Reinforced Architecture Learning (RACHEL): A Fast-Neural Architecture Search Framework Via Ensembling

Alex Zhekai Zhang[1], **Hui Liu**[2]

Received July 08, 2025
Accepted September 21, 2025
Electronic access September 30, 2025

Neural Architecture Search (NAS) automates model design but often requires prohibitive computation, with some methods needing thousands of GPU hours. This study addresses the critical need for an efficient NAS framework. We hypothesized that ensemble learning combined with an incremental reinforcement learning (RL) approach could discover high-performing architectures at a fraction of the typical computational cost. We developed Reinforced ArCHitEcture Learning (RACHEL), a framework integrating an actor-critic RL agent and a stability-ensuring ensemble safety net. Utilizing modern optimizations, RACHEL demonstrated exceptional efficiency on the Canadian Institute for Advanced Research (CIFAR-10) image dataset. It achieved significantly higher accuracy than the AdaNet baseline on binary tasks in under two hours and competitive accuracy on the full dataset in under eight GPU hours. The framework's effectiveness was further validated with state-of-the-art accuracy among compared methods on Street View House Numbers (SVHN) and highly competitive performance on the Fashion Modified National Institute of Standards and Technology dataset (Fashion-MNIST). Our results show that RACHEL provides competitive performance at a drastically reduced computational cost, making advanced NAS methods more accessible.

Keywords: Neural Architecture Search, Reinforcement Learning, Ensemble Learning, Deep Learning, CIFAR-10.

1 Introduction

Artificial Intelligence (AI) has quickly advanced by enabling computer systems to learn from data, a field known as machine learning. Modern machine learning, especially deep learning, is largely based on structures called neural networks, which are modeled after biological brains. The specific design of a neural network is known as its architecture, and its design defines its capacity to learn and its overall performance on tasks such as object recognition in images. Deep learning using neural networks has achieved state-of-the-art results across numerous domains[1], but designing optimal neural network architectures is still a complex and time-consuming task[2]. The effort required in manual design has created a need for methods that can automate the discovery of high-performing architectures, which is what the field of neural architecture search (NAS) aims to do. By replacing labor-intensive trial and error, NAS offers a pathway to democratize AI, accelerate model creation, and reveal task-specific architectures that improve accuracy, efficiency, and scalability across domains like computer vision and natural language processing[2].

Current NAS approaches can be broadly categorized into three types: reinforcement learning-based methods[3], which train agents to iteratively propose architectures; evolutionary algorithms[4], which optimize architectures through genetic operations like mutation and selection; and gradient-based techniques (Differentiable Architecture Search, (DARTS[5])), which leverage differentiable optimization to optimize architecture parameters and weights. A general framework for NAS is illustrated in Figure 1.

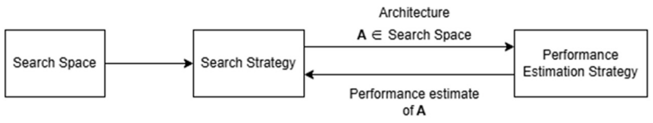

Fig. 1 The General Framework of Neural Architecture Search (NAS). This flowchart illustrates the high-level, cyclical process common to most NAS methods. The framework operates in a loop where a search strategy (e.g., an RL agent or an evolutionary algorithm) selects or generates a neural architecture (A) from a predefined search space of possible architectures. This proposed architecture is then passed to a performance estimation strategy, which trains or approximates the performance of the model on a given task. The resulting performance metric is returned to the search strategy, which uses this feedback to guide the generation of the next architecture[2].

Among various NAS strategies, reinforcement learning (RL) has proven particularly effective4. In RL-based NAS, an RL

[1] *1960 Cate Mesa Road, Cate School, Carpinteria, CA, USA. Email: Alex_Zhang@cate.org*
[2] *Department of Computer Science, Missouri State University, 901 S. National Ave. Springfield, MO, USA Email: HuiLiu@MissouriState.edu*

agent (a controller) learns to generate architectures, receiving feedback based on the performance of the trained architectures (child networks) on a validation dataset. Zoph and Les[3] early work showed that a recurrent neural network policy trained with REINFORCE could discover convolutional cell structures that surpassed human designs on CIFAR10 and ImageNet, albeit at the cost of roughly 22,400 GPUdays, or over 500,000 GPU-hours. Subsequent RL variants aimed to reduce this cost. MetaQNN[6] explored greedy Qlearning, while efficient neural architecture search (ENAS)[7] introduced weight sharing so that child networks inherit parameters from a supergraph, cutting the search time down heavily. Progressive NAS (PNAS)[8] combined RL search with a progressively expanding search space to balance exploration and efficiency.

Ensemble-based NAS has also proven to be very effective in the field. Ensemble-based Knowledge Distillation for NAS (EnNAS)[9] constructs a diverse ensemble of candidate subnets for each training batch. Distillation is then applied to each subnet, allowing the network to achieve competitive accuracies on ImageNet. Neural Ensemble Search for Uncertainty Estimation and Dataset Shift (NES) applies evolutionary NAS concepts to ensemble networks, maintaining multiple fixed-size ensembles using regularized evolution to edit one architecture at a time, creating ensembles that outperform deep ensembles in accuracy, uncertainty calibration, and robustness to dataset shift.

Other types of NAS have focused on efficiency. Training-free Neural Architecture Search (TE-NAS)[10], a type of NAS that ranks candidate architectures by analyzing the architecture itself rather than its performance on a dataset, searches for architectures without training, creating a rapid architecture search pipeline. More recent training-free methods like Rank-based Improved Firefly Algorithm (RB-IFA)[11] have lowered search time to under ten minutes. One-shot methods have also proven effective in improving efficiency; Once-for-All (OFA)[12] trains a single, over-parameterized network once to support all candidate sub-architectures using weight sharing. Child networks can be initiated instantly from the large network based on the deployment scenario.

Despite its success, NAS, especially with RL, faces several obstacles. Firstly, one of the most significant issues is the computation cost. As stated before, Zoph and Le[3] famously utilized 800 GPUs over 28 days, accumulating approximately 22,400 GPU days (over 500,000 GPU hours) to find leading architectures. Subsequent methods, while sometimes improving efficiency through techniques like proxy-less search (ProxylessNAS[13]), still frequently demand substantial computational resources. This high resource requirement limits the accessibility and practicality of NAS. Another important issue is the difficulty in generalizing architectures across tasks and datasets. Many NAS-discovered architectures perform well only on the specific datasets they were optimized for, such as CIFAR-10 or ImageNet, raising concerns about overfitting to proxy tasks or search settings[14]. Additionally, current NAS methods struggle with the search space design dilemmawhile a large, expressive search space offers greater potential (e.g., AutoML-Zero exploring from scratch[4]), it also increases the complexity and cost of the search, while smaller search spaces limit innovation and may exclude optimal solutions altogether.

Moreover, NAS methods often neglect multi-objective optimization. In practice, neural networks must balance multiple criteria beyond accuracy, such as latency, memory footprint, energy efficiency, and fairness. While some recent works attempt to incorporate these aspects (e.g., Mobile Neural Architecture Search (MnasNet)[15], Facebook Berkeley Network (FBNet)[16]), most current NAS algorithms remain single-objective in nature and fail to address real-world deployment constraints, especially for edge or mobile devices.

To address the challenge of computational cost while aiming for competitive performance, we propose Reinforced ArCHitEcture Learning (RACHEL), a NAS framework specifically designed for computational efficiency. We hypothesized that by combining an incremental RL-driven search with a stabilizing ensemble method, we could achieve competitive performance with significantly fewer computational resources. As a result, RACHEL integrates two key ideas:

1. Incremental RL-driven Architecture Candidate Generation: An actor-critic RL controller explores the architecture search space, incrementally building network designs.

2. Ensemble Learning with Safety Net: Instead of searching for a single best architecture, RACHEL builds an ensemble of performant networks discovered during the search, improving robustness and overall accuracy. A safety-net mechanism prevents catastrophic performance drops by reverting to a previously validated, stable model if a newly proposed candidate significantly underperforms.

We implement RACHEL using TensorFlow and a few GPU optimizations, including mixed-precision training and Accelerated Linear Algebra (XLA) compilation, primarily targeting NVIDIA L4 GPUs. Our results highlight RACHEL as a practical and efficient NAS framework, offering a compelling balance between accuracy and resource utilization. In this study, we demonstrate that RACHEL achieves competitive accuracy on standard benchmarks while drastically reducing computational cost, and it surpasses existing baselines on specialized binary classification tasks. We acknowledge that while prioritizing computational efficiency, the final architectures may not surpass the peak accuracy of NAS methods that require vastly more computational power. This study, therefore, accepts a trade-off between state-of-the-art performance and practical accessibility.

2 Architecture Search Process

2.1 Overview

We propose RACHEL, an RL-based framework for neural architecture search that constructs an ensemble of high-performing models through iterative refinement. The flowchart of the RACHEL process is shown in Figure 2.

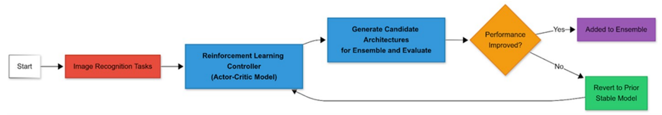
Start
Image Recognition Tasks
Reinforcement Learning Controller (Actor-Critic Model)
Generate Candidate Architectures for Ensemble and Evaluate
Performance Improved?
Yes
Added to Ensemble
No
Revert to Prior Stable Model

Fig. 2 The Iterative Process of the Reinforced Architecture Learning (RACHEL) Framework. This flowchart details the specific operational loop of the RACHEL framework for discovering and ensembling neural network architectures. The process begins with an actor-critic RL controller generating a batch of candidate architectures by incrementally adding layers to a base model. Each candidate is trained and evaluated on a validation set to measure its potential contribution to an evolving ensemble. This performance metric is used as a reward to update the RL controller's policy. The best-performing candidate from the batch is then subjected to a safety-net check; it is only added to the final ensemble if its inclusion improves upon the ensemble's previous state. This cycle repeats for a fixed number of iterations to build a robust final ensemble model.

2.2 Reward

The reward signal is not based on a candidate's standalone performance. To evaluate a candidate, it is temporarily added to the current ensemble. This frames the task of finding the best weights for this temporary ensemble as a convex optimization problem. Because the predictions from each base model are held constant (treated as fixed inputs), the overall cross-entropy loss function is convex with respect to the mixture weights, guaranteeing that a single, global optimal solution exists. These weights are learned by optimizing a trainable tensor of logits for 10 epochs on the validation set. Specifically, a gradient-based optimizer is used to iteratively adjust the logits to minimize the ensemble's cross-entropy loss, as it is the standard loss function for this multi-class classification task. We selected cross-entropy as it is the standard, principled loss function for this multi-class classification task. The resulting validation loss of the hypothetical combined ensemble (final_val_loss) serves as the objective function (obj = final_val_loss) for calculating the reward (r = obj). Since our goal is to minimize error (loss), we define the reward as the negative of the loss; a lower loss results in a higher reward for the RL agent. This objective is negated to form the reward signal (r = obj). The top-performing candidate, subject to a safety-net check that ensures stability, is then added to the growing ensemble. After a fixed number of iterations, the framework finalizes the model by learning the ensemble's prediction weights. This is achieved by optimizing a trainable tensor of logits for 50 epochs on the validation set using the Adam optimizer, with the final mixture weights determined by a softmax function. These mixture weights are then used to combine the models by taking a weighted average of the logits (the raw, pre-activation outputs) produced by each network in the ensemble.

2.3 RL Agent and the Search Space

The RL agent is composed of two parts: an actor that proposes designs and a critic that estimates how good those proposals are. This controller is built using a Long Short-Term Memory (LSTM) network, a type of neural network with memory, which is good for making a sequence of related decisions. The critic is implemented as a feed-forward network composed of a Flatten layer, a Dense hidden layer with 64 units and a ReLU activation, and a final Dense output layer to predict the value. The agent is trained on a combined loss function, which sums the policy gradient actor loss with a critic loss (the mean squared error between predicted values and actual rewards) that is weighted by a factor of 0.5. At each decision step in the architecture generation sequence (up to a predefined max_new_layers, which, when multiplied by the iteration count, defines the maximum number of layers a subnetwork can have), the LSTM generates probability distributions over a set of architectural tokens, each representing a specific design choice for a potential new layer or block. This token generation is its action. To generate these tokens, the process is initiated by a single random tensor, and the LSTM's own hidden state maintains context between sequential decisions.

As seen in Table 1, the RL controller first decides layer presence and layer type (either a standard convolutional block or a residual block) with binary token choices, and then defines the layer width. For example, a sequence of three token choices (0, 0, 50) represents a potential layer where the layer is present (token 1), is of type standard convolutional block (token 0), and has a width of 50 + 16 = 66 units. Specifically, the controller uses three separate dense output layers with 2, 2, and 241 units to generate the logits for layer presence, type, and width, respectively. The final layer width is calculated by adding a base of 16 to the sampled integer (0-240), resulting in a range of 16 to 256. The connectivity is sequential, with new architectural blocks always being appended to the end of the existing network. To control the agent's decision-making process, a sampling temperature is applied to the logits before the softmax operation. This temperature starts at a high value (1.5) and multiplicatively decays at each iteration. This schedule is crucial for balancing exploration and exploitation in the search process. Initially, the high temperature flattens the probability distribution, encouraging the agent to take more random actions and explore a

Table 1 Definition of architectural search space. This table shows the tokenizing process RACHEL uses to define each potential layer in a candidate architecture. Each layer is represented by a sequence of three token choices made by the RL controller.

Decision Component	Description	Token Value	Operation
Layer Presence	A binary decision indicating whether the layer should be added.	0	Absent: No layer is added at this step.
		1	Present: The layer defined by the next two tokens is added.
Layer Type	A categorical choice among predefined block types.	0	Standard Convolutional Block: A sequence of Conv2D (3x3 kernel), BatchNormalization, ReLU, Dropout, and MaxPooling2D.
		1	Residual Block: A two-layer residual block with a skip connection. The main path consists of two Conv2D layers (3x3 kernel), each followed by BatchNormalization and ReLU. The skip connection uses a Conv2D (1x1 kernel) to match dimensions if necessary before being added back to the main path.
Layer Width (Units/Filters)	An integer specifying the number of filters or units in the layer, sampled from a defined range (16 to 256).	0 to 240	Number of Filters/Units: The token value is mapped to a final layer width by the formula: `width = token_value + 16`. This results in a search space for layer widths ranging from 16 to 256. This decision is also ignored if Layer Presence is 0.

diverse range of novel architectures (exploration). As the search progresses and the agent's policy improves, the temperature is lowered, which sharpens the distribution and causes the agent to more consistently sample actions it already believes are best (exploitation). This ensures that the search is both broad enough to discover new ideas and focused enough to refine the most promising ones. The actor component samples actions from the generated distributions, while the critical component estimates the expected reward for a given state represented by the LSTMs input/hidden state, to compute the advantage (A = r - V(s), where A is the advantage, r is the actual reward, and V(s) is the estimated reward) used in policy updates. The given state is the LSTM controllers internal hidden (and cell) state vector after processing the dummy input sequence, which encodes all previous layeraddition decisions. The advantage is a signal that tells the actor how much better or worse their action was compared to the average expected outcome. A summary of the RL loop is shown in Algorithm 1.

The decode_full_candidate function translates a complete token sequence into a runnable Keras model[17]. Keras is a popular library that simplifies the process of building and training neural networks. It instantiates layers based on the tokens, incorporating standard components such as Conv2D, BatchNormalization, ReLU activation, MaxPooling2D, Dropout, and optional residual connections. These are all fundamental building blocks: for instance, Conv2D layers are specialized for finding patterns in images, while Dropout helps prevent the model from overfitting by reducing its tendency to memorize the training data. Input preprocessing consists of normalizing images to (0, 1) and applying on-the-fly data augmentation. The augmentations are limited to random horizontal flips and $\pm 10\%$ translations; no advanced techniques like cutout are used. This conservative approach to which is comparable to or simpler than that of many baselines, helps ensure a fair comparison focused on architectural discovery. Dropout rates and layer widths (number of filters/units) are adapted based on token values and architectural depth, allowing flexible and expressive model generation. Because each candidate model is trained from scratch on the full dataset, our framework avoids the issue of catastrophic forgetting of weights, a common challenge in parameter-sharing NAS methods.

2.4 Safety Net

A safety-net mechanism is employed to increase stability. Before a new candidate architecture is added to the ensemble, its performance contribution is compared against that of the previously accepted model. If the new candidate performs worse (if the objective is 0.01 less than the previous), it is discarded, and the previous best subnetwork is re-added to the ensemble. This prevents noisy evaluations or poor exploratory steps from corrupting the ensemble, ensuring its quality does not decline over time while also reinforcing successful subnetworks by increasing the weight of a proven architecture in the final ensemble. This approach is most advantageous in the early stages of the search, where the RL agent's policy is underdeveloped and prone to generating subpar candidates, and when operating on datasets

Algorithm 1 Pseudocode for the RL Decision Process. The RL agent is rewarded for its decisions based on the overall performance of the ensemble after its proposed subnetwork is added.

```
for each iteration do
    for each candidate in iteration do
        Use the previous subnetwork as a base
        for each new layer (up to max) do
            RL agent $\rightarrow$ choose presence
            RL agent $\rightarrow$ choose layer type
            RL agent $\rightarrow$ choose width
            Add layer to subnetwork
        end for
        Add the new subnetwork to the temporary ensemble
        for 10 epochs do
            Optimize mixture weights
        end for
        Evaluate ensemble
        Reward RL agent based on performance
    end for
    Choose the subnetwork candidate with the best reward to
    add to the ensemble
end for
```

where performance evaluations may be noisy.

2.5 Optimizations and Hyperparameters

Each candidate model was trained using standard optimization procedures. RACHEL uses the Adam optimizer[18], a learning rate scheduler (lr_scheduler) that gradually reduces the learning rate over epochs, and Early Stopping based on validation loss (val_loss) to prevent overfitting. Rachel uses categorical cross-entropy for its loss function, as it is the standard, principled loss function for this multi-class classification task. The Adam optimizer was chosen for its well-established rapid convergence. The optimizer is the engine that adjusts the model's parameters to reduce error.

We also use an Adam optimizer[18] with a learning rate of 1 10 for both actor and critic. The discount factor is set to = 0.9, and an entropy coefficient of 0.01 is applied to encourage exploration. These hyperparameters govern the learning dynamics of the RL agent. The controllers sampling temperature starts at 1.5 and decays multiplicatively by 0.95 at each iteration. All experiments, including RL controller iterations, candidate model training, and ensemble evaluation, were conducted on a single NVIDIA L4 GPU. Other hyperparameters included num_candidates=4 per iteration, iterations=10, and max_new_layers=3 per step. Training for each candidate was limited with early stopping applied using a min_delta=0.001 on validation loss and a patience=10. To assess the reliability of the search process, each task was repeated across 5 independent runs.

2.6 Data

RACHEL is implemented in TensorFlow 2.x[19], utilizing tf.data for efficient data handling and pipeline construction. TensorFlow is a comprehensive software library developed by Google for machine learning tasks. Input images are normalized to (0, 1). To maximize performance on modern GPUs (the NVIDIA L4 in this case), the framework supports Mixed Precision Training[20], Accelerated Linear Algebra (XLA), and the Distributed Training Strategy. These are advanced software optimizations that significantly speed up the computationally intensive training process, allowing us to achieve results much faster.

This study exclusively utilized publicly available and anonymized image datasets, thereby avoiding concerns related to human subject privacy or data confidentiality. The research adheres to standard scientific practices for reproducibility and responsible innovation.

3 Results

1. Canadian Institute For Advanced Research 10 Class Dataset (CIFAR-10): A dataset that consists of 60,000 32x32 color images, categorized into 10 distinct classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset is divided into 45,000 training images, 5,000 validation images, and 10,000 test images.

2. Street View House Numbers dataset (SVHN): Over 600,000-digit images (3232) of street view house numbers. We use the core training (73,257) and test (26,032) sets, with a 10% validation split from training.

3. Fashion Modified National Institute of Standards and Technology database (Fashion-MNIST): 70,000 2828 grayscale images of 10 fashion categories. Split: 55,000 train, 5,000 validation, 10,000 tests.

The first set of experiments focused on binary classification tasks using CIFAR-10, where five binary subsets were constructed to enable direct comparison with the AdaNet1 benchmarks: Cat vs. Dog, Deer vs. Truck, Deer vs. Horse, Automobile vs. Truck, and Dog vs. Horse. The second set involved full multi-class classification on the CIFAR-10, SVHN, and Fashion-MNIST datasets. Test accuracy on the held-out test set served as the primary performance metric, while total GPU Hours measured the end-to-end computational cost of the neural NAS process on the specified hardware when such data was available from the comparison baseline methods.

RACHELs performance was benchmarked against a diverse and comprehensive set of baseline methods tailored to each specific task and dataset. For the binary classification tasks on CIFAR-10, RACHEL was directly compared with AdaNet[21]. For the full multi-class classification tasks, the comparison suite was significantly expanded. On CIFAR-10, RACHEL was evaluated against foundational and widely recognized methods such as MetaQNN[6], NAS with RL[3], and Progressive NAS (PNAS)[8]. On the SVHN dataset, the evaluation included a broad array of modern competitors, including Dirichlet Neural Architecture Search (DrNAS), Random Search with Parameter Sharing (RSPS)[22], Self-Evaluated Template Network (SETN)[23], Gradient-based[24], Partial Channel Connections for Memory-Efficient Architecture Search (PC-DARTS)[25], and a strong ResNet baseline from a previous study[26]. For Fashion-MNIST, RACHELs performance was compared with methods like DeepSwarm[27], an evolutionary algorithm[28], Evolutionary Cross-Topology Neural Architecture Search (ECToNAS)[29], and MO-ResNet[30].

To gauge the contributions of ensembling and RL, two ablation studies were conducted on RACHEL, one with no ensembling and one with random search instead of RL, both evaluated on CIFAR-10.

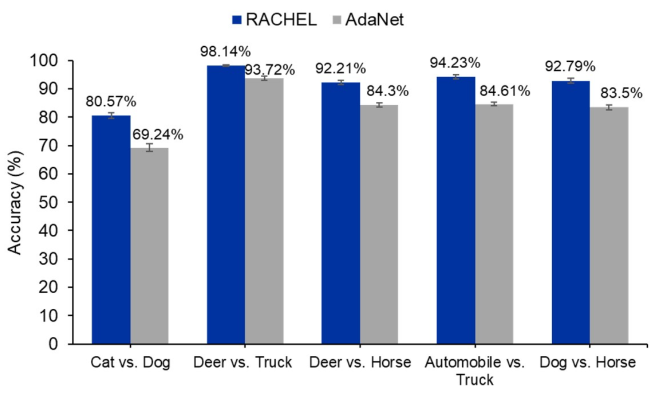

Fig. 3 RACHEL Significantly Outperforms AdaNet on Binary CIFAR-10 Classification Tasks (N=5). This bar chart compares the mean test accuracy of the RACHEL framework against the AdaNet baseline across five binary classification tasks derived from the CIFAR-10 dataset. For each task, the RACHEL framework was run five independent times (N=5) to discover and train an ensemble model, with the final accuracy evaluated on a held-out test set. Each RACHEL bar represents the mean accuracy across the five runs, and error bars represent the standard deviation.

3.1 Binary Classification Tasks (CIFAR-10)

For the binary tasks on CIFAR-10, RACHEL consistently outperformed AdaNet[21] across all tasks, with accuracy improvements ranging from approximately 4% to over 11%. RACHEL achieved substantial accuracy improvements over Adanet[21]: Cat vs. Dog (80.57%±0.91% vs. 69.24%±1.29%, an 11.33% improvement, $p<0.001$), Deer vs. Truck (98.14%±0.22% vs. 93.72%±0.82%, a 4.42% improvement, $p<0.005$), Deer vs. Horse (92.21%±0.84% vs. 84.30%±0.76%, a 7.91% improvement, $p<0.001$), Automobile vs. Truck (94.23%±0.66% vs. 84.61%±0.69%, a 9.62% improvement, $p<0.001$), and Dog vs. Horse (92.79%±0.80% vs. 83.50%±0.89%, a 9.29% improvement, $p<0.001$) (Figure 3). AdaNet results are benchmark values from Cortes et al.[21] Each binary task required under two hours of training on a single NVIDIA L4 GPU, though a computational cost comparison with AdaNet is difficult due to AdaNet not specifying training times or GPU type. The safety-net mechanism was rarely activated (only once or twice per 10 iterations), suggesting that the RL controller and adaptive training strategy enabled stable and effective model selection.

3.2 Full Multi-Class Classification

For the full CIFAR-10 classification task, RACHEL was benchmarked against several established NAS methods. RACHEL achieved a competitive average accuracy of 92.67%±0.21%, slightly trailing MetaQNN[6,31] (93.08%) and falling short of the higher accuracies achieved by NAS with RL[3] (96.35%) and PNAS[8] (96.59%, $p<0.001$) (Figure 4). Data for competitor methods are benchmark values reported in their respective publications[3,6,8], and the GPU time for NAS with RL was reported in Zhao et al.[32] However, these methods required vastly greater computational resources, ranging from 2,400 to over 500,000 GPU hours[32], compared to RACHELs ∼8 GPU hours on a single NVIDIA L4 GPU. We trained a standard ResNet-18 architecture to convergence using the identical simplified training protocol to provide a more direct baseline. Under these controlled conditions on an NVIDIA L4 GPU with XLA and Mixed Precision and simple augmentation, the ResNet-18 baseline achieved a mean test accuracy of 83.57%±0.10%. RACHEL significantly outperformed ResNet-18 ($p<0.001$). To assess the real-world applicability of RACHEL, the inference times and average sizes for each model on runs on CIFAR-10 were recorded, averaging 17.121s±0.901s per pass over the 10,000 images in CIFAR-10s test set. In addition, the average number of parameters per ensemble was 14,811,516 (an average of 1.48 million per subnetwork). The fast inference times and reasonable parameter count show that RACHEL is practical in deployment.

It is important to note that the experimental setups between the compared studies varied greatly. The type of GPU MetaQNN[6] was trained on was specified only as Nvidia GPUs, but it is specified that the experiment utilized 10 GPUs simultaneously. Training took up to 10 days to complete with a batch size set to 128, resulting in ∼2400 GPU hours (240 hours x 10 GPUs = 2400 GPU hours). PNAS[8] completed training using P100 GPUs, and since it is stated in Liu et. al.[8] that PNAS[8] is 8 times faster

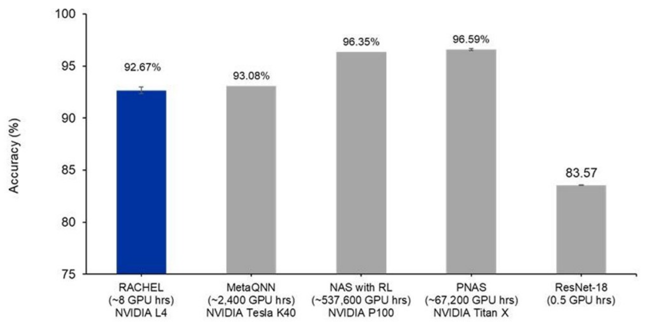

Fig. 4 RACHEL Achieves Competitive Accuracy on CIFAR-10 with Drastically Reduced Computational Cost (N=5). This bar chart compares the mean test accuracy of RACHEL against other prominent NAS methods (MetaQNN, NAS with RL, PNAS) on the full 10-class CIFAR-10 classification task. The computational cost, measured in GPU hours, is noted below each method's bar. RACHEL's accuracy was determined by averaging the performance of the final discovered ensembles on the CIFAR-10 test set across five independent runs (N=5) on a single NVIDIA L4 GPU. The RACHEL bar shows the mean accuracy, and its error bar represents the standard deviation.

than NAS with RL[3], which took up to ∼537,600 GPU hours (recorded in ENAS[7]), PNAS[8] ran for a total of ∼67,200 GPU hours. The exact GPU type used in NAS with RL[3] was not specified, but the massive GPU hour count derived from ENASs[7] reports evidently positions RACHEL as a more efficient NAS system. Compared to the massive computational resources required by the other methods, RACHEL demonstrates practical efficiency, even though additional tests were performed on a faster setup using an L4 GPU with XLA and mixed precision. Beyond these hardware differences, RACHEL employs a simple data augmentation strategy (random flips and translations), whereas the baseline papers do not detail their augmentation pipelines beyond standard CIFAR-10 preprocessing. By avoiding advanced augmentation techniques, we ensure a more direct and fair comparison of the search strategies themselves.

On SVHN, RACHEL achieved 96.62%±0.29% test accuracy in approximately 7 GPU hours. RACHEL slightly outperforms all other compared methods (Figure 5). It demonstrates a slight advantage over the next-best models, including DrNAS (96.30%±0.03%), RSPS (96.17%±0.06%), and a manually designed ResNet baseline (96.13%±0.10%) (Figure 5). RACHEL also slightly surpasses SETN, which scores 96.02%±0.06% (Figure 5). RACHEL's accuracy is comparably higher than that of Differentiable Architecture Sampler (GDAS) (95.57%±0.25%) and PC-DARTS (95.40%±0.33%) (Figure 5). All competitor results are benchmark values, including the ResNet baseline, were reported by Lee et al., 2021[26]. To account for the class imbalance in the SVHN dataset, we also used the macro F1 score. This metric provides a better measure of model performance by balancing precision (minimizing false positives) and recall (minimizing false negatives). The achieved score of 0.9629±0.0031 demonstrates that RACHEL performs exceptionally well across all classes, not just the most frequent ones. By consistently performing competently against a wide array of established NAS techniques, RACHEL demonstrates its practical capability in discovering high-performance architectures and achieving high performance among the compared methods.

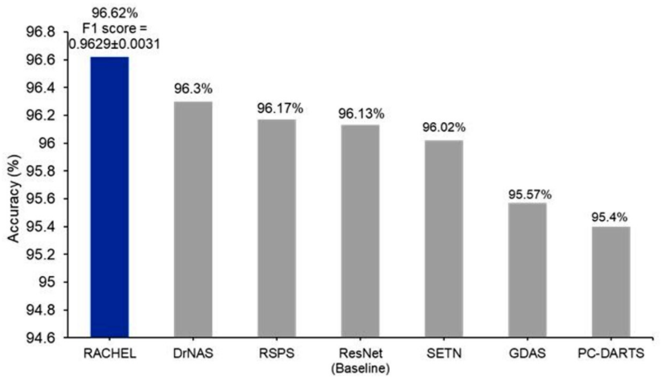

Fig. 5 RACHEL Achieves State-of-the-Art Accuracy Among Compared Methods on the SVHN Dataset (N=5). This bar chart compares the mean test accuracy of RACHEL with DrNAS, RSPS, ResNet, SETN, GDAS, and PC-DARTS. The F1 score is also included for RACHEL. RACHEL's performance was evaluated by averaging the test accuracy of the final ensembles from five independent search runs (N=5).

On Fashion-MNIST, RACHEL achieved 94.76%±0.47% test accuracy in approximately 12 GPU hours. GPU hour comparisons are limited as they are not consistently reported for all competitors on this dataset. RACHEL demonstrates strong performance on Fashion-MNIST, outperforming DeepSwarm (93.25%)[27], an evolutionary algorithm (93.20%)[28], and significantly over ECToNAS (87.20%±1.5%, $p<0.001$)[29]. While MO-ResNet[30] reports a higher accuracy of 95.91%, RACHELs accuracy (94.76%±0.47%) is highly competitive, especially considering its demonstrated efficiency on other datasets (Figure 6). Competitor accuracies are benchmark values sourced from their respective publications[28–30]. To provide a more direct baseline, we trained a standard ResNet-18 architecture to convergence under the same conditions as RACHEL. Under these controlled conditions on an NVIDIA L4 GPU with XLA and Mixed Precision, the ResNet-18 baseline achieved a mean test accuracy of 91.19%±0.13%, whereas RACHELs accuracy of (94.76%±0.47%) is significantly higher than the result of ResNet-18 in this case ($p<0.001$) (Figure 6).

The full RACHEL framework (RLdriven search + ensembling + safetynet) attains 92.67% ± 0.21% in accuracy (Figure 7). Removing ensembling and reporting the single best RLdiscovered

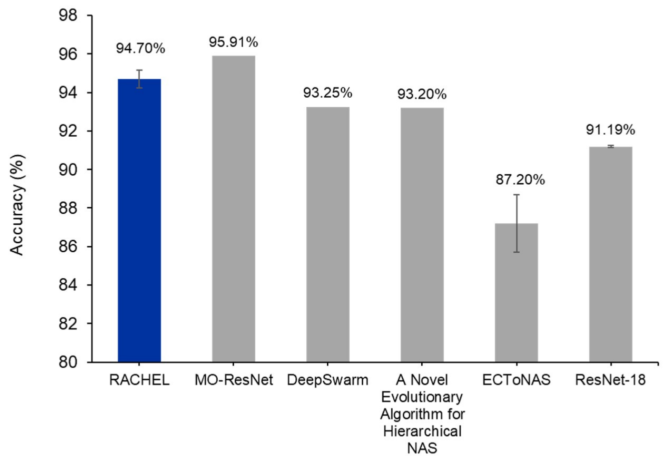

Fig. 6 RACHEL Demonstrates Highly Competitive Performance on the Fashion-MNIST Dataset (N=5). This bar chart compares the mean test accuracy of RACHEL with other published methods on the Fashion-MNIST classification task. The accuracy for RACHEL was calculated as the mean performance on the Fashion-MNIST test set across five complete, independent search and evaluation runs (N=5).

model reduces accuracy to 88.44% $\pm$ 0.34%, $p<0.001$, indicating that ensembling contributes a +4.23% improvement under our settings. Overall, these results indicate that ensembling (with the safetynet) is essential to the models performance.

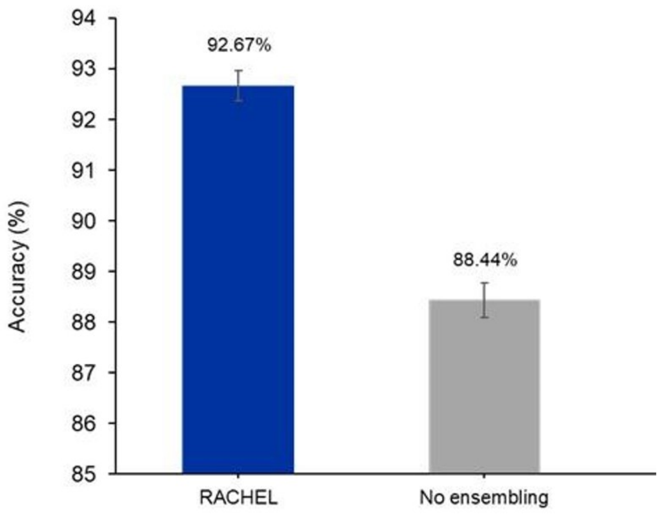

Fig. 7 Accuracy of full RACHEL system vs. RACHEL without ensembling vs. RACHEL with random search instead of RL on CIFAR-10. This bar chart compares the mean test accuracy of the full version of RACHEL with the mean test accuracy of RACHEL without ensembling. The accuracy for RACHEL was calculated as the mean performance on the CIFAR-10 test set across five complete, independent search and evaluation runs (N=5).

4 Discussion

In this work, we introduced RACHEL, a novel NAS framework designed to balance high performance with computational efficiency. The pioneering work[3] required over 500,000 GPU hours, and even more advanced methods like PNAS[8] still demanded thousands of GPU hours. On the other hand, RACHEL produced a high-performing architecture on CIFAR-10 in approximately 8 GPU hours on a single GPU (the NVIDIA L4). This efficiency was achieved by avoiding a full search from scratch at each step and using the previously validated architectures instead with the RL controller. On the binary CIFAR-10 tasks, RACHEL's higher accuracies over AdaNet[21] suggest that its RL-guided candidate generation is better than AdaNets heuristic search[21]. On the SVHN dataset, RACHEL surpassed a wide array of studies. RACHEL also achieved a competitive accuracy on Fashion-MNIST (94.76%$\pm$0.47%), further demonstrating its adaptability. Furthermore, the safety net was rarely triggered in our experiments, indicating that the RL controller successfully learns to propose consistently beneficial edits.

While its accuracy on CIFAR-10 (92.67%) is competitive, it still does not perform better in accuracy than the other CIFAR-10 state-of-the-art projects. RACHEL prioritizes achieving a highly competitive result while spending a minimal amount of GPU hours, rather than pursuing the marginal gains that require significantly more computation. Its competitive performance on the other datasets, namely SVHN and Fashion-MNIST, further demonstrates its competence and adaptability to different image recognition tasks. RACHELs design goals are more similar to resource-efficient methods like MnasNet[15] and FBNet[16], although it focuses on GPU-hour efficiency rather than on-device latency. Compared to evolutionary approaches like AutoML-Zero[4], which evolves algorithms from scratch, RACHEL uses a more constrained search space, simplifying the task for the RL-controller and hastening convergence.

RACHEL's primary application lies in scenarios requiring rapid model development with limited computational resources. Academic labs, startups, or researchers can use it to quickly generate high-performing, custom architectures for specific image classification tasks without needing large-scale GPU clusters. This effectively democratizes access to automated model design.

Dataset scale is an obstacle that challenges the efficiency of RACHEL. Since our method ensures each candidate is trained thoroughly via early stopping, the time required to reach convergence for a single model a larger dataset like ImageNet would be huge. The cost of generating a single reward signal would become prohibitively expensive, diminishing the framework's core advantage of rapid search. RACHEL's efficiency-focused design also faces many challenges beyond the dataset. The first arises from its incremental, append-only search strategy, where an early, suboptimal architectural choice can permanently weaken the ensemble. In addition, the controller can only assemble architectures from a predefined set of blocks (convolutional and residual). It cannot invent novel operations, meaning that if a task is best solved by a different type of architecture, such as a transformer, RACHEL would be unable to discover it without modification to the search space. In addition, the hyperparameters for RACHEL in this experiment werent tuned thoroughly, so future experiments for optimization of these values could be conducted.

Thus, even though Rachel has achieved promising results, there is future work that can be done. The architectural search space could be expanded to include more diverse operations, such as depth wise separable convolutions or attention modules. However, this would make the search problem significantly harder for the RL agent, as it would need to explore a much larger, more complex policy landscape. Consequently, converging to a high-performing architecture would likely require increasing the number of candidates and iterations, increasing the overall computational cost. Innovations are needed to keep the efficiency of RACHEL while allowing it to be more architecturally expressive.

In addition, the RL algorithm could be improved. Testing more advanced RL algorithms (e.g., Proximal Policy Optimization) or more nuanced reward-shaping strategies, especially when coupled with an expanded search space that would require more complex algorithms to navigate, could further increase search quality. Finally, Rachel could be applied to many more diverse tasks, even beyond image classification, to further assess its adaptability. Future work could assess its generalizability by applying it to other tasks, such as natural language processing or time-series analysis, by modifying the search space to include domain-appropriate operations, as stated in the architectural future work above. Furthermore, explainability methods like Grad-CAM could be used to interpret the learned models, visualizing the image features that influence the final classification to ensure they are semantically meaningful.

In the end, RACHEL proved to be an effective NAS framework that combines RL-based architecture search with ensemble learning and a safety-net mechanism for an efficient and robust search, boasting competitive performance across diverse datasets, demonstrating its adaptability.

References

1 Y. LeCun, Y. Bengio and G. l. n. Hinton, `https://doi.org/10.1038/nature14539`., DOI:.

2 T. Elsken, J. Metzen and F. Hutter, *Neural architecture search: A survey*, `https://doi.org/10.48550/arXiv.1808.05377`., DOI:.

3 B. Zoph and Q. Le, *Neural architecture search with reinforcement learning*, `https://doi.org/10.48550/arXiv.1611.01578`., arXiv preprint arXiv:1611.01578 2016. DOI:.

4 E. Real, C. Liang, D. So and Q. Le, *Automl-zero: Evolving machine learning algorithms from scratch*, `https://doi.org/10.48550/arXiv.2003.03384`., DOI:.

5 H. Liu, K. Simonyan and Y. Yang, *Darts: Differentiable architecture search*, `https://doi.org/10.48550/arXiv.1806.09055`., arXiv preprint arXiv:1806.09055 2018. DOI:.

6 B. Baker, O. Gupta, N. Naik and R. Raskar, *Designing neural network architectures using reinforcement learning*, `https://doi.org/10.48550/arXiv.1611.02167`., arXiv preprint arXiv:1611.02167 2016. DOI:.

7 H. Pham, M. Guan, B. Zoph, Q. Le and J. Dean, *Efficient neural architecture search via parameters sharing*, `https://doi.org/10.48550/arXiv.1802.03268`., DOI:.

8 C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang and K. Murphy, *Progressive neural architecture search*, `https://doi.org/10.48550/arXiv.1712.00559`., DOI:.

9 F. Li, S. Zhao, H. Pi, Y. QING, Y. Fu, S. Wang and H. Cui, *Neural Architecture Search via Ensemble-based Knowledge Distillation*, `https://openreview.net/pdf?id=G9M4FU8Ggo`.

10 W. Chen, X. Gong and Z. Wang, *Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective*, `https://doi.org/10.48550/arXiv.2102.11535`., arXiv preprint arXiv:2102.11535 2021. DOI:.

11 N. T., M. N. and N. A, *Neural Architecture Search: Tradeoff Between Performance and Efficiency*.

12 H. Cai, C. Gan, T. Wang, Z. Zhang and S.-f.-a. Han, *Train one network and specialize it for efficient deployment*, `https://doi.org/10.48550/arXiv.1908.09791`., arXiv preprint arXiv:1908.09791 2019. DOI:.

13 H. Cai, L. Zhu and S. Han, *Proxylessnas: Direct neural architecture search on target task and hardware*, `https://doi.org/10.48550/arXiv.1812.00332`., arXiv preprint arXiv:1812.00332 2018. DOI:.

14 A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox and F. Hutter, *Understanding and robustifying differentiable architecture search*, `https://doi.org/10.48550/arXiv.1909.09656`., arXiv preprint arXiv:1909.09656 2019. DOI:.

15 M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard and Q. Le, *Mnasnet: Platform-aware neural architecture search for mobile*, `https://doi.org/10.48550/arXiv.1807.11626`., DOI:.

16 B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia and K. Keutzer, *Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search*, `https://doi.org/10.48550/arXiv.1812.03443`., DOI:.

17 Keras, `https://keras.io/`,, accessed 05/02/2025.

18 D. Kingma and J. Ba, *Adam: A method for stochastic optimization*, `https://doi.org/10.48550/arXiv.1412.6980`., arXiv preprint arXiv:1412.6980.

19 M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving and M. Isard, *TensorFlow: a system for Large-Scale machine learning*, `https://doi.org/10.48550/arXiv.1605.08695`., DOI:.

20 P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev and G. Venkatesh, *Mixed precision training*, `https://doi.org/10.48550/arXiv.1710.03740`., arXiv preprint arXiv:1710.03740 2017. DOI:.

21 C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri and S. Yang, *Adaptive structural learning of artificial neural networks*, `https://doi.org/10.48550/arXiv.1607.01097`., DOI:.

22 L. Li and A. Talwalkar, *Random search and reproducibility for neural architecture search*, https://doi.org/10.48550/arXiv.1902.07638., DOI:.

23 X. Dong and Y. Yang, *One-shot neural architecture search via self-evaluated template network*, https://doi.org/10.48550/arXiv.1910.05733., DOI:.

24 X. Dong and Y. Yang, *Searching for a robust neural architecture in four gpu hours*, https://doi.org/10.48550/arXiv.1910.04465., DOI:.

25 Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian and H. Xiong, *Pc-darts: Partial channel connections for memory-efficient architecture search*, https://doi.org/10.48550/arXiv.1907.05737., arXiv preprint arXiv:1907.05737 2019. DOI:.

26 H. Lee, E. Hyung and S. Hwang, *Rapid neural architecture search by learning to generate graphs from datasets*, https://doi.org/10.48550/arXiv.2107.00860., arXiv preprint arXiv:2107.00860 2021. DOI:.

27 E. Byla and W. Pang, *Optimising convolutional neural networks using swarm intelligence*, https://doi.org/10.48550/arXiv.1905.07350., DOI:.

28 A. Christoforidis, G. Kyriakides and K. Margaritis, *A novel evolutionary algorithm for hierarchical neural architecture search*, https://doi.org/10.48550/arXiv.2107.08484., arXiv preprint arXiv:2107.08484 2021. DOI:.

29 E. Schiessler, R. Aydin and C. Cyron, *ECToNAS: Evolutionary Cross-Topology Neural Architecture Search*, https://doi.org/10.48550/arXiv.2403.05123., arXiv preprint arXiv:2403.05123.

30 S. Wang, H. Tang and J. Ouyang, *A Neural Architecture Search Method using Auxiliary Evaluation Metric based on ResNet Architecture*, https://doi.org/10.48550/arXiv.2505.01313., DOI:.

31 B. Baker, O. Gupta, R. Raskar and N. Naik, *Accelerating neural architecture search using performance prediction*, https://doi.org/10.48550/arXiv.1705.10823., arXiv preprint arXiv:1705.10823 2017. DOI:.

32 Y. Zhao, Y. Liu, B. Jiang and T. Guo, *CE-NAS: An End-to-End Carbon-Efficient Neural Architecture Search Framework*, https://doi.org/10.48550/arXiv.2406.01414., arXiv preprint arXiv:2406.01414.