

A A Simple Prompt

To empirically validate the hypothesis presented in Section 2, we employ the prompt depicted in Figure 1 to assess the capability of LLMs in generating effective ZCPs.

Figure 1: A representative minimal prompt used in our naive experiments.

Prompt for Naive

Please help me design **five novel Zero-Cost Proxies** for evaluating the **expressive performance** of a neural network on a given batch of image data using **training-free metrics**.

These proxies should require **no training**, and must be computationally cheap to evaluate, similar to existing zero-cost proxy methods such as **NWOT**, **SynFlow**, or **Jacobian Covariance**. However, you should not **copy or imitate** these methods — your goal is to **creatively propose original ideas** that align with the following requirements:

For each of the **five proxies**, do the following:
Firstly, provide a **one-paragraph description** of the idea, including how the score is computed.
Secondly, Implement the proxy in **Python**, as a function named 'evaluate(model, data)' that: Takes a neural network object ('model')(pytorch model) and image batch ('data')(tensor) as input. Returns a scalar score ('score') reflecting expressive performance.

Note: Do **not** use training. Avoid random sampling or stochastic components. The heuristic must work on general-purpose image data and standard neural networks (e.g., CNNs). Do **not** provide any additional explanation outside of the required description and code.

B Experimental settings

B.1 Evolution Settings

Table 1 summarizes the default hyper-parameter configuration used in APD evolutionary experiments. Searches are conducted on a single RTX4090 GPU with a fixed random seed of 0. Each candidate proxy’s performance is averaged over 5 independent forward passes using a batch of 16 random samples. We adopt CIFAR-10 as the primary benchmark and report transfer results on CIFAR-10 and ImageNet-16. The actor-critic controller is a two layer MLP with 256 hidden units. The actor and critic learning rates are set to 1e-3 and 1e-2, respectively, with a discount factor $\gamma = 0.9$. During search we run 10 episodes of 100 steps search and test every individual for 200 architectures. The controller retains a history window of 5.

Table 1: Hyper-parameters for Automatic Proxy Discovery.

Parameter	Value	Parameter	Value
Episodes	10	Steps	100
History windows	5	Discount factor	0.9
Actor learning rate	1e-3	Critic learning rate	1e-2
Input batch size	16	Repeats	5
Hidden size	256	Number of layers	2
Population size	5	β	1

B.2 DARTS Settings

We list the training hyper-parameters we adopt for all experiments on DARTS. Figure 22 shows the optimal architecture obtained by APD in the DARTS search space.

As detailed in Table 2, models are trained for 600 epochs with a fixed random seed of 42. We employ SGD with momentum 0.9, an initial learning rate of 0.025 that is decayed to zero via cosine annealing, weight decay of 5e-4, and gradient clipping at a maximum norm of 5. Architectures begin with 36

Table 2: Hyper-parameters for DARTS training.

Parameter	Value	Parameter	Value
Seed	42	Batch size	96
CutOut length	16	Initial channels	36
Cells / layers	20	Aux. loss weight	0.4
Learning rate	0.025	Momentum	0.9
Weight decay	5e-4	Grad clip (norm)	5
DropPath prob.	0.2	Lr scheduler	CosineAnnealing

channels and stack 20 cells. An auxiliary classifier is attached with a loss weight of 0.4. We apply CutOut regularization (length=16) and linearly increase DropPath probability 0.2 over the course of training. All runs use a batch size of 96.

B.3 Autoformer Settings

For all ImageNet-1k training runs, we adhere to the core setup prescribed by AutoFormer Yang et al. [2020], which is listed in Table 3. The subnet discovered by APD are reported in Figure 23, Figure 24 and Figure 25.

We retrain discovered AutoFormer architectures for 300 epochs on ImageNet-1k using AdamW with an initial learning rate of 5e-4, cosine decay to 1e-5, and a 5 epoch warm-up. The network consumes 224×224 images split into 16×16 patches, and optimization runs with a batch of 128. Regularization follows the AutoFormer baseline. RandAugment (9 transformations, magnitude 0.5), mixup-cutmix blending ($\alpha = 0.8$, switch probability 0.5), random erasing (probability 0.25, pixel model), and a linearly ramped drop-path of 0.1. All experiments use the retrain mode, thereby only the chosen subnet’s weights are updated while the supernet is frozen.

Table 3: Hyper-parameter configuration used to retrain the subnet on ImageNet-1K.

Category	Parameter	Value	Notes
Data	Input resolution	224×224	—
	Patch size	16	ViT patch size
	Batch size	128	—
Optimiser	Optimizer	AdamW	$\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$
	Initial LR	5e-4	Scaled by batch size / 512
	Weight decay	0.05	Decoupled
	LR schedule	Cosine	5 warm-up + 300 epochs
Regularisation	Drop-path rate	0.10	Linearly increased
	RandAugment	rand-m9-mstd0.5-inc1	Timm default
	Mixup / CutMix	$\alpha=0.8$, prob. = 1	Switch prob. = 0.5
Model	Training mode	retrain	Only subnet weights updated
	Max rel. position	14	Bias radius

B.4 LLMs Settings

For proxy generation, we interface with GPT-4o via the Chat Completion API Achiam et al. [2023] and evaluate APD across multiple LLMs, including Claude 3.7 Anthropic [2025], Grok 3 xAI [2025], Gemini 2.0 Flash Google Cloud [2025], Deepseek V3 Liu et al. [2024], and Qwen Plus Qwen Team [2024]. To balance diversity and syntactic correctness, APD is performed with a temperature of 0.5 and the output length is capped at 8192 tokens. We supply a specialized system prompt that constrains LLM to emit JSON object with two fields: thought + code.

C Prompt Engineering

In this section, we outline the guiding principles underlying the design of our prompts. Given that the effectiveness of LLM-driven heuristic discovery critically hinges on prompt quality, our prompts are carefully crafted to maintain clarity, specificity, and structured output. Particularly, we ensure

clarity and specificity by precisely defining the computational boundaries and structural properties of candidate proxies within each prompt, clearly delineating the allowable search space of the metrics to facilitate efficient exploration by the LLM. Inspired by previous auto proxy discovery research, we prompt the LLM to construct compositional proxies represented as directed acyclic graphs that integrate network-derived signals (e.g., gradients, activations, or weights) with basic statistical and arithmetic operations. Collectively, this rigorous prompt engineering strategy significantly enhances the efficiency, validity, and innovativeness of the ZCPs by APD.

Moreover, through extensive empirical experimentation, we observe that the initial code representation of a generated ZCP typically fails to fully realize its underlying conceptual potential. Specifically, the initial instantiation of the proxy, directly produces from a single prompt, often exhibits suboptimal correlation with the final performance metrics. It is only after applying evolutionary strategies such as mutation and crossover operations that iteratively refine the proxy structure and composition that the intrinsic effectiveness of the proxy becomes evident. This empirical insight underscores the necessity of iterative exploration and refinement within the APD, highlighting mutation and crossover as critical mechanisms for achieving robust and effective ZCPs.

C.1 Prompt For NAS-Bench-201

We provide the detailed prompts employed in our experiments on NAS-Bench-201 Dong and Yang [2020]. These prompts serve as explicit instructions guiding the LLM to systematically generate novel ZCPs. Specifically, we design three variants of prompts as depicted in Figure 2, 3, and 4. Each prompt variant progressively introduces structured definitions, constraints on computational operations, and explicit guidelines to ensure that the generated proxies adhere to our framework requirements, thereby facilitating effective exploration of the proxy search space and enhancing the quality of the discovered ZCPs.

C.2 Prompt For TransNAS-Bench-101

We detail the specific prompts utilized for proxy discovery experiments on TransNAS-Bench-101 Duan et al. [2021]. These prompts are carefully crafted to instruct the large language model (LLM) to produce novel zero-cost proxies tailored specifically for their diverse tasks, including JIGSAW, autoencoding, scene classification, and object classification. To accommodate these distinct downstream tasks, we designed three prompt variants as shown in Figure 5, 6 and 7.

C.3 Prompt For AutoFormer

We present the detailed prompts employed for proxy discovery experiments conducted on the AutoFormer architecture search space. Given AutoFormer’s specific structural characteristics and its transformer-based design for vision tasks, we carefully tailor three distinct prompt variants following previous designs in Figure 8, 9 and 10.

D Extended Experimental Results

D.1 Additional Ablation Studies

Analysis on Mutation and Crossover operations To figure out the roles of the mutation and crossover operations, we evaluate the individual contribution of the mutation and crossover operation by selectively disabling each component during proxy search and observing the resulting performance changes. Figure 4 illustrates that omitting the mutation prompt reduces the proxy’s correlation with ground-truth metrics while removal of the crossover operation narrows the architectural exploration space and weakens proxy accuracy. These findings confirm that mutation and crossover provide distinct but complementary functions. collectively enabling the effective of high-quality zero-cost proxies.

Analysis on population size To understand how the size of the evolving population affects the quality and convergence of our zero-cost proxy search, we perform an ablation study over three different population sizes: 1, 2, 3, 4, 5 and 10. A larger population can offer richer diversity and robustness but incurs greater computational overhead and makes it harder for the LLM to follow

Figure 2: Prompt used for initialization in NAS-Bench-201.

Prompt for Initialization

Your Task: Please design **5 Zero-Cost Proxies** to evaluate the representation capability of different convolutional network architectures on a given dataset. The final goal of each proxy is to provide a **scalar score** that reflects the performance of a given network. You should generate Zero-Cost Proxies according to the following specifications, which include: **proxy requirements**, **description of the proxy search space** (defining the structure of how the proxy is computed), **the search space of the networks being evaluated**, the **given dataset**, and the **output format**, as described below:

Proxy Requirements:

Training-free: No gradient descent, weight updates, or learned parameters

Efficient: Low computational cost, suitable for early-stage model selection

Deterministic: No stochastic elements or randomness

Input-aware: The proxy should utilize a batch of input image data

Model-sensitive: The proxy should reflect meaningful differences between models.

Proxy Search Space: The computation of the proxy can be represented as a **directed acyclic graph (DAG)**. For any given node, its in-degree must satisfy $0 \leq \text{degree}_{in} \leq 2$.

Nodes with only outgoing edges and no incoming edges are called **inputs**. Inputs consist of two parts: the input module and the input property. The input module refers to different types of network layers (e.g., convolutional layers, BatchNorm, activation layers, etc.), and the input property is one of the following: gradient G , weights W , or output feature maps Z . An input is defined as a combination of these two components. For example, (convolutional layer + gradient G) refers to the gradients of the convolutional layer’s parameters. A proxy can have multiple inputs.

Nodes with both incoming and outgoing edges are referred to as **operations**, which are further divided into two types:

- When $\text{degree}_{in} = 1$, the node typically performs a **statistical operation** such as computing the mean or standard deviation of the preceding node’s output.
- When $\text{degree}_{in} = 2$, the node typically performs a **binary operation**, such as addition, subtraction, or division between two inputs.
- Nodes with only incoming edges are **outputs**. There must be **only one output node** in the DAG. The input to this final node must be a **scalar**.

Network Search Space: Networks drawn from NASBench-201, varying only in cell structure. Macro architecture: `input → conv → cell_n → residual (stride=2) → cell_n → residual (stride=2) → cell_*n → global average pooling`. Cells: DAGs of four nodes with operations: zeroize, skip-connection, 1×1 conv, 3×3 conv, 3×3 avg-pool.

Given Dataset: CIFAR-10 classification with input shape $(3 \times 32 \times 32)$ and 10-dimensional output.

Output: For each proxy, provide a **Description** paragraph and **Code** demo implementing `evaluate(model, inputs, targets)` in PyTorch. Ensure numerical stability, deterministic outputs, and move any returned tensors to CPU.

Note: Your Zero-Cost Proxies must not be the same as any existing ones (such as `nwot`, `snip`, etc.), though you may draw inspiration from them. Ensure numerical stability in your functions (avoid `inf` or `nan` during computation). Do not provide any additional explanation outside of the required description and code.

context, thereby reducing ZCPs quality. In contrast, a smaller population accelerates each generation but risks premature convergence. We fix all other settings and report the average Spearman correlation between proxy scores in Table 5.

Analysis on hyperparameter To assess how the discount factor γ and the history window size affect proxy quality, we conduct two separate ablations, one varying γ over $\{0.5, 0.7, 0.9, 0.99\}$, and the other varying the history window over $\{1, 3, 5, 10\}$. All other settings remain fixed. Results are reported in Table 6 and Table 7.

Analysis on actor-critic To evaluate the effect of varying the number of hidden layers in the actor-critic controller on proxy search performance and convergence speed, we conduct an ablation

Figure 3: Prompt used for mutation in NAS-Bench-201.

Prompt for Mutation

Your Task: Building upon the initial set of **Zero-Cost Proxies**, you will now perform a **mutation** step to generate novel and distinct proxies. You should introduce creative variations into existing proxies through systematic modifications. Follow the mutation rules and constraints defined below to design **5 distinct mutated proxies**:

Mutation Rules: When mutating a proxy, you must choose at least **one** and up to **four** of the following mutations to apply:

- **Input Mutation:** Replace, add, or remove one type of input (e.g., switch from convolutional layer gradients (G) to activation outputs (Z)). Consider using less commonly utilized inputs such as BatchNorm statistics or residual block outputs.
- **Operation Mutation:** Change statistical operations (mean, variance, max, min) to another statistic. Substitute binary operations (+, -, , /) with different arithmetic operations, ensuring numerical stability.
- **Structure Mutation:** Adjust the computational graph by altering the connections (edges) between nodes, ensuring the DAG property remains valid. Introduce an additional intermediate node for richer representations.
- **Aggregation Mutation:** Modify how multiple layer values are aggregated (e.g., sum, mean, weighted average).

Mutated Proxy Search Space:

- Each node has an in-degree $0 \leq degree_{in} \leq 2$.
- Exactly **one output node** producing a scalar value.
- Inputs: Combinations of modules (conv, BatchNorm, activation layers, residual block outputs) and properties (gradients G , weights W , outputs Z).
- Operations: Statistical (mean, variance, max, min, sum) or binary (+, -, , /).

Network Search Space: Networks drawn from NASBench-201, varying only in cell structure. Macro architecture: $input \rightarrow conv \rightarrow cell_n \rightarrow residual (stride=2) \rightarrow cell_n \rightarrow residual (stride=2) \rightarrow cell_n \rightarrow global\ average\ pooling$. Cells: DAGs of four nodes with operations: zeroize, skip-connection, 1×1 conv, 3×3 conv, 3×3 avg-pool.

Given Dataset: CIFAR-10 classification with input shape $(3 \times 32 \times 32)$ and 10-dimensional output.

Output: For each proxy, provide a **Description** paragraph and **Code** demo implementing `evaluate(model, inputs, targets)` in PyTorch. Ensure numerical stability, deterministic outputs, and move any returned tensors to CPU.

Note: Do **not** use training. Avoid random sampling or stochastic components. The heuristic must work on general-purpose image data and standard neural networks (e.g., CNNs). Do **not** provide any additional explanation outside of the required description and code.

Existing Proxies:
<Existing Proxies>

over depths of 2, 3, 4, and 5 layers. Table 8 reports the final Spearman correlation achieved after convergence and the generation at which the search stabilized for each depth. The results indicate that evolution planning itself is relatively straightforward. A two-layer actor-critic controller already achieves convergence within 50 generations. Employing additional layers results in an increased number of ineffective generations, incurring unnecessary computational overhead without further improving performance (e.g., in the later stages of evolution when proxy quality improvements become minimal, continuing with fixed initialization procedures can result in diminishing returns, incurring unnecessary computational overhead).

Figure 4: Prompt used for crossover in NAS-Bench-201.

Prompt for Crossover

Your Task: Expanding upon the previously defined Zero-Cost Proxies, your next step is to implement a **crossover** operation to create novel proxies. The crossover process involves combining aspects of **two parent proxies** to produce **5 distinct crossover proxies**. Adhere to the crossover rules and constraints defined below:

Crossover Rules: To generate crossover proxies, select two parent proxies from previously defined proxies. Apply at least **one** and up to **two** of the following crossover techniques:

- **Input Crossover:** Exchange input nodes between parent proxies, such as combining gradients (G) from one proxy and outputs (Z) from another.
- **Operation Crossover:** Merge operation nodes by replacing a statistical or binary operation from one parent with an operation from the other parent.
- **Structure Crossover:** Blend the computational graph structures of two proxies by interchanging node connections, ensuring the resulting DAG remains valid.
- **Aggregation Crossover:** Combine aggregation strategies (e.g., sum, mean, weighted average) from the two parent proxies.

Crossover Proxy Search Space:

- Nodes must have an in-degree $0 \leq \text{degree_in} \leq 2$.
- Exactly **one output node** producing a scalar value.
- Inputs: combinations of modules (conv, BatchNorm, activation layers, residual block outputs) and properties (gradients G , weights W , outputs Z).
- Operations: statistical (mean, variance, max, min, sum) or binary (+, -, , /).

Network Search Space: Networks drawn from NASBench-201, varying only in cell structure. Macro architecture: $\text{input} \rightarrow \text{conv} \rightarrow \text{cell_n} \rightarrow \text{residual}(\text{stride}=2) \rightarrow \text{cell_n} \rightarrow \text{residual}(\text{stride}=2) \rightarrow \text{cell_n} \rightarrow \text{global average pooling}$. Cells: DAGs of four nodes with operations: zeroize, skip-connection, 1×1 conv, 3×3 conv, 3×3 avg-pool.

Output: For each proxy, provide a **Description** paragraph and **Code** demo implementing `evaluate(model, inputs, targets)` in PyTorch. Ensure numerical stability, deterministic outputs, and move any returned tensors to CPU.

Note: Do **not** use training. Avoid random sampling or stochastic components. The heuristic must work on general-purpose image data and standard neural networks (e.g., CNNs). **Do not** provide any additional explanation outside of the required description and code.

Existing Proxies:
<Existing Proxies>

D.2 NAS-Bench-201 Results

The complete set of results on NAS-Bench-201, as reported in the main text, can be found in Table 9.

D.3 Detailed Cross-LLM Performance

We track the evolution of average Spearman correlation over 20 generations for each LLM backbone under. Table 10 reports the per-generation correlation on NAS-Bench-201 for all seven models.

D.4 AutoFormer Accuracy Results

We evaluate the architectures discovered by APD in the AutoFormer search space on ImageNet-1k classification. Table 11 reports the Top-1 accuracies for the tiny, small, and base variants: the tiny model achieves 76.1 %, the small model 81.5 %. These results demonstrate that our search not only produces proxies with high correlation to ground truth but also yields architectures that deliver competitive performance on a large-scale vision benchmark.

Figure 5: Prompt for initialization in TransNAS-Bench-101.

Prompt for Initialization

Your Task: Please design **5 novel Zero-Cost proxies** to evaluate the representation capability of different convolutional network architectures on a given downstream task. The final goal of each proxy is to provide a **scalar score** that reflects the performance of a given network. You should generate Zero-Cost Proxies according to the following specifications, which include: **proxy requirements**, **description of the proxy search space** (defining the structure of how the proxy is computed), the **search space of the networks being evaluated**, the **given dataset**, and the **output format**, as described below:

Proxy Requirements:

- **Training-free:** No gradient descent, weight updates, or learned parameters.
- **Efficient:** Low computational cost, suitable for early-stage model selection.
- **Deterministic:** No stochastic elements or randomness.
- **Model-sensitive:** The proxy should reflect meaningful differences between models.

Proxy Search Space: The computation of the proxy can be represented as a **directed acyclic graph (DAG)**. For any given node, its in-degree must satisfy $0 \leq \text{degree}_{in} \leq 2$.

Nodes with only outgoing edges and no incoming edges are called **inputs**. Inputs consist of two parts: the input module and the input property. The input module refers to different types of network layers (e.g., convolutional layers, BatchNorm, activation layers, etc.), and the input property is one of the following: gradient G , weights W , or output feature maps Z . An input is defined as a combination of these two components. For example, (convolutional layer + gradient G) refers to the gradients of the convolutional layer’s parameters. A proxy can have multiple inputs.

Nodes with both incoming and outgoing edges are referred to as ****operations****, which are further divided into two types:

- When $\text{degree}_{in} = 1$, the node typically performs a **statistical operation** such as computing the mean or standard deviation of the preceding node’s output.
- When $\text{degree}_{in} = 2$, the node typically performs a **binary operation**, such as addition, subtraction, or division between two inputs.

Nodes with only incoming edges are **outputs**. There must be **only one output node** in the DAG. The input to this final node must be a **scalar**.

Network Search Space: The networks to be evaluated come from **TransNAS-Bench-101**, which includes variations in both **cell-level** and **macro-level structures**.

Macro architecture: $\text{img} \rightarrow \text{searched backbone (composed of stacked cells)} \rightarrow \text{task-specific decoder}$. The macro structure consists of **three stages**, each containing **modules** made of **residual blocks**. After each stage, downsampling and channel doubling occur. For example: Stage 1 (Module 1) \rightarrow residual blocks \rightarrow Stage 2 (Module 2, 3) \rightarrow residual blocks \rightarrow Stage 3 (Module 4) \rightarrow residual blocks \rightarrow task-specific head.

Cell structure: Each cell is modeled as a **directed acyclic graph (DAG)** with six nodes. For any $v_i, v_j \in V$, if $i < j$, then $e_{ij} \in E$. Each node represents a latent feature tensor, and each edge represents a candidate operation from the following set:

- zeroize
- skip-connection
- 1×1 convolution
- 3×3 convolution

Each cell forms the base unit of the backbone and is repeatedly stacked according to the macro-level configuration. This enables flexible network designs across different tasks such as object classification, semantic segmentation, surface normal estimation, and more.

Given Downstream Task:
<Given Downstream Task>

Figure 6: Prompt for mutation in TransNAS-Bench-101.

Prompt for Mutation

Your Task: Building upon the initial set of **Zero-Cost Proxies**, you will now perform a **mutation** step to generate novel and distinct proxies. You should introduce creative variations into existing proxies through systematic modifications. Follow the mutation rules and constraints defined below to design **5 distinct mutated proxies**:

Mutation Rules: When mutating a proxy, you must choose at least **one** and up to **four** of the following mutations to apply:

- **Input Mutation:** Replace, add, or remove one type of input (e.g., switch from convolutional layer gradients (G) to activation outputs (Z)). Consider using less commonly utilized inputs such as BatchNorm statistics or residual block outputs.
- **Operation Mutation:** Change statistical operations (mean, variance, max, min) to another statistic. Substitute binary operations (+, -, , /) with different arithmetic operations, ensuring numerical stability.
- **Structure Mutation:** Adjust the computational graph by altering the connections (edges) between nodes, ensuring the DAG property remains valid. Introduce an additional intermediate node for richer representations.
- **Aggregation Mutation:** Modify how multiple layer values are aggregated (e.g., sum, mean, weighted average).

Proxy Search Space: The computation of the proxy can be represented as a **directed acyclic graph (DAG)**. For any given node, its in-degree must satisfy $0 \leq \text{degree}_{in} \leq 2$.

Nodes with only outgoing edges and no incoming edges are called **inputs**. Inputs consist of two parts: the input module and the input property. The input module refers to different types of network layers (e.g., convolutional layers, BatchNorm, activation layers, etc.), and the input property is one of the following: gradient G , weights W , or output feature maps Z . An input is defined as a combination of these two components. For example, (convolutional layer + gradient G) refers to the gradients of the convolutional layer’s parameters. A proxy can have multiple inputs.

Nodes with both incoming and outgoing edges are referred to as ****operations****, which are further divided into two types:

- When $\text{degree}_{in} = 1$, the node typically performs a **statistical operation** such as computing the mean or standard deviation of the preceding node’s output.
- When $\text{degree}_{in} = 2$, the node typically performs a **binary operation**, such as addition, subtraction, or division between two inputs.

Nodes with only incoming edges are **outputs**. There must be **only one output node** in the DAG. The input to this final node must be a **scalar**.

Network Search Space: The networks to be evaluated come from **TransNAS-Bench-101**, which includes variations in both **cell-level** and **macro-level structures**.

Macro architecture: $\text{img} \rightarrow \text{searched backbone (composed of stacked cells)} \rightarrow \text{task-specific decoder}$. The macro structure consists of **three stages**, each containing **modules** made of **residual blocks**. After each stage, downsampling and channel doubling occur. For example: Stage 1 (Module 1) \rightarrow residual blocks \rightarrow Stage 2 (Module 2, 3) \rightarrow residual blocks \rightarrow Stage 3 (Module 4) \rightarrow residual blocks \rightarrow task-specific head.

Cell structure: Each cell is modeled as a **directed acyclic graph (DAG)** with six nodes. For any $v_i, v_j \in V$, if $i < j$, then $e_{ij} \in E$. Each node represents a latent feature tensor, and each edge represents a candidate operation from the following set:

- zeroize
- skip-connection
- 1×1 convolution
- 3×3 convolution

Each cell forms the base unit of the backbone and is repeatedly stacked according to the macro-level configuration. This enables flexible network designs across different tasks such as object classification, semantic segmentation, surface normal estimation, and more.

Given Downstream Task:
<Given Downstream Task>

Figure 7: Prompt for crossover in TransNAS-Bench-101.

Prompt for Crossover

Your Task: Expanding upon the previously defined Zero-Cost Proxies, your next step is to implement a **crossover** operation to create novel proxies. The crossover process involves combining aspects of **two parent proxies** to produce **5 distinct crossover proxies**. Adhere to the crossover rules and constraints defined below:

Crossover Rules: To generate crossover proxies, select two parent proxies from previously defined or mutated proxies. Apply at least **one** and up to **two** of the following crossover techniques:

- **Input Crossover:** Exchange input nodes between parent proxies, such as combining gradients (G) from one proxy and outputs (Z) from another.
- **Operation Crossover:** Blend the computational graph structures of two proxies by interchanging node connections, ensuring the resulting DAG remains valid.
- **Structure Crossover:** Blend the computational graph structures of two proxies by interchanging node connections, ensuring the resulting DAG remains valid.
- **Aggregation Crossover:** Combine aggregation strategies (e.g., sum, proxies).

Proxy Search Space: The computation of the proxy can be represented as a **directed acyclic graph (DAG)**. For any given node, its in-degree must satisfy $0 \leq degree_{in} \leq 2$.

Nodes with only outgoing edges and no incoming edges are called **inputs**. Inputs consist of two parts: the input module and the input property. The input module refers to different types of network layers (e.g., convolutional layers, BatchNorm, activation layers, etc.), and the input property is one of the following: gradient G , weights W , or output feature maps Z . An input is defined as a combination of these two components. For example, (convolutional layer + gradient G) refers to the gradients of the convolutional layer’s parameters. A proxy can have multiple inputs.

Nodes with both incoming and outgoing edges are referred to as ****operations****, which are further divided into two types:

- When $degree_{in} = 1$, the node typically performs a **statistical operation** such as computing the mean or standard deviation of the preceding node’s output.
- When $degree_{in} = 2$, the node typically performs a **binary operation**, such as addition, subtraction, or division between two inputs.

Nodes with only incoming edges are **outputs**. There must be **only one output node** in the DAG. The input to this final node must be a **scalar**.

Network Search Space: The networks to be evaluated come from **TransNAS-Bench-101**, which includes variations in both **cell-level** and **macro-level structures**.

Macro architecture: $img \rightarrow$ searched backbone (composed of stacked cells) \rightarrow task-specific decoder. The macro structure consists of **three stages**, each containing **modules** made of **residual blocks**. After each stage, downsampling and channel doubling occur. For example: Stage 1 (Module 1) \rightarrow residual blocks \rightarrow Stage 2 (Module 2, 3) \rightarrow residual blocks \rightarrow Stage 3 (Module 4) \rightarrow residual blocks \rightarrow task-specific head.

Cell structure: Each cell is modeled as a **directed acyclic graph (DAG)** with six nodes. For any $v_i, v_j \in V$, if $i < j$, then $e_{ij} \in E$. Each node represents a latent feature tensor, and each edge represents a candidate operation from the following set:

- zeroize
- skip-connection
- 1×1 convolution
- 3×3 convolution

Each cell forms the base unit of the backbone and is repeatedly stacked according to the macro-level configuration. This enables flexible network designs across different tasks such as object classification, semantic segmentation, surface normal estimation, and more.

Given Downstream Task:
<Given Downstream Task>

Figure 8: Prompt for initialization in AutoFormer.

Prompt for Initialization

Your Task: Please design **5 novel Zero-Cost Proxies** to evaluate the representation capability of different transformer network architectures on a given dataset. The final goal of each proxy is to provide a **scalar score** that reflects the performance of a given network. You should generate zero-cost proxies according to the following specifications, which include: **proxy requirements**, **description of the proxy search space** (defining the structure of how the metric is computed), the **search space of the networks being evaluated**, the **given dataset**, and the **output format**, as described below:

Proxy Requirements:

- **Training-free:** No gradient descent, weight updates, or learned parameters.
- **Efficient:** Low computational cost, suitable for early-stage model selection.
- **Deterministic:** No stochastic elements or randomness.
- **Model-sensitive:** The proxy should reflect meaningful differences between models.

Proxy Search Space: The computation of the proxy can be represented as a **directed acyclic graph (DAG)**. For any given node, its in-degree must satisfy $0 \leq \text{degree}_{in} \leq 2$.

Nodes with only outgoing edges and no incoming edges are called **inputs**. Inputs consist of two parts: the input module and the input property. The input module refers to different types of network layers, and the input property is one of the following: gradient G , weights W , or output feature maps Z . An input is defined as a combination of these two components. For example, (convolutional layer + gradient G) refers to the gradients of the convolutional layer’s parameters. A proxy can have multiple inputs.

Nodes with both incoming and outgoing edges are referred to as **operations**, which are further divided into two types:

- When $\text{degree}_{in} = 1$, the node typically performs a **statistical operation** such as computing the mean or standard deviation of the preceding node’s output.
- When $\text{degree}_{in} = 2$, the node typically performs a **binary operation**, such as addition, subtraction, or division between two inputs.

Nodes with only incoming edges are **outputs**. There must be **only one output node** in the DAG. The input to this final node must be a **scalar**.

Search Space: The search space defines a set of possible architectural configurations derived from the supernet. Instead of fixed parameters, it provides discrete choices or ranges for each architectural hyperparameter, enabling the selection or evaluation of optimal subnetworks.

A typical search space includes the following parameters with explicit candidate values:

- **EMBED DIM:** A set of embedding dimensions ($D \in \{D_1, D_2, \dots, D_n\}$), e.g., ($D \in \{528, 576, 624\}$).
- **NUM HEADS:** Choices for attention heads ($H \in \{H_1, H_2, \dots, H_m\}$), e.g., ($H \in \{9, 10\}$).
- **MLP RATIO:** Options for the MLP expansion ratio ($R_{mlp} \in \{R_1, R_2, \dots, R_k\}$), e.g., ($R_{mlp} \in \{3.0, 3.5, 4.0\}$).
- **DEPTH:** Number of transformer blocks ($L \in \{L_1, L_2, \dots, L_s\}$), e.g., ($L \in \{14, 15, 16\}$).

Formally, the search space \mathcal{A} can be defined as a Cartesian product of discrete hyperparameter sets: $\mathcal{A} = \{(D, H, R_{mlp}, L) \mid D \in \{528, 576, 624\}, H \in \{9, 10\}, R_{mlp} \in \{3.0, 3.5, 4.0\}, L \in \{14, 15, 16\}\}$

Given Dataset: The architectures will be evaluated on an image classification task using **IMAGE-NET**. The input image shape is **(3, 224, 224)**, and the network outputs a **1000-dimensional vector**.

Note: Your Zero-Cost Proxies must not be the same as any existing ones (such as nwot, snip, etc.), though you may draw inspiration from them. Ensure numerical stability in your functions (avoid inf or nan during computation).

Figure 9: Prompt for mutation in AutoFormer.

Prompt for Mutation

Your Task: Given **5 existing Zero-Cost proxies**, **mutate** each of these proxies to create **5 new ones** (one mutated version per original proxy). Each mutated proxy should still be **training-free**, **computationally efficient**, **deterministic**, **input-aware**, and **model-sensitive**. The final goal is to produce **5 updated or extended training-free proxies** that remain valid for early-stage model selection but differ in approach from the original set.

When generating each mutated proxy, you may:

- Slightly alter or combine existing nodes in the computation DAG
- Introduce new operations or input types (still abiding by the in-degree constraints)
- Change the nature of the final scalar computation
- Ensure numerical stability (avoid division by zero, log of non-positive numbers, etc.)

Proxy Requirements:

- **Training-free:** Do **not** use any form of parameter updates.
- **Efficient:** Maintain low overhead in computation.
- **Deterministic:** Must produce consistent outputs for the same model and data.
- **Model-sensitive:** Should vary meaningfully with different subnetworks sampled from the supernet.

Proxy Search Space: All mutated proxies should still form a **directed acyclic graph (DAG)** in their computation. For any given node:

- **In-degree Constraint:** $0 \leq \text{degree}_{in} \leq 2$.
- **Input Nodes:** Consist of a network module. Tied to a property (W , G , or Z). You may now choose to introduce new input properties that are still relevant to a training-free scenario (e.g., a logarithm of weights, or a combined dimension of feature maps), but be sure to keep them deterministic and valid.
- **Operation Nodes (single-operand):** Perform statistical transformations (e.g., mean, variance, norms, etc.).
- **Operation Nodes (binary):** Perform pairwise operations (e.g., addition, multiplication, ratio).
- **Output Node:** Must be a single scalar value (with no outgoing edges).

Search Space: The search space defines a set of possible architectural configurations derived from the supernet. Instead of fixed parameters, it provides discrete choices or ranges for each architectural hyperparameter, enabling the selection or evaluation of optimal subnetworks.

A typical search space includes the following parameters with explicit candidate values:

- **EMBED DIM:** A set of embedding dimensions ($D \in \{D_1, D_2, \dots, D_n\}$), e.g., ($D \in \{528, 576, 624\}$).
- **NUM HEADS:** Choices for attention heads ($H \in \{H_1, H_2, \dots, H_m\}$), e.g., ($H \in \{9, 10\}$).
- **MLP RATIO:** Options for the MLP expansion ratio ($R_{mlp} \in \{R_1, R_2, \dots, R_k\}$), e.g., ($R_{mlp} \in \{3.0, 3.5, 4.0\}$).
- **DEPTH:** Number of transformer blocks ($L \in \{L_1, L_2, \dots, L_s\}$), e.g., ($L \in \{14, 15, 16\}$).

Formally, the search space \mathcal{A} can be defined as a Cartesian product of discrete hyperparameter sets: $\mathcal{A} = \{(D, H, R_{mlp}, L) \mid D \in \{528, 576, 624\}, H \in \{9, 10\}, R_{mlp} \in \{3.0, 3.5, 4.0\}, L \in \{14, 15, 16\}\}$

Given Dataset: The architectures will be evaluated on an image classification task using **IMAGE-NET**. The input image shape is **(3, 224, 224)**, and the network outputs a **1000-dimensional vector**.

Note: Your Zero-Cost Proxies must not be the same as any existing ones (such as nwt, snip, etc.), though you may draw inspiration from them. Ensure numerical stability in your functions (avoid inf or nan during computation).

Existing Proxies:
<Existing Proxies>

Figure 10: Prompt for crossover in AutoFormer.

Prompt for Crossover

Your Task: Please design **5 novel Zero-Cost Proxies** to evaluate the representation capability of different transformer network architectures on a given dataset, **using a crossover-based approach**. This means each of your new proxies should be created by **combining at least two distinct ideas, components, or sub-nodes** from hypothetical or existing parent proxies. For instance, if you have two parent proxies A and B with specific inputs or operations, you must form a new child proxy by **merging** relevant parts from each parent, ensuring the final proxy remains logically consistent and provides a **scalar score** that reflects the performance of a given network. You should generate Zero-Cost Proxies according to the following specifications, which include: **proxy requirements, description of the proxy search space, the search space of the networks being evaluated, the given dataset, and the output format**, as described below:

Proxy Requirements:

- **Training-free:** Do **not** use any form of parameter updates.
- **Efficient:** Maintain low overhead in computation.
- **Deterministic:** Must produce consistent outputs for the same model and data.
- **Model-sensitive:** Should vary meaningfully with different subnetworks sampled from the supernet.
- **Crossover-based:** Each proxy must **inherit or merge** conceptual “genes” (e.g., particular input sources or operations) from **at least two parent proxies**, forming a new, valid proxy design.

Proxy Search Space: All mutated proxies should still form a **directed acyclic graph (DAG)** in their computation. For any given node:

- **In-degree Constraint:** $0 \leq \text{degree}_{in} \leq 2$.
- **Input Nodes:** Consist of a network module. Tied to a property (W , G , or Z). You may now choose to introduce new input properties that are still relevant to a training-free scenario (e.g., a logarithm of weights, or a combined dimension of feature maps), but be sure to keep them deterministic and valid.
- **Operation Nodes (single-operand):** Perform statistical transformations (e.g., mean, variance, norms, etc.).
- **Operation Nodes (binary):** Perform pairwise operations (e.g., addition, multiplication, ratio).
- **Output Node:** Must be a single scalar value (with no outgoing edges).

Search Space: The search space defines a set of possible architectural configurations derived from the supernet. Instead of fixed parameters, it provides discrete choices or ranges for each architectural hyperparameter, enabling the selection or evaluation of optimal subnetworks.

A typical search space includes the following parameters with explicit candidate values:

- **EMBED DIM:** A set of embedding dimensions ($D \in \{D_1, D_2, \dots, D_n\}$), e.g., ($D \in \{528, 576, 624\}$).
- **NUM HEADS:** Choices for attention heads ($H \in \{H_1, H_2, \dots, H_m\}$), e.g., ($H \in \{9, 10\}$).
- **MLP RATIO:** Options for the MLP expansion ratio ($R_{mlp} \in \{R_1, R_2, \dots, R_k\}$), e.g., ($R_{mlp} \in \{3.0, 3.5, 4.0\}$).
- **DEPTH:** Number of transformer blocks ($L \in \{L_1, L_2, \dots, L_s\}$), e.g., ($L \in \{14, 15, 16\}$).

Formally, the search space \mathcal{A} can be defined as a Cartesian product of discrete hyperparameter sets: $\mathcal{A} = \{(D, H, R_{mlp}, L) \mid D \in \{528, 576, 624\}, H \in \{9, 10\}, R_{mlp} \in \{3.0, 3.5, 4.0\}, L \in \{14, 15, 16\}\}$

Given Dataset: The architectures will be evaluated on an image classification task using **IMAGE-NET**. The input image shape is **(3, 224, 224)**, and the network outputs a **1000-dimensional vector**.

Note: Your Zero-Cost Proxies must not be the same as any existing ones (such as nwot, snip, etc.), though you may draw inspiration from them. Ensure numerical stability in your functions (avoid inf or nan during computation).

Table 4: Average proxy correlation over 20 generations for different prompt operations on NAS-Bench-201. Full denotes the complete operation configuration, incorporating both mutation and crossover.

Generation	No Crossover	No Mutation	Full
1	0.444	0.193	0.246
2	0.533	0.237	0.470
3	0.533	0.349	0.617
4	0.533	0.439	0.628
5	0.541	0.463	0.662
6	0.541	0.463	0.662
7	0.541	0.463	0.705
8	0.572	0.463	0.705
9	0.627	0.547	0.754
10	0.640	0.547	0.794
11	0.640	0.547	0.800
12	0.640	0.669	0.800
13	0.640	0.713	0.800
14	0.640	0.761	0.800
15	0.640	0.761	0.800
16	0.680	0.780	0.800
17	0.714	0.780	0.803
18	0.739	0.780	0.803
19	0.739	0.780	0.813
20	0.758	0.780	0.813

Table 5: Effect of population size on the average Spearman correlation of the full prompt configuration over 20 generations.

Population Size	Avg. Spearman ρ
1	0.6023
2	0.6837
3	0.7541
4	0.8013
5	0.8124
10	0.7895

E Limitations

While APD markedly advances training-free NAS, several caveats remain. We group them here for clarity.

Black-box optimization Because APD delegates the generation of every candidate proxy to a LLM, the token-level reasoning that maps a prompt to executable code is not observable, raising black-box concerns. We acknowledge that the LLM itself remains opaque. nevertheless, APD is designed so that its outer optimization loop is inspectable. The LLM’s outputs are evaluated by a public fitness function, and the actor-critic scheduler is trained only on this scalar reward. All decisions that influence the search trajectory therefore pass through a measurable, task-specific signal rather than hidden logits. Thus, while some inner LLM reasoning remains inaccessible, the information APD exposes is already adequate for transparency and troubleshooting, so the residual opacity of LLM is unlikely to undermine the method’s value.

Need for a small ground-truth subset Computing fitness requires baseline accuracies for roughly 2-3% of each benchmark, which is standard in NAS studies and inexpensive compared with full training. Although a small ground-truth subset is indispensable for APD, we have empirically shown that the ZCPs obtained on one such subset continue to generalize when transferred to search space with no significant differences. Consequently, even for previously unseen NAS tasks, the overhead introduced by APD remains minimal and well within acceptable bounds.

Prompt-context length growth on very large searches The generator receives at most C_t proxies per step, and both prompt and context are explicitly bounded to fit the LLM window. Empirically the

Table 6: Effect of discount factor γ on average Spearman correlation. Best value is highlighted in yellow.

Discount Factor γ	Avg. Spearman ρ
0.50	0.7124
0.70	0.7823
0.90	0.8137
0.99	0.8015

Table 7: Effect of history window size on average Spearman correlation. Best value is highlighted in yellow.

History Window	Avg. Spearman ρ
1	0.6987
3	0.8042
5	0.8259
10	0.7791

combined length stayed under 6k tokens in specific population size, and we have to down-sample the context if future domains require longer code.

Security and stability when executing LLM-generated code Each proxy is arbitrary code and could, in principle, issue unsafe or resource-hungry calls. Therefore, code have to be run inside a resource-capped sandbox.

F Discovered AZPs

F.1 Proxy for NAS-Bench-201

To demonstrate the effectiveness of our approach, we present three representative proxies that achieved strong performance on NAS-Bench-201, which are illustrated in Figures 11, 12, and 13.

F.2 Proxy for TransNAS-Bench-101

We present the zero-cost proxies discovered on TransNAS-Bench-101 by APD in Figure 14, 15, and 16.

F.3 Proxy for AutoFormer

In Figure 17, 18, and 19, we present the zero-cost proxies discovered through APD on AutoFormer.

G Searched Architectures

In the DARTS search space, the optimal architecture discovered by APD is shown in Figure 22, with its normal and reduction cells visualized in Figures 20 and 21, respectively. In the AutoFormer setting, the architectures found for the tiny, small, and base variants are presented in Figure 23, 24, and 25, respectively.

H Related Work

H.1 Neural Architecture Search

Neural Architecture Search (NAS) aims to seek high-performance neural networks in an automatic manner. Early works cast the problem as black-box reinforcement learning or evolutionary optimization. NASNet Zoph and Le [2017] and AmoebaNet Real et al. [2019] trained thousands of architectures from scratch on large GPU clusters, demonstrating the promise of search but at prohibitive cost. Subsequent one-shot methods reduce cost by sharing parameters into a single supernet.

Table 8: Impact of A2C network depth on proxy search quality and convergence.

Hidden Layers	Final Spearman ρ	Convergence Generation
2	0.8216	42
3	0.8367	79
4	0.8175	96
5	0.8230	103

Table 9: Performance comparison on NAS-Bench-201 (NB-201) search space with CIFAR-10/100, and ImageNet16-120 datasets, and NAS-Bench-101 (NB-101) search space with CIFAR-10 dataset. We report Kendall’s τ (KT) and Spearman’s ρ (SPR) computed with all candidate architectures. In addition, we report the average independent runs results obtained from our method. The red, blue, and orange indicate the best, second-best, and third-best results, respectively.

Method	CIFAR-10(NB-201)			CIFAR-100(NB-201)			ImageNet16-120(NB-201)			CIFAR10(NB-101)		Runtime (ms/arch)
	SPR	KT	Test acc.(%)	SPR	KT	Test acc.(%)	SPR	KT	Test acc.(%)	SPR	Test acc.(%)	
Params	0.751	0.576	93.61 \pm 0.08	0.726	0.552	70.95 \pm 0.37	0.690	0.519	41.67 \pm 0.64	0.38	92.18 \pm 1.53	-
FLOPs	0.733	0.541	93.61 \pm 0.08	0.708	0.517	70.95 \pm 0.37	0.691	0.517	41.67 \pm 0.64	0.67	92.18 \pm 1.53	-
Snip Abdelfattah et al. [2021]	0.614	0.455	86.63 \pm 3.89	0.618	0.461	56.53 \pm 7.56	0.545	0.409	15.13 \pm 10.86	0.71	85.41 \pm 1.38	45.78
GraSP Abdelfattah et al. [2021]	0.460	0.318	88.01 \pm 3.14	0.470	0.329	61.43 \pm 7.04	0.406	0.282	30.85 \pm 5.66	0.45	87.08 \pm 1.40	93.70
Synflow Abdelfattah et al. [2021]	0.769	0.571	93.67 \pm 0.39	0.758	0.562	71.70 \pm 0.94	0.745	0.553	43.39 \pm 3.21	0.38	89.93 \pm 2.97	78.26
NWOT Mellor et al. [2021]	0.743	0.557	91.95 \pm 1.29	0.769	0.579	68.88 \pm 1.60	0.760	0.573	42.31 \pm 3.43	0.32	93.16 \pm 0.36	36.54
ZenNAS Lin et al. [2021]	0.365	0.283	89.55 \pm 1.12	0.338	0.245	64.69 \pm 3.86	0.372	0.260	37.18 \pm 3.17	0.65	93.06 \pm 0.73	30.07
ZiCo Li et al. [2023]	0.784	0.589	93.69 \pm 0.07	0.813	0.620	70.63 \pm 1.08	0.804	0.614	41.42 \pm 0.97	0.65	92.64 \pm 0.99	75.50
AZ-NAS Lee and Ham [2024]	0.913	0.741	93.49 \pm 0.30	0.900	0.723	70.33 \pm 1.16	0.886	0.710	44.34 \pm 1.26	0.42	92.01 \pm 0.85	69.43
SWAP Peng et al. [2024]	0.810	0.634	90.48 \pm 0.94	0.820	0.649	67.13 \pm 1.83	0.774	0.610	35.40 \pm 3.96	0.44	90.51 \pm 2.08	47.61
APD	0.832	0.635	93.76 \pm 0.09 (0.07)	0.843	0.654	72.22 \pm 0.65 (0.52)	0.817	0.633	45.03 \pm 0.76 (0.69)	0.73	93.49 \pm 0.34 (0.33)	16.81 (13.26)
Optimal	-	-	94.33 \pm 0.08	-	-	73.30 \pm 0.20	-	-	46.97 \pm 0.19	-	93.87 \pm 0.08	-

ENAS Zhou et al. [2020] introduces weight sharing with a recurrent controller, while DARTS Liu et al. [2018] relaxes the discrete search space to a differentiable mixture, enabling gradient-based updates but inheriting optimization bias and collapse issues that spawned a line of robust variants. However, performance estimation remains the key throughput bottleneck Mellor et al. [2021].

H.2 Zero-Cost Proxies

NAS traditionally relies on partial or full training to judge candidate models, making large-scale exploration prohibitive. Zero-Cost Proxies (ZCPs) address this bottleneck by estimating an architecture’s potential from its randomly initialized weights in significantly less time and constitute a standard component of training-free NAS. The earliest ZCPs exploit gradient saliency. SNIP Lee et al. evaluates weights with the first-order loss sensitivities, GraSP Abdelfattah et al. [2021] extends the idea to second-order information, and SynFlow Abdelfattah et al. [2021] removes data dependence by propagating an all-ones input, yielding strong correlations in deep convolutional spaces. A second strand replaces gradients with cheap statistics of activations or parameter layouts. NWOT Mellor et al. [2021] measures neural-tangent-kernel overlap, Zen-score Lin et al. [2021] evaluates Jacobian log-determinants, while ZiCo Li et al. [2023] shows that even simple norms such as parameter count or FLOPs can outperform many hand-crafted heuristics once appropriately normalized.

Despite steady progress, two limitations persist. Firstly, all existing proxies are manually specified. Their functional form rarely transfers intact across tasks (e.g., from NAS-Bench-201 CIFAR10 to TransNAS-Bench-101 autoencoding). Secondly, correlation improvements have largely plateaued, and gains often come at the cost of additional forward passes or Jacobian computations.

H.3 LLMs + Heuristic Evolution

Designing problem-specific heuristics has long been framed as a search problem in its own right. Early hyper-heuristic studies define a meta-level search over spaces of low-level rules, using evolutionary operators or tabu search Glover and Laguna [1998] to compose scheduling and graph-coloring heuristics. Genetic programming variants evolved symbolic programs that act as constructive or improvement heuristics, with recent multitask formulations sharing code fragments across optimization domains to boost sample efficiency Durasevic et al. [2025].

The advent of large language models (LLMs) has expanded heuristic evolution from symbolic trees to natural language "thought + code" artifacts. FunSearch Romera-Paredes et al. [2024] first demonstrated that LLMs augmented with self-consistency can rediscover classic algorithms, but required evaluation of thousands of candidates. Evolution of Heuristics (EoH) Liu et al. [2024] frames

Table 10: Evolution of average Spearman correlation over 20 generations on NAS-Bench-201 for different LLM backbones.

Gen.	Claude 3.7	Gemini 2.0 Flash	GPT-4o	Deepseek V3	Qwen Plus	Grok 3	Llama4
1	0.0000	0.3460	0.0989	0.0853	0.0228	0.1293	0.0000
2	0.5040	0.4290	0.1091	0.1720	0.0732	0.1293	0.1319
3	0.5040	0.5027	0.1816	0.1720	0.1292	0.1293	0.1319
4	0.5040	0.5560	0.3909	0.1720	0.3230	0.1293	0.1319
5	0.6553	0.5560	0.4312	0.2126	0.3807	0.1293	0.1319
6	0.7087	0.6033	0.4312	0.2126	0.3807	0.1293	0.1319
7	0.7304	0.6212	0.5036	0.3189	0.3962	0.2669	0.1319
8	0.7304	0.6714	0.6028	0.4923	0.5244	0.2669	0.1319
9	0.7304	0.6714	0.6278	0.4969	0.5518	0.2669	0.1319
10	0.7304	0.6714	0.6806	0.6531	0.6109	0.4071	0.1605
11	0.7713	0.6869	0.7411	0.6531	0.6303	0.6933	0.1605
12	0.7713	0.6869	0.7411	0.6531	0.6303	0.6933	0.1605
13	0.7713	0.6869	0.7744	0.7944	0.6303	0.6933	0.2809
14	0.7713	0.6986	0.7898	0.7944	0.6303	0.7081	0.2809
15	0.7728	0.7317	0.7898	0.7944	0.6439	0.7395	0.3011
16	0.7728	0.7317	0.8090	0.8024	0.6439	0.7395	0.4225
17	0.8114	0.7317	0.8090	0.8024	0.6994	0.7395	0.4348
18	0.8114	0.7517	0.8090	0.8024	0.7212	0.7395	0.5598
19	0.8114	0.7517	0.8090	0.8024	0.7212	0.7697	0.6495
20	0.8114	0.7522	0.8110	0.8024	0.7212	0.7697	0.6973

Table 11: The comparison results on the Autoformer search space in the ImageNet dataset. * denotes the results reported by Heo et al. [2021].

Models	#Param (M)	FLOPS (B)	Top-1 (%)	Top-5 (%)	Model Type	Design Type	Years	GPU Days
Tiny search space								
ResNet-18* He et al. [2016]	11.7	1.8	72.5	-	CNNs	Manual	2015	-
MobileNet-V3 Sandler et al. [2018]	5.5	-	75.2	-	CNNs	Manual	2015	-
DeiT-Ti Touvron et al. [2021]	5.7	1.2	72.2	91.1	Transformer	Manual	2015	-
TNT-Ti Han et al. [2021]	6.1	1.4	73.9	91.9	Transformer	Manual	2015	-
ViT-Ti Dosovitskiy et al. [2020]	5.7	-	74.5	-	Transformer	Manual	ICLR2020	-
CPVT-Ti Chu et al. [2021]	6.0	-	74.9	92.6	Transformer	Manual	2015	-
PVT-Tiny Wang et al. [2021]	13.2	1.9	75.1	-	Transformer	Manual	2015	-
AutoFormer-Ti Chen et al. [2021]	5.7	1.3	74.7	92.6	Transformer	Auto	CVPR2021	24
GLiT-Ti Chen et al. [2021]	7.2	1.4	76.3	-	Hybrid	Auto	ICCV2021	N/A
ViTAS-C Su et al. [2022]	5.6	1.3	74.7	91.6	Transformer	Auto	ECCV2022	32
TF-TAS-Ti Zhou et al. [2022]	5.9	1.4	75.3	92.8	Transformer	Auto	CVPR2022	0.5
Auto-Prox Wei et al. [2024]	6.4	-	75.6	-	Transformer	Auto	AAAI2024	0.1
AZ-NAS Lee and Ham [2024]	6.2	1.4	76.4	-	Transformer	Auto	CVPR2024	0.04
CET-TAS (Ours)	8.4	1.9	76.1	93.1	Transformer	Auto	-	0.25
Small search space								
ResNet-50* He et al. [2016]	25.6	4.1	80.2	-	CNNs	Manual	2015	-
RegNetY-4GF He et al. [2019]	20.6	-	79.4	-	CNNs	Manual	2015	-
DeiT-S Touvron et al. [2021]	22.1	4.7	79.9	95.0	Transformer	Manual	2015	-
ViT-S/16 Dosovitskiy et al. [2020]	22.1	4.7	78.8	-	Transformer	Manual	2015	-
PVT-Small Wang et al. [2021]	24.5	3.8	79.8	-	Transformer	Manual	2015	-
Swin-T Liu et al. [2021]	29.0	4.5	81.3	-	Transformer	Manual	2015	-
TNT-S Han et al. [2021]	23.8	5.2	81.5	95.7	Transformer	Manual	2015	-
CPVT-S Chu et al. [2021]	23.0	-	81.5	95.7	Transformer	Manual	2015	-
T2T-ViT ₁₋₁₄ Yuan et al. [2021]	21.5	-	81.7	-	Transformer	Manual	2015	-
AutoFormer-S Chen et al. [2021]	22.9	5.1	81.7	95.7	Transformer	Auto	CVPR2021	24
GLiT-S Chen et al. [2021]	24.6	4.4	80.5	-	Hybrid	Auto	ICCV2021	N/A
ViTAS-F Su et al. [2022]	27.6	6.0	80.5	95.1	Transformer	Auto	ECCV2022	32
TF-TAS-S Zhou et al. [2022]	22.8	5.0	81.9	95.8	Transformer	Auto	CVPR2022	0.5
AZ-NAS Lee and Ham [2024]	23.8	5.1	82.2	-	Transformer	Auto	CVPR2024	0.07
CET-TAS (Ours)	30.9	6.3	81.5	95.3	Transformer	Auto	-	0.25

heuristic discovery as an edit-based evolutionary loop driven by a large language model. Crucially, because the edit-based representation is task-agnostic, swapping the evaluation harness suffices to retarget the search. We leverage that property when evolving analytic ZCPs for NAS.

Despite rapid progress, existing Heuristic Evolution frameworks still evaluate each candidate by executing it on full problem instances and remain largely confined to classical, well-studied problems (e.g. TSP, online bin packing), which in turn constrains their practical applicability and masks their potential for more complicated, real-world-oriented downstream tasks.

H.4 LLMs for Downstream Tasks

LLMs are increasingly embedded inside end-to-end learning pipelines, where their role is to design data, signals or intermediate artifacts that materially improve a difficult scientific task rather than merely emit text. LLM-AutoDA Wang et al. [2024] treats long-tailed recognition as a reinforcement learning problem over the space of augmentation policies. Beyond data augmentation, LLMs have also been enlisted as planners inside vision and robotics systems. AntGPT Zhao et al. [2024]

Figure 11: zero-cost proxy1 discovered by APD in the NAS-Bench-201.

```
import torch
import torch.nn as nn

def proxy1(model, inputs, targets):
    bn_ranks = []
    ratios = []
    hooks = []

    def bn_hook(module, inp, out):
        if isinstance(out, torch.Tensor):
            B, C, H, W = out.shape
            mat = out.view(B, C, -1).permute(0, 2, 1).reshape(-1, C)
            frob_norm = torch.linalg.matrix_norm(mat, ord='fro')**2
            spec_norm = torch.linalg.matrix_norm(mat, ord=2)**2
            stable_rank = frob_norm / (spec_norm + 1e-6)
            bn_ranks.append(stable_rank.mean())

    for layer in model.modules():
        if isinstance(layer, nn.BatchNorm2d):
            hooks.append(layer.register_forward_hook(bn_hook))
        elif isinstance(layer, nn.Conv2d):
            weights = layer.weight
            l1_norm = weights.abs().sum(dim=(1,2,3)).mean()
            l2_norm = weights.norm(p=2, dim=(1,2,3)).mean()
            ratios.append((l1_norm / l2_norm).item())

    with torch.no_grad():
        model(inputs)

    for hook in hooks:
        hook.remove()

    bn_sum = torch.stack(bn_ranks).sum().item() if bn_ranks else 0.0
    ratio_sum = sum(ratios) if ratios else 0.0

    return bn_sum * ratio_sum
```

formulates long-term action anticipation as a language conditioned planning task. A video encoder produces current action tokens, and an LLM rolls out future verb-noun sequences by leveraging procedural knowledge learned from recipe corpora.

Prior work has shown that LLMs can be leveraged to great effect in demanding, domain-specific downstream tasks. Yet the accompanying frameworks are almost always built around bespoke, task-dependent pipelines whose design assumptions are tightly coupled to a single application, leaving no transferable paradigm that can be deployed on qualitatively different tasks without extensive re-engineering.

H.5 Automatic Proxy Discovery

Automatic Proxy Discovery aims to relieve experts from hand-crafting ZCPs by treating the proxy itself as the search object. The most systematic effort so far is Auto-Prox Wei et al. [2024]. It encodes a proxy as a computation graph whose nodes are primitive arithmetic or operations (e.g., exp, log). An evolutionary algorithm with elitism preservation mutates these graphs to maximize a joint correlation objective. Auto-Prox, however, is specialized to ViTs. Its search space hard-codes transformer-specific primitives. Similar variants, such as EZNAS Akhauri et al. [2022] and Auto-DAS Sun et al. [2024] follow identical graph enumeration but remain tied to fixed settings, respectively, limiting cross-domain portability and placing heavy reliance on human expertise.

Figure 12: zero-cost proxy2 discovered by APD in the NAS-Bench-201.

```
import torch
import torch.nn as nn

def proxy2(model, inputs, targets):
    ratios = []
    spatial_stds = []

    activations = []
    def hook_fn(module, input, output):
        activations.append((module.weight.detach(), output.detach()))

    hooks = []
    for layer in model.modules():
        if isinstance(layer, nn.Conv2d):
            hooks.append(layer.register_forward_hook(hook_fn))

    with torch.no_grad():
        model(inputs)

    for hook in hooks:
        hook.remove()

    for weight, act in activations:
        weight_norm = torch.norm(weight, p='fro')
        act_norm = torch.norm(act, p='fro')
        if act_norm != 0:
            ratios.append((weight_norm / act_norm).item())

        spatial_std = act.std(dim=1, keepdim=True)
        spatial_stds.append(spatial_std.mean().item())

    avg_ratio = sum(ratios) / len(ratios) if ratios else 0.0
    avg_std = sum(spatial_stds) / len(spatial_stds) if spatial_stds
        else 0.0
    return -2 * (avg_ratio * avg_std) / (avg_ratio + avg_std + 1e-10)
```

Figure 13: zero-cost proxy3 discovered by APD in the NAS-Bench-201.

```
import torch
import torch.nn as nn

def proxy3(model, inputs, targets):
    activations = []

    def hook_cnn(module, input, output):
        activations.append(output.detach().flatten(2))

    hooks = []
    for layer in model.modules():
        if isinstance(layer, nn.Conv2d):
            hooks.append(layer.register_forward_hook(hook_cnn))

    with torch.no_grad():
        _ = model(inputs)

    for h in hooks:
        h.remove()

    scores = []
    for act in activations:
        act = act - act.mean(dim=-1, keepdim=True)
        channel_mean = act.abs().mean(dim=(0, 2))
        if channel_mean.mean() > 1e-6:
            cv = channel_mean.std() / channel_mean.mean()
            scores.append(cv.item())

    return -sum(scores) / len(scores) if scores else 0.0
```

Figure 14: zero-cost proxy1 discovered by APD in the TransNAS-Bench-101.

```
import torch
import torch.nn.functional as F

def proxy1(model, inputs, targets):
    inputs.requires_grad_(True)

    with torch.enable_grad():
        output = model(inputs)
        loss = F.cross_entropy(output, targets)
        grad_clean = torch.autograd.grad(loss, inputs)[0]

        output = model(inputs + 0.01)
        loss = F.cross_entropy(output, targets)
        grad_noisy = torch.autograd.grad(loss, inputs)[0]

    inputs.requires_grad_(False)
    cos_sim = F.cosine_similarity(grad_clean.flatten(), grad_noisy.
        flatten(), dim=0)
    return cos_sim.cpu().item()
    return score.cpu()
```

Figure 15: zero-cost proxy2 discovered by APD in the TransNAS-Bench-101.

```
import torch

def proxy2(model, inputs, targets):
    with torch.no_grad():
        clean_output = model(inputs)
        scaled_inputs = inputs * 1.01 # Small fixed scaling
        scaled_output = model(scaled_inputs)

    logit_diff = torch.norm(clean_output - scaled_output, p=2, dim=1)
    .mean()
    return logit_diff.cpu().item()
```

Figure 16: zero-cost proxy3 discovered by APD in the TransNAS-Bench-101.

```
import torch
import torch.nn.functional as F

def proxy3(model, inputs, targets):
    with torch.no_grad():
        clean_output = model(inputs)
        noise = torch.zeros_like(inputs)
        for i in range(inputs.shape[2]):
            for j in range(inputs.shape[3]):
                noise[:, :, i, j] = 0.01 * ((i + j) % 2 * 2 - 1)
        noisy_output = model(inputs + noise)

    clean_probs = F.softmax(clean_output, dim=1)
    noisy_probs = F.softmax(noisy_output, dim=1)

    kl_div = F.kl_div(clean_probs.log(), noisy_probs, reduction='
    batchmean')
    return kl_div.cpu().item()
    return score.cpu()
```

Figure 17: zero-cost proxy1 discovered by APD in the AutoFormer.

```
import torch
import torch.nn.functional as F

def heuristic_5(model, inputs, targets, loss_fn):
    with torch.no_grad():
        clean_output = model(inputs)
        noise_levels = [0.01, 0.05, 0.1]

        kl_values = []
        clean_probs = F.softmax(clean_output, dim=1)

        for level in noise_levels:
            noise = torch.randn_like(inputs) * level
            noisy_output = model(inputs + noise)
            noisy_probs = F.softmax(noisy_output, dim=1)
            kl_div = F.kl_div(clean_probs.log(), noisy_probs,
                reduction='batchmean')
            kl_values.append(kl_div)

        score = torch.prod(torch.stack(kl_values)) ** (1/len(
            kl_values))
    return score.cpu().item()
```

Figure 18: zero-cost proxy2 discovered by APD in the AutoFormer.

```
import torch
import torch.nn.functional as F

def proxy2(model, inputs, targets, loss_fn):
    with torch.no_grad():
        # Input perturbation component
        noise = torch.randn_like(inputs) * 0.01
        perturbed_inputs = inputs + noise
        original_outputs = model(inputs)
        perturbed_outputs = model(perturbed_inputs)
        diff = torch.norm(original_outputs - perturbed_outputs, p=2,
                           dim=1).mean()

        # KL divergence component
        original_softmax = original_outputs.softmax(dim=1)
        perturbed_softmax = perturbed_outputs.softmax(dim=1)
        kl_div = F.kl_div(original_softmax.log(), perturbed_softmax,
                           reduction='batchmean')

    return (diff + kl_div).cpu().item()
```

Figure 19: zero-cost proxy3 discovered by APD in the AutoFormer.

```
import torch

def proxy2(model, inputs, targets):
    with torch.no_grad():
        clean_output = model(inputs)
        scaled_inputs = inputs * 1.01 # Small fixed scaling
        scaled_output = model(scaled_inputs)

    logit_diff = torch.norm(clean_output - scaled_output, p=2, dim=1)
    .mean()
    return logit_diff.cpu().item()
```

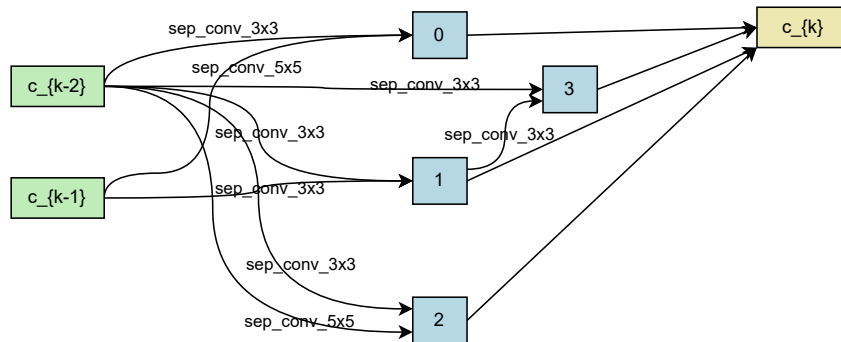


Figure 20: Normal cell searched by APD.

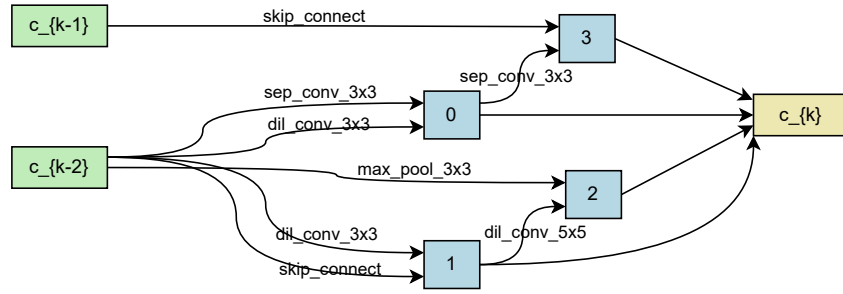


Figure 21: Reduction cell searched by APD.

Figure 22: Optimal structure found in the DARTS search space by APD.

```
Genotype(normal=[('sep_conv_3x3', 0), ('sep_conv_5x5', 1),
('sep_conv_3x3', 0), ('sep_conv_3x3', 1),
('sep_conv_5x5', 0), ('sep_conv_5x5', 0),
('sep_conv_3x3', 3), ('sep_conv_3x3', 0)], normal_concat=range(2, 6),
reduce=[('sep_conv_3x3', 0), ('dil_conv_3x3', 0),
('dil_conv_3x3', 0), ('skip_connect', 0),
('max_pool_3x3', 0), ('dil_conv_5x5', 3),
('sep_conv_3x3', 2), ('skip_connect', 1)], reduce_concat=range(2, 6))
```

Figure 23: Subnet discovered in the AutoFormer tiny search space by APD.

```
RETRAIN:
MLP_RATIO:
- 4
- 3.5
- 3.5
- 3.5
- 4
- 4
- 4
- 4
- 4
- 3.5
- 3.5
- 3.5
NUM_HEADS:
- 4
- 3
- 4
- 4
- 4
- 4
- 4
- 4
- 4
- 3
- 3
- 3
- 4
DEPTH: 12
EMBED_DIM: 240
```

Figure 24: Subnet discovered in the AutoFormer small search space by APD.

```
RETRAIN :  
  MLP_RATIO :  
    - 3.5  
    - 3.0  
    - 4.0  
    - 4.0  
    - 4.0  
    - 3.0  
    - 3.0  
    - 3.0  
    - 4.0  
    - 4.0  
    - 4.0  
    - 3.0  
    - 4.0  
    - 3.0  
  NUM_HEADS :  
    - 7  
    - 6  
    - 7  
    - 5  
    - 7  
    - 7  
    - 6  
    - 7  
    - 7  
    - 6  
    - 5  
    - 5  
    - 7  
    - 5  
  DEPTH: 14  
  EMBED_DIM: 448
```

Figure 25: Subnet discovered in the AutoFormer base search space by APD.

```
RETRAIN :
MLP_RATIO :
- 3.0
- 3.0
- 3.0
- 4.0
- 4.0
- 3.5
- 3.0
- 4.0
- 3.5
- 3.0
- 4.0
- 3.0
- 3.0
- 4.0
- 3.0
NUM_HEADS :
- 9
- 10
- 9
- 10
- 10
- 9
- 10
- 9
- 9
- 10
- 9
- 9
- 10
- 9
- 10
- 9
DEPTH: 15
EMBED_DIM: 576
```


References

- [1] Mohamed S. Abdelfattah, Abhinav Mehrotra, Lukasz Dudziak, and Nicholas Donald Lane. Zero-cost proxies for lightweight NAS. In 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021.
- [2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.
- [3] Yash Akhauri, J. Pablo Munoz, Nilesch Jain, and Ravi Iyer. Eznas: Evolving zero cost proxies for neural architecture scoring. arXiv preprint arXiv:2209.07413, 2022. URL <https://arxiv.org/abs/2209.07413>.
- [4] Anthropic. Claude 3.7 Sonnet. <https://www.anthropic.com/news/claude-3-7-sonnet>, 2025.
- [5] Boyu Chen, Peixia Li, Chuming Li, Baopu Li, Lei Bai, Chen Lin, Ming Sun, Wanli Ouyang, et al. Glit: Neural architecture search for global and local image transformer. In ICCV, 2021.
- [6] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In International Conference on Computer Vision, 2021.
- [7] Xiangxiang Chu, Zhi Tian, Bo Zhang, Xinlong Wang, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Conditional positional encodings for vision transformers. Arxiv preprint 2102.10882, 2021. URL <https://arxiv.org/pdf/2102.10882.pdf>.
- [8] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, 2020.
- [9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In International Conference on Learning Representations, 2020.
- [10] Yawen Duan, Xin Chen, Hang Xu, Zewei Chen, Xiaodan Liang, Tong Zhang, and Zhenguo Li. Transnas-bench-101: Improving transferability and generalizability of cross-task neural architecture search. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 5251–5260, 2021.
- [11] Marko Durasevic, Mateja Dumic, and Francisco Javier Gil Gala. Multitask genetic programming for automated design of heuristics for the container relocation problem. Engineering Applications of Artificial Intelligence, 144:110001, 2025. doi: 10.1016/j.engappai.2025.110001.
- [12] Fred Glover and Manuel Laguna. Tabu Search. Springer, 1998.
- [13] Google Cloud. Gemini 2.0 Flash. <https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash>, 2025.
- [14] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. In NeurIPS, 2021.
- [15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778, 2016.
- [16] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In CVPR, 2019.
- [17] Byeongho Heo, Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Junsuk Choe, and Seong Joon Oh. Rethinking spatial dimensions of vision transformers. In ICCV, 2021.

- [18] Junghyup Lee and Bumsub Ham. Az-nas: Assembling zero-cost proxies for network architecture search. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 5893–5903, 2024.
- [19] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning based on connection sensitivity. In International Conference on Learning Representations.
- [20] Guihong Li, Yuedong Yang, Kartikeya Bhardwaj, and Radu Marculescu. Zico: Zero-shot nas via inverse coefficient of variation on gradients. arXiv preprint arXiv:2301.11300, 2023.
- [21] Ming Lin, Pichao Wang, Zhenhong Sun, Heseng Chen, Xiuyu Sun, Qi Qian, Hao Li, and Rong Jin. Zen-nas: A zero-shot NAS for high-performance image recognition. In 2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021, pages 337–346. IEEE, 2021. doi: 10.1109/ICCV48922.2021.00040.
- [22] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. arXiv preprint arXiv:2412.19437, 2024.
- [23] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In Proceedings of the 41st International Conference on Machine Learning, volume 202 of Proceedings of Machine Learning Research, pages 32201–32223. PMLR, 2024. URL <https://openreview.net/forum?id=H9aqvdXaf1>.
- [24] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. ArXiv, abs/1806.09055, 2018.
- [25] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In ICCV, 2021.
- [26] Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. In International Conference on Machine Learning, pages 7588–7598. PMLR, 2021.
- [27] Yameng Peng, Andy Song, Haytham M. Fayek, Vic Ciesielski, and Xiaojun Chang. SWAP-NAS: Sample-wise activation patterns for ultra-fast NAS. In The Twelfth International Conference on Learning Representations, 2024.
- [28] Qwen Team. Qwen2.5 technical report. arXiv preprint arXiv:2412.15115, 2024. URL <https://arxiv.org/abs/2412.15115>. Includes the Qwen-Plus Mixture-of-Experts variant.
- [29] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In AAAI Conference on Artificial Intelligence, volume 33, page 4780–4789, 2019.
- [30] Brais Romera-Paredes, Marjan Barekatain, Alexander Novikov, Milos Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. Ruiz, Jonathan S. Ellenberg, Peng Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. Nature, 625(7995): 468–475, 2024.
- [31] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4510–4520, 2018.
- [32] Xiu Su, Shan You, Jiyang Xie, Mingkai Zheng, Fei Wang, Chen Qian, Changshui Zhang, Xiaogang Wang, and Chang Xu. Vision transformer architecture search. In Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXI, page 139–157, 2022.
- [33] Haosen Sun, Lujun Li, Peijie Dong, Zimian Wei, and Shitong Shao. Auto-das: Automated proxy discovery for training-free distillation-aware architecture search. In Computer Vision – ECCV 2024, Lecture Notes in Computer Science, pages 56–73. Springer Cham, 2024. doi: 10.1007/978-3-031-72652-1_4.

- [34] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In ICML. PMLR, 2021.
- [35] Pengkun Wang, Zhe Zhao, Haibin Wen, Fanfu Wang, Binwu Wang, Qingfu Zhang, and Yang Wang. Llm-autoda: Large language model-driven automatic data augmentation for long-tailed problems. In Advances in Neural Information Processing Systems, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/7755fafa64beceac16d3eab17f8ed3d6-Abstract-Conference.html.
- [36] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In ICCV, 2021.
- [37] Zimian Wei, Peijie Dong, Zheng Hui, Anggeng Li, Lujun Li, Menglong Lu, Hengyue Pan, and Dongsheng Li. Auto-prox: Training-free vision transformer architecture search via automatic proxy discovery. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 38, pages 15814–15822, 2024.
- [38] xAI. Grok 3: The age of reasoning agents. <https://x.ai/news/grok-3>, 2025.
- [39] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. CARS: continuous evolution for efficient neural architecture search. In 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020, pages 1826–1835. IEEE, 2020.
- [40] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zihang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In ICCV, 2021.
- [41] Qi Zhao, Shijie Wang, Ce Zhang, Changcheng Fu, Minh Quan Do, Nakul Agarwal, Kwonjoon Lee, and Chen Sun. Antgpt: Can large language models help long-term action anticipation from videos? In International Conference on Learning Representations, 2024. URL <https://openreview.net/forum?id=Bb21JPnhhr>.
- [42] Dongzhan Zhou, Xinchu Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. Econas: Finding proxies for economical neural architecture search. In 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020, pages 11393–11401. IEEE, 2020.
- [43] Qinqin Zhou, Kekai Sheng, Xiawu Zheng, Ke Li, Xing Sun, Yonghong Tian, Jie Chen, and Rongrong Ji. Training-free transformer architecture search. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 10894–10903, 2022.
- [44] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In International Conference on Learning Representations (ICLR), 2017.