Prometheus: A Recursively Self-Improving NAS System

Anonymous[1]

[1]Anonymous Institution

Abstract Neural Architecture Search (NAS) automates model design, but for systems involving a Reinforcement Learning (RL) controller, one limitation is the fixed intelligence of the controller itself. We introduce Prometheus, a proof-of-concept NAS system that addresses this barrier through recursive self-improvement. Prometheus utilizes an RL agent that not only edits a target convolutional neural network using network morphism but also modifies its own architecture. This self-editing allows it to increase its intellectual capacity to achieve better rewards. We demonstrate that this approach, combining a self-editing GNN controller with heuristic-driven adaptation, achieves competitive performance on standard image classification benchmarks like CIFAR-10 (95.58%±0.64%), SVHN (97.21%±0.09%), and Fashion-MNIST (95.52%±0.22%), opening a new avenue of research in self-improving AI.

1 Introduction

Neural Architecture Search (NAS) automates network design, evolving from computation-heavy early methods (Zoph and Le, 2017) to more efficient approaches such as weight sharing (Pham et al., 2018), differentiable search (Liu et al., 2019), and network morphism that enables function-preserving edits during training (Chen et al., 2016). More recent advances include extensible zero-cost proxies like Eproxy (Li et al., 2023), robust training-free NAS techniques (He et al., 2024), and hardware-aware multi-objective differentiable NAS (Sukthanker et al., 2025). However, for NAS with RL, most systems rely on fixed-capacity, human-designed RL controllers. We ask: can a NAS agent improve both the target model and itself? We introduce Prometheus, a recursively self-improving NAS system whose Graph Neural Network (GNN)-based RL controller edits its own architecture as well as the target network. Unlike EAS (Cai et al., 2018), which applies morphism only to the target, Prometheus applies it to the agent too, using a block-based search space, graph representations, and heuristic-triggered self-modification. Across standard image classification benchmarks, Prometheus achieves competitive performance (95.58%±0.64% on CIFAR-10, 97.21%±0.09% on SVHN, and 95.52%±0.22% on Fashion-MNIST) while adding a new level of autonomy to NAS.

2 Methods

Prometheus is a two-network system, with an RL controller network editing a target network (the network being trained on the target task; image recognition in this case). The RL controller is a GNN, specifically a Graph Convolutional Network (GCN) (Kipf and Welling, 2017). In this formulation, the target CNN is represented as a graph $\mathcal{G}_t = (\mathcal{V}, \mathcal{E})$, where each node $v_i \in \mathcal{V}$ represents an operation (e.g., Conv2D (LeCun et al., 1998), ReLU (Nair and Hinton, 2010)) with a feature vector encoding its properties, and edges $\mathcal{E}$ represent the data flow. On every forward pass, the GCN performs message passing, allowing each node to aggregate information from its neighbors. The resulting network graph G_t feeds into policy heads that issue structurally informed actions. The GCN encoder produces node embeddings Z from the graph $Z = \text{GCN}(\mathcal{G}_t)$.

The resulting matrix Z is defined as belonging to the space $Z \in \mathbb{R}^{|\mathcal{V}| \times d_h}$, where $\mathbb{R}$ is the set of all real numbers, $|\mathcal{V}|$ is the number of nodes in the target network graph, and d_h is the dimensionality of those nodes. A global graph embedding z_g is then obtained by mean-pooling the node embeddings.

At the architectural level, every permissible edit is a block transformation. To avoid performance drops and retraining from scratch, these edits leverage network morphism techniques (Chen et al.,

2016), allowing the agent to modify the target network's architecture without completely resetting its learned weights. The specific function-preserving transformations implemented are:

- **Net2Wider (widen):** Following the operation from Chen et al. (2016), this action increases a layer's width (number of output channels). The new weight tensor is intelligently populated from the old one, and the subsequent layer is adjusted to keep the network's output invariant.
- **Net2Deeper (deepen):** This operation increases network depth by inserting a new layer initialized to perform an identity mapping.
- **Custom Thinning (thin)**: As the inverse of widening, this custom structured pruning operation reduces a layer's width by discarding filters. It is a "lossy" transformation that reduces complexity and relies on fine-tuning to recover performance.

The controller can choose from the following edits:

- **Add Convolutional Block:** Appends a Conv2d → BatchNorm2d → ReLU trio at a stage's end. The Conv2d is identity-initialized (Net2Deeper style), and BatchNorm2d starts with $\gamma = 1.0, \beta = 0.0$. After the block is appended, the agent chooses a channel multiplier to set the block's width, which is applied using Net2Wider.
- **Add Linear Block:** Deepens the classifier by inserting a Linear → BatchNorm1d → ReLU right before the final layer, initialized as an identity mapping. The agent then chooses a width that is applied using Net2Wider.
- **Resize Layer:** Selects any Conv2d or Linear node and scales its output dimension by a chosen factor using Net2Wider. A learned attention head pinpoints the most promising layer.
- **Add Skip Connection:** Creates a shortcut between two nodes within the same stage. If channel counts differ, an identity-initialized 1×1 convolution is automatically inserted.

The search process is initialized with a simple VGG-style CNN backbone (Simonyan and Zisserman, 2014). This starting architecture consists of three sequential stages, each featuring a Conv2d → BatchNorm2d → ReLU block, with channel dimensions increasing from 64 to 128 and finally to 256. The first two stages are followed by max-pooling. The network is connected to a classifier head composed of an adaptive average pooling layer and a single linear layer. The initial network was pre-trained for 50 epochs to ensure that the initial rewards for the RL agent are representative of the edit quality.

The RL component was optimized with Advantage Actor-Critic (A2C). The controller's operation is formalized as a Markov Decision Process (MDP). At each timestep t, the controller receives a state s_t representing the target network's graph structure and performance metrics. It then samples an action a_t from its policy $\pi_\theta(a_t|s_t)$. The reward R_t now combines accuracy with a quadratic penalty for exceeding a parameter budget, explicitly steering the search toward compact models. The reward function is:

$$R_t = 100 \cdot (\text{acc}_{t+1} - \text{acc}_t) - \lambda_p \cdot \max\left(0, \frac{P_{t+1} - P_{\text{thresh}}}{10^6}\right)^2 \tag{1}$$

where acc_{t+1} is the post-edit validation accuracy, P_{t+1} is the new parameter count, P_{thresh} is a budget (20M parameters), and λ_p is a penalty coefficient (0.2). If the parameter count exceeds 30 million, the model is automatically reverted, and the controller receives a -50 reward.

The controller's parameters θ and the value function's parameters ϕ are updated by minimizing a composite loss function $L(\theta, \phi)$, composed of a policy loss, a value loss, and an entropy bonus:

$$L(\theta, \phi) = -\log \pi_\theta(a_t|s_t) A_t + \beta_v (R_t - V_\phi(s_t))^2 - \beta_e \mathcal{H}(\pi_\theta(\cdot|s_t)) \tag{2}$$

where $A_t = R_t - V_\phi(s_t)$ is the advantage, $V_\phi(s_t)$ is the critic's value estimate, $\mathcal{H}$ is the policy entropy, and β_v, β_e are loss coefficients. An entropy bonus ($-0.0005 \cdot \text{entropy}$) was applied to the final loss function to encourage exploration.

The number of post-edit training epochs, E_{post}, scales in proportion to the magnitude of the architectural change:

$$E_{\text{post}} = \min\left(100, \text{round}\left(E_{\text{base}} \cdot \max\left(1.0, \frac{P_{t+1}}{P_t}\right)\right)\right) \tag{3}$$

where $E_{\text{base}} = 25$, and P_t, P_{t+1} are the parameter counts before and after the edit.

The RL agent was given the ability to prune itself in addition to growing itself, but only after certain heuristic triggers. If validation accuracy fails to improve for 5 iterations, a growth self-edit is triggered. If the model fails 3 consecutive dummy forward passes (a check for immediate NaN errors), a pruning self-edit occurs. Growth choices comprise deepening the GNN, widening its hidden layers, or deepening a policy head, all with function preserving operations. Pruning options, which are the reverse, were enabled only after the controller's parameter count exceeded 15,000.

Finally, the meta-agent's own learning rate is adaptive, annealing based on the target model's accuracy to balance exploration and exploitation:

$$lr_{\text{meta}} = lr_{\text{base}} - \min\left(1, \frac{\max(0, \text{acc} - \text{acc}_{\text{base}})}{\text{acc}_{\text{target}} - \text{acc}_{\text{base}}}\right)(lr_{\text{base}} - lr_{\text{min}}) \tag{4}$$

where $\text{acc}_{\text{base}} = 0.80$ and $\text{acc}_{\text{target}} = 0.93$ define the accuracy range for annealing.

3 Experiments and Results

We evaluated Prometheus on CIFAR-10, SVHN, and Fashion-MNIST over five independent runs for each experiment. The search was performed on a single NVIDIA L4 GPU.

3.1 Analysis of Self-Editing Mechanism

Table 1: Ablation study of the self-editing mechanism on CIFAR-10. Results are reported as mean ± standard deviation over five runs.

System Variant	Self-Editing	Peak Acc. (%)
Prometheus (Ablated)	Disabled	95.16±0.77
Prometheus (Full)	**Enabled**	**95.58±0.64**

An ablation study was conducted to measure the effects of self-editing. The ablation in Table 1 shows a 0.42% average gain when the controller's self-editing capability is enabled. While the upward trend is encouraging, it is not yet statistically significant. We conjecture that the benefit of self-editing would be magnified on higher-complexity datasets like CIFAR-100 and ImageNet where a fixed-capacity controller may not be enough. We also conjecture that the self-editing function would be better in longer searches that give the heuristic triggers more opportunities to activate and refine the controller. A broader study involving more datasets is required before drawing conclusions.

3.2 Benchmark Performance

Table 2: CIFAR-10 accuracy vs. other controller-based NAS methods.

Method	Top-1 Acc. (%)
NAS-RL (Zoph and Le, 2017)	96.35
PNAS (Liu et al., 2018)	96.60
ENAS (Pham et al., 2018)	97.11
EAS (CNN only) (Cai et al., 2018)	95.77
Prometheus (average)	**95.58±0.64**
Prometheus (best of 5 runs)	**96.48**

On CIFAR-10 (Table 2), Prometheus is competitive with its closest antecedent, EAS, while introducing the novel self-editing dynamic. It operates with a far smaller computational budget than methods like NAS-RL or PNAS. The search time took an average of 23 GPU hours.

Table 3: Test accuracy comparison on SVHN.

Method	Accuracy (%)
DrNAS (Chen et al., 2021) (reported by Lee et al., 2021)	96.30±0.05
EAS (Cai et al., 2018)	98.17
ResNet (baseline from Lee et al.; ResNet-56) (Lee et al., 2021)	96.13±0.19
Prometheus	**97.21±0.09**

On SVHN (Table 3), Prometheus outperforms all compared baselines except for EAS, demonstrating strong generalization without any dataset-specific tuning. This search took an average of 49 GPU hours.

Table 4: Test accuracy comparison on Fashion-MNIST.

Method	Accuracy (%)
MO-ResNet (Wang et al., 2025)	95.91
DeepSwarm (Byla and Pang, 2019)	93.56
Hierarchical NAS (Christoforidis et al., 2023)	93.25
Prometheus	**95.52±0.22**

On Fashion-MNIST (Table 4), it achieves 95.52%±0.22%, outperforming several evolutionary methods and remaining highly competitive with the state-of-the-art. The average search time was 20 GPU hours.

4 Conclusion

We introduced Prometheus, a NAS system built on the principle of recursive self-improvement. The system, which combines a self-editing GNN controller with block-based actions and heuristic-driven adaptation, achieves competitive accuracy on multiple benchmarks. The primary contribution of this work, however, is the mechanism. We present Prometheus as a successful proof-of-concept for a more autonomous class of NAS agents that can manage their own complexity. A key limitation is the reliance on hard-coded heuristics to trigger self-modification. Future work should aim to integrate this decision directly into the agent's learning process, for example, through a hierarchical RL policy. Solving this credit assignment problem is a key step toward building more general and truly autonomous machine learning systems.

References

Byla, E. and Pang, W. (2019). Deepswarm: Optimising convolutional neural networks using swarm intelligence. In *Proceedings of the 19th UK Workshop on Computational Intelligence (UKCI)*.

Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang, J. (2018). Efficient architecture search by network transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, pages 2787–2794.

Chen, T., Goodfellow, I., and Shlens, J. (2016). Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*.

Chen, X., Wang, R., Cheng, M., Tang, X., and Hsieh, C.-J. (2021). Drnas: Dirichlet neural architecture search. In *International Conference on Learning Representations*.

Christoforidis, A., Kyriakides, G., and Margaritis, K. (2023). A novel evolutionary algorithm for hierarchical neural architecture search. *arXiv preprint arXiv:2107.08484*.

He, Z., Shu, Y., Dai, Z., and Low, B. K. H. (2024). Robustifying and boosting training-free neural architecture search. In *International Conference on Learning Representations (ICLR) Posters*.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, H., Hyung, E., and Hwang, S. J. (2021). Rapid neural architecture search by learning to generate graphs from datasets. *arXiv preprint arXiv:2107.00860*.

Li, Y., Li, J., Hao, C., Li, P., Xiong, J., and Chen, D. (2023). Extensible and efficient proxy for neural architecture search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6199–6210.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. (2018). Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34.

Liu, H., Simonyan, K., and Yang, Y. (2019). DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proc. of the 27th International Conference on Machine Learning (ICML)*, pages 807–814.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. (2018). Efficient neural architecture search via parameter sharing. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 4095–4104.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sukthanker, R. S., Zela, A., Staffler, B., Dooley, S., Grabocka, J., and Hutter, F. (2025). Multi-objective differentiable neural architecture search. In *International Conference on Learning Representations (ICLR) Posters*.

Wang, S., Tang, H., and Ouyang, J. (2025). A neural architecture search method using auxiliary evaluation metrics based on resnet architecture. *arXiv preprint arXiv:2505.01313*.

Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.