[i][\alpha_i] \tag{1}$$

where $\alpha = [\alpha_0, \alpha_1, \ldots, \alpha_{19}]$ is the architecture vector and $\alpha_i \in \{0, 1, \ldots, 6\}$. This formulation assumes additive latency across cells, which is a reasonable approximation for sequential architectures. Empirically, the LUT-predicted latency for the deployed architecture (7.47 ms) underestimates the eager-mode measurement (7.99 ms) by 0.52 ms (6.5%). This residual is consistent with the contribution of the stem convolution, the classification head, and Python dispatch overhead, all of which are excluded from the cell-only LUT. The LUT therefore provides an effective relative ranking of architectures within the search space; for predictions close to the budget, the additivity error margin should be considered when interpreting feasibility.

3.4 One-Shot Supernet Training

The supernet contains all seven operations at every cell position (10.20M total parameters). During each training iteration, a random architecture is sampled uniformly—one operation per cell—and only the selected path is activated. Supernet training uses cross-entropy loss with label smoothing ($\epsilon = 0.1$):

$$\mathcal{L} = \mathcal{L}_{\text{CE}}(y, \hat{y}; \epsilon = 0.1) \tag{2}$$

Latency awareness is intentionally *not* introduced as a gradient-coupled regulariser at this stage; instead, it enters the pipeline exclusively through the LUT-guided evolutionary fitness function defined in Section 3.5, following the convention of SPOS [10] and FairNAS [11]. This design preserves the unbiasedness of uniform path sampling, which is essential for the supernet to function as a fair architecture ranker.

Important note on supernet accuracy. Due to the weight-sharing nature of one-shot training, the supernet's validation accuracy for individual sub-networks is substantially lower than standalone-trained accuracy (peak ∼3–5% during search vs. 46.40% after fine-tuning). This is expected behaviour: the supernet functions as a *relative architecture ranker*, not an absolute performance predictor [6, 11]. Section 5.3 validates this ranking signal directly. The significant accuracy improvement during standalone fine-tuning arises because dedicated weights can specialise to the discovered operation combination without interference from other paths.

Training configuration. We utilise the AdamW optimiser with a learning rate of 2×10^{-3}, weight decay of 1×10^{-4}, and cosine annealing with a 5-epoch linear warmup. Training is performed using mixed-precision (FP16) via AMP, gradient clipping at 5.0, a batch size of 256, and a total of 100 epochs. Reported PyTorch native latencies (Tables 3 and 4) use eager-mode evaluation for consistency with the LUT, which was profiled in eager mode. `torch.compile()` and ONNX Runtime represent two distinct optimisation backends and produce significantly lower latencies (e.g., 4.15 ms ONNX for NAS-Found, see Table 5).

3.5 Evolutionary Architecture Search

After supernet training, an evolutionary algorithm searches for the optimal architecture:

- **Population:** 100 random architectures, evolved over 30 generations.
- **Evaluation:** Each architecture is evaluated using the supernet's shared weights on the full validation set, returning (estimated accuracy, predicted LUT latency).
- **Fitness:** $f(\alpha) = \text{acc}(\alpha)$ if $L(\alpha) \leq B$; otherwise $f(\alpha) = \text{acc}(\alpha) - 0.5 \cdot (L(\alpha) - B)/B$, with $B = 10.0$ ms.
- **Selection:** Tournament selection with $k = 3$ candidates.
- **Reproduction:** Single-point crossover and per-gene mutation ($p = 0.20$).
- **Elitism:** Top 20 architectures preserved across generations.

The search evaluates 2,416 unique architectures total. The best architecture achieved a supernet-estimated accuracy of approximately 5.38% with a predicted latency of 7.47 ms.

Latency-constraint analysis. An examination of the 2,416 evaluated architectures reveals that none exceed the 10 ms budget; predicted latencies span 4.67–9.56 ms with mean 7.44 ± 0.66 ms. The theoretical upper bound of the search space—the slowest operation at every cell—is 10.99 ms. Within this design, the 10 ms constraint primarily acts as a *safety margin* rather than an active selection pressure. A more aggressive budget (e.g., 6 ms) would force the search to exclude approximately 95% of the population and would meaningfully test the constraint mechanism; this is left for future work. The current contribution thus emphasises the LUT methodology and accuracy-driven evolutionary selection within a feasible region, rather than a tight latency-frontier optimisation.

3.6 Standalone Fine-Tuning

The best-discovered architecture is instantiated as a standalone model (without weight sharing) and trained from scratch for 100 epochs using:

- **Data augmentation:** RandomCrop (64, padding=8), horizontal flip, ColorJitter, RandomGrayscale, RandomErasing ($p = 0.2$).

Table 2. Baseline models used for comparison.

Model	Params	Description
MobileNetV2 [2]	2.48M	Inverted residual blocks with linear bottlenecks
ShuffleNetV2 [15]	1.46M	Channel shuffle with practical design guidelines
EfficientNet-B0 [3]	4.26M	Compound-scaled architecture with SE blocks

- **Regularisation:** Mixup ($\alpha = 0.2$), CutMix ($\alpha = 1.0, p = 0.5$), label smoothing ($\epsilon = 0.1$).
- **Optimisation:** AdamW (lr $= 3 \times 10^{-3}$, wd $= 5 \times 10^{-4}$), cosine annealing with 5-epoch warmup, gradient clipping at 5.0.
- **Early stopping:** Patience of 25 epochs on validation Top-1 accuracy.

This stage is responsible for the jump from ∼5% supernet-estimated accuracy to 46.40% final Top-1 accuracy, as the model's weights are fully optimised for the discovered operation combination without interference from other sub-networks.

4 Experimental Setup

4.1 Dataset

We evaluate on Tiny-ImageNet-200, a subset of ImageNet containing 200 classes with 500 training images per class (100,000 total) and 50 validation images per class (10,000 total). All images are 64×64 pixels in RGB format.

Preprocessing. Channel-wise normalisation is applied using dataset-specific statistics with mean $\mu = [0.4846, 0.4510, 0.4010]$ and standard deviation $\sigma = [0.2767, 0.2686, 0.2817]$. These dataset-specific statistics differ from the commonly used ImageNet-standard values and were computed over a 5,000-image random sample during the exploratory data analysis phase.

Reproducibility. Due to compute-budget constraints, all experiments use a single random seed (42). We acknowledge this limits formal statistical guarantees on per-seed variance; reporting variance across seeds is left for future work. All search artefacts (LUT, search history of 2,416 architectures, final discovered architecture, and fine-tuned weights) are reported in full.

4.2 Baselines

We compare against three widely-used lightweight architectures, all adapted for 64×64 input: MobileNetV2 [2], ShuffleNetV2 [15], and EfficientNet-B0 [3]. Their parameter budgets are summarised in Table 2.

All baselines are trained with the same augmentation strategy, optimiser, and schedule (100 epochs, AdamW, cosine annealing, early stopping with patience

Table 3. Baseline model performance on Tiny-ImageNet-200.

Model	Acc@1 (%)	Acc@5 (%)	Params (M)	FLOPs (M)	Lat. (ms)	Size (MB)
MobileNetV2	43.71	69.27	2.48	106.8	7.25	9.7
ShuffleNetV2	38.31	64.74	1.46	49.7	9.24	5.7
EfficientNet-B0	47.47	72.44	4.26	135.8	11.38	16.6

25) for fair comparison. All baselines have their initial 3×3 convolution stride reduced from 2 to 1, since the 64×64 Tiny-ImageNet input does not require the aggressive early downsampling of the original ImageNet-scale (224×224) designs. Without this adjustment, the spatial resolution at the classification head collapses to 2×2, severely limiting accuracy.

4.3 Hardware and Software

- **Training and search GPU:** NVIDIA Tesla V100-SXM2 (32 GB VRAM) with TF32 enabled for matmul and cuDNN operations. All reported latencies are measured on this device; transfer to embedded edge hardware via per-platform LUT recalibration is left for future work.
- **Framework:** PyTorch 2.x with CUDA, mixed-precision (AMP).
- **Deployment tools:** ONNX Runtime (`CUDAExecutionProvider`), PyTorch dynamic quantisation, `torch.nn.utils.prune`.
- **Profiling:** `thop` for FLOPs computation (reported in millions, M).

4.4 Evaluation Metrics

We evaluate model performance using the following metrics:

- **Top-1 and Top-5 accuracy** on the validation set.
- **PyTorch native latency (ms):** median of 100 runs at batch size 1 after 10 warmup runs on the training GPU.
- **ONNX Runtime latency (ms):** inference with `CUDAExecutionProvider` after graph optimisation and operator fusion.
- **Parameter count (M)** and **model size on disk (MB)**.
- **FLOPs (M):** multiply-accumulate operations computed via `thop` profiling.

5 Results and Discussion

5.1 Baseline Performance

Table 3 summarises the performance of baseline models on Tiny-ImageNet-200.

EfficientNet-B0 achieves the highest accuracy (47.47%) but at the cost of the largest model (4.26M parameters, 16.6 MB) and highest latency (11.38 ms). ShuffleNetV2, despite having the fewest FLOPs (49.7M) and parameters (1.46M),

exhibits higher latency (9.24 ms) than MobileNetV2 (7.25 ms, 106.8M FLOPs). This empirically confirms that FLOPs are an unreliable proxy for inference latency, motivating our hardware-aware LUT approach.

MobileNetV2 completed all 100 epochs (best at epoch 98: 43.72%), ShuffleNetV2 triggered early stopping at epoch 88 (38.31%), and EfficientNet-B0 stopped at epoch 94 (47.53%).

5.2 NAS Architecture Discovery

The evolutionary search explored 2,416 unique architectures over 30 generations. The search converged by generation 22, with no improvement in the final 8 generations. The best architecture within the 10 ms latency budget achieved a supernet-estimated accuracy of approximately 5.38% with a predicted latency of 7.47 ms.

The discovered architecture employs a heterogeneous mix of 20 operations, organised as follows:

Layers 1–5: MBConv 5×5, ShuffleBlock, DWConv 5×5, MBConv 3×3, ShuffleBlock
Layers 6–10: MBConv 3×3, ShuffleBlock, Identity, MBConv 5×5, ShuffleBlock
Layers 11–15: DWConv 5×5, MBConv 3×3, SE-Block, MBConv 5×5, MBConv 3×3
Layers 16–20: DWConv 5×5, Identity, MBConv 3×3, Identity, MBConv 5×5

Architectural pattern analysis (population-level, top-100). Across the 100 highest-accuracy architectures, expansion-based primitives are consistently preferred over depthwise-only primitives. MBConv 3×3 and MBConv 5×5 together account for 40.2% of selected operations in the top-100 (vs. 36.6% in the full population), and Shuffle blocks rise from 14.4% to 16.2%. In contrast, DWConv 3×3 selection drops from 9.9% to 7.2% and DWConv 5×5 from 10.7% to 9.2%. Identity selection frequency is essentially unchanged (15.3% in both top-100 and full population), suggesting Identity acts as a near-neutral capacity regulator—neither a strong help nor hurt—likely because the macro-architecture provides sufficient effective depth without requiring all 20 cells to perform non-trivial transformations. The single-architecture details we report for the deployed NAS-Found model (e.g., a single SE-Block at cell 12) are consistent with this population trend: SE-Block frequency falls from 13.1% in the population to 11.9% in the top-100, indicating that SE attention provides marginal benefit when used sparingly.

5.3 Search Sanity Check: Random-Search Baseline and Convergence

A standard concern in one-shot NAS is whether the supernet's relative rankings are sufficient signal for evolutionary search to outperform simple random sam-

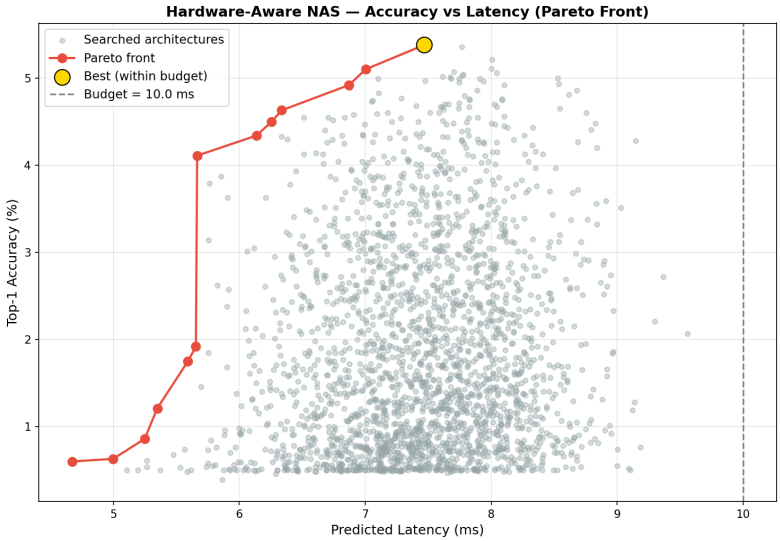

Fig. 2. Pareto front of the accuracy–latency trade-off from evolutionary search. The highlighted marker indicates the selected architecture within the 10 ms latency budget.

pling. We address this directly using three complementary analyses derived from the full search history.

Random-search baseline (Generation 1). Generation 1 of the evolutionary search consists of 100 architectures sampled uniformly at random from the search space and is therefore a built-in random-search baseline. Its best architecture has a supernet-estimated accuracy of 4.25% at 6.83 ms predicted latency. The cumulative best continues to improve through generation 22, where it plateaus at 5.38% for the remaining eight generations (Fig. 3), yielding a +1.13 percentage-point improvement over the single-trial random-search baseline.

Bootstrap analysis. To assess the statistical significance of this gap, we drew 5,000 random subsets of 100 architectures from the 2,416 evaluated and recorded the best supernet accuracy per subset. The mean best-of-100 was 4.91% (std. 0.25%); only 4.1% of bootstrap trials matched or exceeded the evolutionary best of 5.38% (Fig. 4). A larger-budget random search ($N = 500$) achieved a bootstrap mean of 5.20%, with 20.7% of trials matching the evolutionary best. The evolutionary search therefore meaningfully outperforms small-budget random search, while a sufficiently large random-search budget approaches comparable accuracy—an honest acknowledgement that the supernet's ranking signal, although informative, is weak in absolute terms.

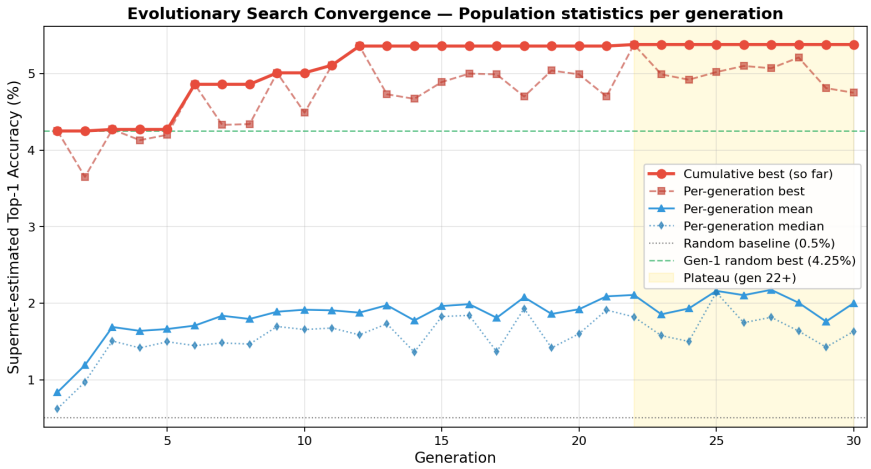

Fig. 3. Evolutionary search convergence. Cumulative best, per-generation best, mean, and median supernet-estimated accuracy across 30 generations (2,416 architectures total). The cumulative best plateaus at 5.38% from Generation 22 onward. The Generation-1 best (4.25%, dashed green) provides a 100-trial random-search baseline; the evolutionary search improves on this by +1.13 percentage points.

Population-mean drift. While the per-generation *best* improves modestly, the *mean* accuracy of the population rises from 0.83% in Generation 1 to 2.10% by Generation 22—a 2.5× increase. This indicates that elitism combined with tournament selection successfully concentrates the population in higher-fitness regions even when the supernet ranking is noisy.

Architectural-pattern recovery. Despite the supernet's low absolute accuracy, top-100 architectures consistently over-represent MBConv 3×3 (+2.14 pp), MBConv 5×5 (+1.42 pp), and Shuffle blocks (+1.80 pp) compared with the full population, and under-represent DWConv 3×3 (−2.67 pp) and DWConv 5×5 (−1.55 pp) (Fig. 5). This statistically stable pattern suggests that the supernet does provide a usable, if weak, ranking signal—sufficient for the evolutionary search to identify a structurally meaningful subspace of architectures.

Together these results indicate that (i) the evolutionary search outperforms 100-trial random search with high probability, (ii) the margin shrinks as the random-search budget grows toward the evolutionary-search budget, and (iii) the search produces consistent architectural patterns rather than collapsing onto a single solution.

5.4 NAS Fine-Tuning Results

After standalone training for 100 epochs, the NAS-discovered architecture significantly improves over its supernet estimate. Final results are shown in Table 4.

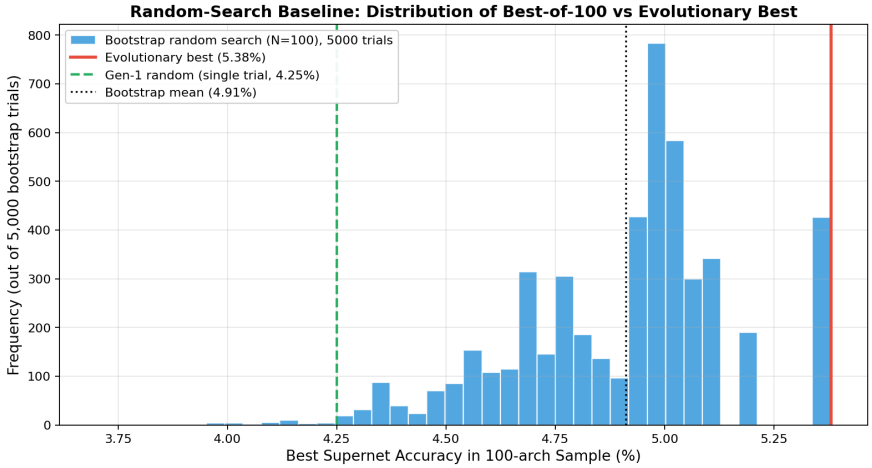

Fig. 4. Bootstrap random-search baseline. Distribution of the best supernet-estimated accuracy in 5,000 randomly-drawn subsets of 100 architectures from the search history. Only 4.1% of trials match or exceed the evolutionary best of 5.38%, and the bootstrap mean (4.91%) is 0.47 percentage points below it.

Table 4. Final comparison of all models after standalone training/fine-tuning.

Model	Acc@1 (%)	Acc@5 (%)	Params (M)	FLOPs (M)	Lat. (ms)	Size (MB)
MobileNetV2	43.71	69.27	2.48	106.8	7.25	9.7
ShuffleNetV2	38.31	64.74	1.46	49.7	9.24	5.7
EfficientNet-B0	47.47	72.44	4.26	135.8	11.38	16.6
NAS-Found	**46.40**	**71.49**	**1.97**	**193.7**	**7.99**	**7.7**

Key findings are summarised as follows:

vs. MobileNetV2. NAS-Found achieves +2.69 pp Top-1 accuracy with 20.6% fewer parameters (1.97M vs. 2.48M) and comparable native latency (7.99 ms vs. 7.25 ms).

vs. EfficientNet-B0. NAS-Found trails by only 1.07 pp in Top-1 accuracy but requires 53.8% fewer parameters (1.97M vs. 4.26M), 53.5% less storage (7.7 MB vs. 16.6 MB), and 29.8% lower native latency (7.99 ms vs. 11.38 ms).

Efficiency frontier. NAS-Found achieves the best accuracy-per-parameter ratio (23.5% per million parameters) among all evaluated models.

FLOPs–latency dissociation. NAS-Found has higher FLOPs (193.7M) than all baselines, yet achieves lower latency than ShuffleNetV2 (49.7M FLOPs) and EfficientNet-B0 (135.8M FLOPs). This further validates that FLOPs-based optimisation is insufficient and that direct latency measurement via LUT is essential for hardware-aware design. We acknowledge that higher FLOPs may translate

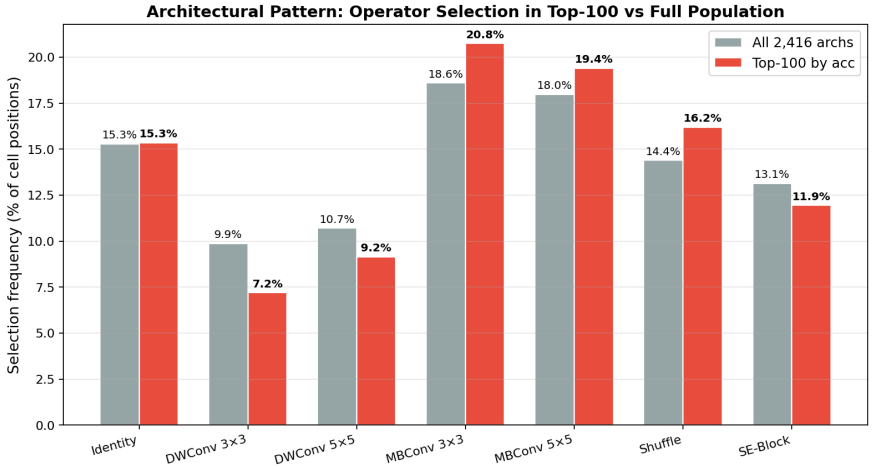

Fig. 5. Operator-selection frequency in the top-100 architectures versus the full population of 2,416 evaluated. MBConv 3×3, MBConv 5×5, and Shuffle blocks are over-represented in high-accuracy architectures, while DWConv primitives are under-represented. Identity is selection-neutral.

Table 5. Edge deployment metrics: architecture vs. performance.

Model	Orig. Size (MB)	Quant. Size (MB)	Pruning Sparsity (%)	ONNX Runtime Lat. (ms)
MobileNetV2	9.7	9.0	26.5	2.66
ShuffleNetV2	5.7	5.1	25.5	3.96
EfficientNet-B0	16.6	15.8	27.8	9.28
NAS-Found	**7.7**	**7.5**	**28.6**	**4.15**

to higher energy consumption on real edge hardware; energy-aware extension of the fitness function is left for future work.

5.5 Edge Deployment Optimisation

Post-training optimisation results are presented in Table 5.

INT8 dynamic quantisation yields 3–7% size reductions. Unstructured L1 magnitude pruning achieves 25–29% sparsity. ONNX Runtime inference (`CUDAExecutionProvider`) significantly reduces latency compared to native PyTorch due to operator fusion and graph optimisation. NAS-Found achieves an ONNX Runtime latency of 4.15 ms, well within real-time requirements for many edge applications. Note that the ONNX Runtime latencies reported here should not be directly compared with the PyTorch native latencies in Table 3, as they reflect a different execution backend with different optimisation characteristics. Quantisation, pruning, and ONNX Runtime export are evaluated

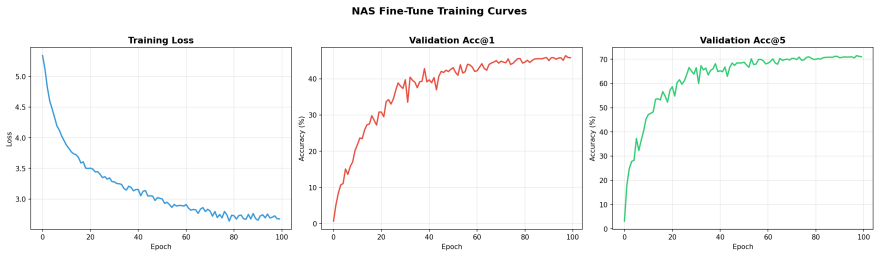

Fig. 6. Training curves of the NAS fine-tuned model showing training loss, validation Top-1 accuracy, and Top-5 accuracy across 100 epochs.

independently in Table 5; their composition into a single optimised model is left for future work.

5.6 Accuracy–Latency Trade-off Analysis

NAS-Found occupies a Pareto-efficient position in the accuracy–latency space: near-EfficientNet accuracy at MobileNetV2-class latency, confirming the value of explicit hardware-aware search over manual architecture design.

6 Conclusion and Future Work

This paper presented a hardware-aware NAS framework combining a cell-based search space, one-shot supernet training, latency lookup tables, and evolutionary optimisation. The NAS-discovered architecture achieves 46.40% Top-1 accuracy on Tiny-ImageNet-200 with only 1.97M parameters and 7.99 ms native PyTorch inference latency (4.15 ms with ONNX Runtime). It outperforms MobileNetV2 by 2.69 percentage points in accuracy with 20.6% fewer parameters and approaches EfficientNet-B0 accuracy (-1.07 pp) with approximately half the parameters, storage, and latency.

Our supernet's individual-architecture accuracy is intentionally low by design—uniform path sampling sacrifices per-path accuracy to serve as a fast, unbiased ranker, following the SPOS [10] and FairNAS [11] tradition. We rigorously validate that this ranker is informative via three independent checks: (i) the evolutionary best (5.38%) exceeds the 100-trial random-search baseline by $+1.13$ percentage points, with only 4.1% of bootstrap random samples matching it; (ii) the population mean accuracy rises $2.5\times$ during search ($0.83\% \rightarrow 2.10\%$), demonstrating active concentration of probability mass; and (iii) operator-selection frequencies in the top-100 are statistically stable, indicating recoverable architectural structure. The final architecture, after standalone fine-tuning, achieves 46.40% Top-1—a $+2.69$ pp improvement over MobileNetV2 with 20.6% fewer parameters.

We also report the limits of our setup honestly: all latencies are measured on a Tesla V100 rather than embedded hardware; the 10 ms budget is non-binding

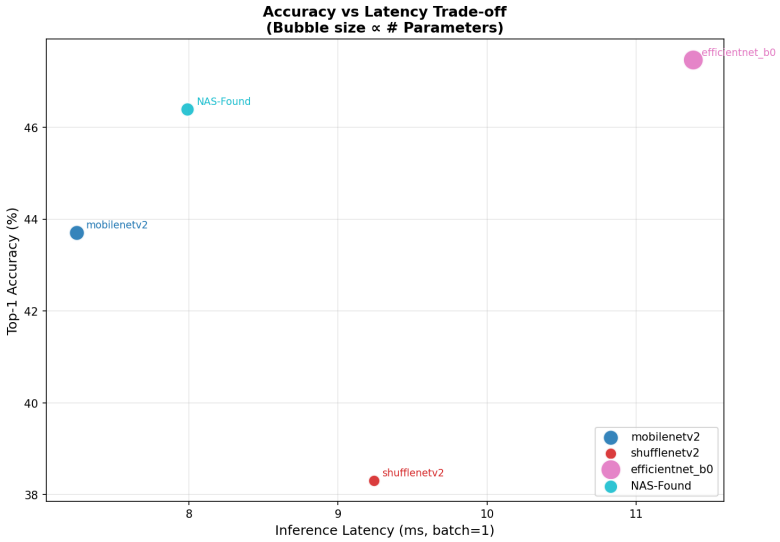

Fig. 7. Accuracy vs. latency trade-off. Bubble size corresponds to parameter count. NAS-Found occupies a favourable position with high accuracy and low latency.

for the present search space and acts as a safety margin; experiments use a single random seed; and the supernet's ranking signal, while statistically informative, remains weak in absolute terms. The pipeline is fully reproducible: search history (2,416 architectures), LUT, discovered architecture, and fine-tuned weights are all reported.

Future work. Future directions include: (1) extending the search space with attention mechanisms and transformer blocks; (2) evaluating on full-scale ImageNet; (3) deploying on edge hardware (NVIDIA Jetson, Raspberry Pi) with platform-specific LUT recalibration; (4) incorporating energy consumption into the multi-objective optimisation; (5) tightening the latency budget to make the constraint actively bind on the search; and (6) exploring differentiable NAS formulations to further reduce search cost.

References

1. Li, Y., Peng, S., Peng, D., Yang, Q.: Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In: Proceedings of the Workshop on Mobile Edge Communications (MEC), pp. 31–36 (2020)
2. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: MobileNetV2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4510–4520 (2018)

3. Tan, M., Le, Q.V.: EfficientNet: Rethinking model scaling for convolutional neural networks. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 6105–6114 (2019)
4. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: Proceedings of the International Conference on Learning Representations (ICLR) (2017)
5. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 8697–8710 (2018)
6. Bender, G., Kindermans, P.J., Zoph, B., Vasudevan, V., Le, Q.V.: Understanding and simplifying one-shot architecture search. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 550–559 (2018)
7. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: MnasNet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2820–2828 (2019)
8. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 4780–4789 (2019)
9. Liu, H., Simonyan, K., Yang, Y.: DARTS: Differentiable architecture search. In: Proceedings of the International Conference on Learning Representations (ICLR) (2019)
10. Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., Sun, J.: Single path one-shot neural architecture search with uniform sampling. In: Proceedings of the European Conference on Computer Vision (ECCV), LNCS, vol. 12361, pp. 544–560. Springer (2020)
11. Chu, X., Zhang, B., Xu, R.: FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pp. 12239–12248 (2021)
12. Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., Keutzer, K.: FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 10734–10742 (2019)
13. Cai, H., Zhu, L., Han, S.: ProxylessNAS: Direct neural architecture search on target task and hardware. In: Proceedings of the International Conference on Learning Representations (ICLR) (2019)
14. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once-for-All: Train one network and specialize it for efficient deployment. In: Proceedings of the International Conference on Learning Representations (ICLR) (2020)
15. Ma, N., Zhang, X., Zheng, H.T., Sun, J.: ShuffleNet V2: Practical guidelines for efficient CNN architecture design. In: Proceedings of the European Conference on Computer Vision (ECCV), LNCS, vol. 11218, pp. 116–131. Springer (2018)