LLM AS A TOOL, NOT AN AGENT: CODE-MINED TREE TRANSFORMATIONS FOR NEURAL ARCHITECTURE SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural Architecture Search (NAS) aims to automatically discover high-performing deep neural network (DNN) architectures. However, conventional algorithm-driven NAS relies on carefully hand-crafted search spaces to ensure executability, which restricts open-ended exploration. Recent coding-based agentic approaches using large language models (LLMs) reduce manual design, but current LLMs struggle to reliably generate complex, valid architectures, and their proposals are often biased toward a narrow set of patterns observed in their training data. To bridge reliable algorithmic search with powerful LLM assistance, we propose LLMasTool, a hierarchical tree-based NAS framework for stable and open-ended model evolution. Our method automatically extracts reusable modules from arbitrary source code and represents full architectures as hierarchical trees, enabling evolution through reliable tree transformations rather than code generation. At each evolution step, coarse-level planning is governed by a diversity-guided algorithm that leverages Bayesian modeling to improve exploration efficiency, while the LLM resolves the remaining degrees of freedom to ensure a meaningful evolutionary trajectory and an executable generated architecture. With this formulation, instead of fully agentic LLM approaches, our method explores diverse directions beyond the inherent biases in the LLM. Our method improves over existing NAS methods by 0.69, 1.83, and 2.68 points on CIFAR-10, CIFAR-100, and ImageNet16-120, demonstrating its effectiveness.

1 INTRODUCTION

Designing high-performing neural architectures remains a key driver of progress in deep learning, yet often requires substantial expertise and trial-and-error. Neural Architecture Search (NAS) aims to automate this process by exploring candidate architectures and optimizing their validation performance. Early NAS demonstrated that reinforcement learning (Zoph & Le, 2017) and evolutionary strategies (Real et al., 2017) can discover competitive architectures, while cell-based search enabled reusable designs across networks (Zoph et al., 2018). Subsequent work improved search efficiency through parameter sharing (Pham et al., 2018), differentiable optimization (Liu et al., 2019b), and surrogate-guided search (Liu et al., 2018). However, these methods typically rely on handcrafted search spaces and primitives to guarantee executability, restricting open-ended exploration and embedding strong designer priors (Elsken et al., 2019).

Another line of work evolves existing architectures through structured edits. Network morphism reduces the cost of iterative modification by preserving network functions (Wei et al., 2016), while Auto-Keras combines structured edits with Bayesian optimization (Jin et al., 2019). Hierarchical decompositions further factorize architecture search across abstraction levels (Liu et al., 2019a; Qian et al., 2022). Nevertheless, when edits become fine-grained across module choices, connectivity, and hyperparameters, the branching factor grows combinatorially, making useful feedback increasingly sparse (Jin et al., 2019; Russo et al., 2018).

Recently, large language models (LLMs) have emerged as a promising tool for NAS. LLMs can propose meaningful architectural modifications and abstract large low-level action spaces into semantically coherent decisions (Chen et al., 2023; Zhou et al., 2025a). However, LLM-driven NAS

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

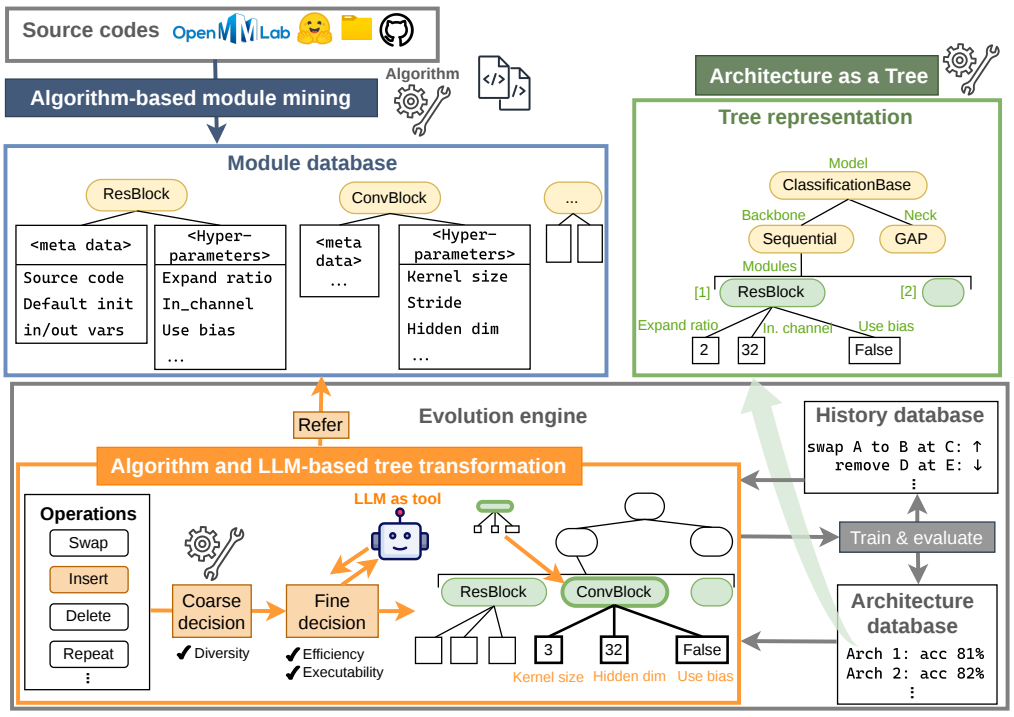

Figure 1: **Overview of the proposed NAS pipeline.** (Top-left) We mine reusable modules from source code to build a module database. (Top-right) architectures are represented as deploy-friendly hierarchical trees. (Bottom-left) an evolution engine applies controlled tree transformations, with an LLM serving as a tool for fine-grained refinement.

often suffers from limited controllability and difficulty guaranteeing executability, particularly when the LLM acts as an autonomous agent that generates architectures end-to-end (Yang et al., 2025b). Furthermore, LLM proposals are inevitably influenced by patterns in their training data, which can limit exploration beyond familiar designs. These observations motivate a different paradigm: rather than asking an LLM to autonomously generate architectures, we use it as a *tool* for localized decisions inside an algorithmically controlled and diversified search process.

In this work, we propose **LLMasTool**, an open-ended NAS framework that combines algorithmically diversity-guided evolution planning with LLM assistance. The key challenge is to provide a representation that is well-structured to be processed by algorithmic manipulation while expressive for LLM to understand. The source code representation is expressive for LLM but difficult to manipulate with algorithms. Existing NAS methods commonly use data-flow graphs, where features are represented as nodes and operations as edges. However, such representations are poorly aligned with how DNNs are implemented in source codes. In PyTorch (PyTorch), for example, models are expressed as hierarchically nested modules, with the data flow specified inside each module. Converting such implementations into data-flow graphs is non-trivial and requires intrusive rewriting.

We therefore represent architectures as *deploy-friendly hierarchical trees*, whose nodes correspond to models, modules, components, and hyperparameters. We further develop an algorithmic module-mining procedure that extracts reusable modules from arbitrary PyTorch source code and stores them in a module database. This enables LLMasTool to directly reuse existing implementations and research knowledge rather than relying on LLM memorization.

On top of this representation, architecture evolution is formulated as a controlled tree transformation rather than free-form code generation. To prevent the search from being dominated by LLM priors, we employ a hierarchical coarse-to-fine decision process. A diversity-guided algorithm first determines the coarse direction of an edit, while the LLM resolves the remaining fine-grained choices. Historical outcomes are accumulated at the upper levels of this hierarchy, allowing Bayesian Thompson sampling to guide exploration and exploitation despite the sparsity of feedback in the open-ended search space.

Experiments on CIFAR-10, CIFAR-100, ImageNet16-120, and ImageNet-100 demonstrate the effectiveness of LLMasTool. The contributions are summarized as follows:

- We formulate open-ended NAS as algorithm-driven exploration, with the LLM acting as a tool for localized design decisions rather than an autonomous architecture designer.
- We introduce a deploy-friendly hierarchical tree representation together with algorithmic module mining from source code, allowing architecture evolution to be formulated as executable tree transformations rather than unrestricted code generation.
- We propose a diversity-guided coarse-to-fine planning strategy and hierarchical Thompson sampling that effectively balance exploration and exploitation.
- Extensive experiments across four datasets demonstrate the effectiveness of the proposed formulation, while ablation studies show that reducing reliance on the LLM can lead to better architecture discovery.

2 Related Work

2.1 Neural Architecture Search (NAS)

NAS optimizes neural network architectures in a data-driven manner. Most approaches represent architectures as data-flow graphs and optimize node types, hyperparameters, and connectivity. Reinforcement learning (Zoph & Le, 2017; Zoph et al., 2018; Pham et al., 2018) and evolutionary algorithms (Suganuma et al., 2017; Liu et al., 2018; So et al., 2019) have been widely used for this purpose. Another major direction searches within predefined super-networks through differentiable edge selection (Liu et al., 2019b; Wu et al., 2019; Cai et al., 2019; Stamoulis et al., 2019; Zhang et al., 2024) or partial weight and block sampling (Yu et al., 2020; Cai et al., 2020; Zhao et al., 2021; Lu et al., 2023; Jeon et al., 2025; Oh et al., 2025; Chen et al., 2021a;b; Sakuma et al., 2024). GNN-based approaches have also been explored to improve architecture optimization efficiency (Mills et al., 2023; Salameh et al., 2023; Mills et al., 2024).

Although these methods can efficiently identify architectures within their search spaces, their predefined primitives and connectivity rules are mainly designed to ensure validity. This substantially restricts the diversity of architectures that can be explored.

2.2 LLMs for NAS

Recent work has investigated LLMs for both conventional and open-ended NAS. GENEIUS (Zheng et al., 2023) uses an LLM to improve evolutionary search within a conventional search space, while LAPT (Zhou et al., 2025b) uses an LLM to prune low-potential candidates. EvoPrompt (Chen et al., 2023) employs a coding LLM for mutation and crossover in open-ended NAS, and LLMatic (Nasir et al., 2024) further combines LLM-based evolution with quality-diversity optimization. Genesys (Cheng et al., 2025) explores coding-based NAS for language models, while APD (Kang et al., 2025) uses an LLM to propose zero-cost performance proxies. NADER (Yang et al., 2025b) extracts architectural ideas from recent papers and generates data-flow graphs rather than source code to reduce generation difficulty.

Despite their promise, LLM-based NAS methods still face challenges in controllability and executability, particularly when the LLM directly determines the complete architecture. Moreover, their exploration can remain strongly influenced by the LLM's prior knowledge. LLMasTool instead separates global algorithmic exploration from local LLM decision making, using an executable tree transformation framework to systematically explore a broader space.

3 Methodology

We introduce a NAS framework that represents DNN architectures as hierarchical trees and performs architecture evolution through tree transformations, while using an LLM as a tool rather than an autonomous architecture generator. As illustrated in Fig. 1, the framework consists of three components: (1) algorithmic module mining from existing source code, (2) a deploy-friendly hierarchical tree representation, and (3) diversity-guided coarse-to-fine evolution combining algorithmic decisions with LLM-assisted refinement.

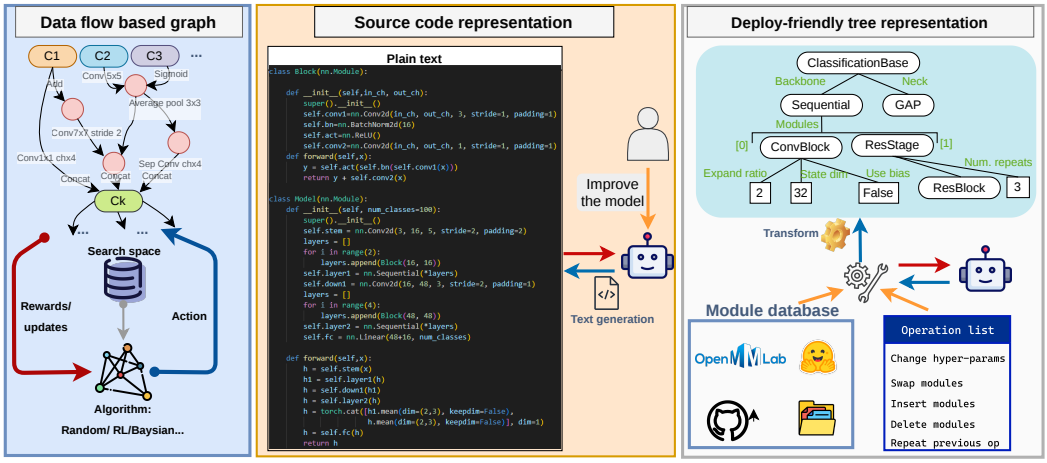

Figure 2: **Comparison of data-flow graph-, code-, and tree-based NAS representations.** Our deploy-friendly tree directly reflects the hierarchical structure of implemented DNN modules.

3.1 Module Mining from Existing Source Code

We first construct a module database $\mathcal{D}_M$ from existing PyTorch source code. The mining procedure is entirely rule-based and does not rely on an LLM.

Given arbitrary PyTorch source codes, we parse it into an Abstract Syntax Tree (AST) and identify all classes inheriting from `torch.nn.Module`. Each extracted module is stored together with its search space and metadata: $\mathcal{D}_M = (m_i, s_i, \mu_i)$, where m_i, s_i, and μ_i denote a module, its tunable search space, and its metadata.

The constructor arguments of each module are treated as tunable search space, allowing the search to modify properties such as channel width, kernel size, and internal modules. We additionally record default argument values, the number of input/output variables of the `forward` method, and the relevant source code as its metadata. These metadata provide the information required to instantiate and transform modules during evolution.

Because module extraction is performed directly from source code using deterministic AST analysis, the resulting module database is reproducible and can incorporate implementations from arbitrary PyTorch-based repositories without manual intervention.

3.2 Deploy-Friendly Tree Representation of DNN Architectures

Unlike conventional NAS methods that represent architectures as data-flow graphs (Liu et al., 2019b; 2018; Mills et al., 2023; Salameh et al., 2023), we represent an architecture as a deploy-friendly hierarchical tree, as shown in Fig. 2. Higher-level nodes represent conceptual components such as the model and backbone, while lower-level nodes represent modules and their constructor arguments. Module nodes can contain further submodules, whereas hyperparameters form leaf nodes.

The tree is extended with list structures to support variable-length sequential compositions, such as stacking an arbitrary number of layers. This representation closely matches how DNNs are implemented in common frameworks and therefore enables direct reuse of existing source-code.

The hierarchical structure also naturally supports top-down refinement: a coarse architecture can first be selected and then progressively modified at lower levels, which is difficult with data-flow based graph representation.

3.3 Architecture Evolution

Our evolutionary search maintains a module database $\mathcal{D}_M$, an architecture database $\mathcal{D}_A$, and a history database $\mathcal{D}_H$. The architecture database is defined as $\mathcal{D}_A = \{(A_i, e_i)\}_{i=1}^{N}$, where A_i and e_i denote the tree representation of a DNN architecture and the corresponding evaluation metric (e.g.,

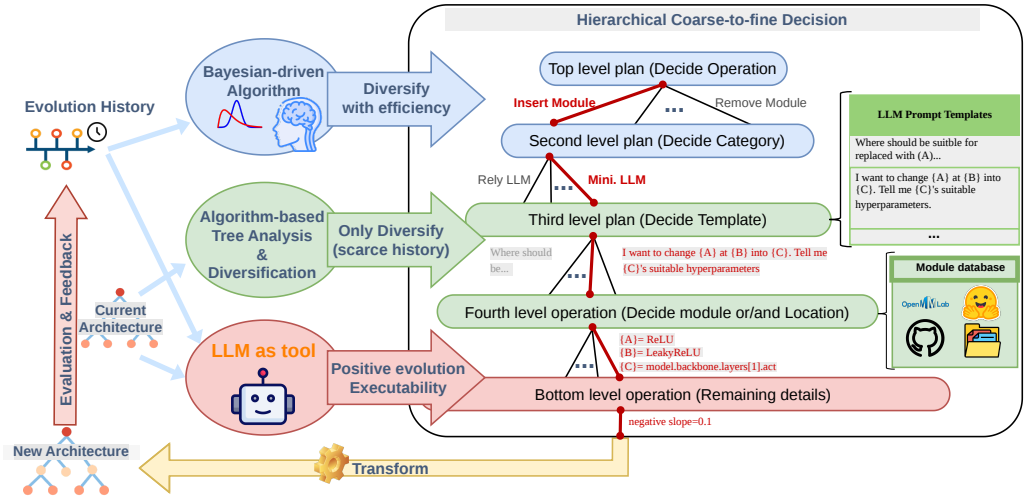

Figure 3: **Coarse-to-fine hierarchical decision process.** The high level balances exploration and exploitation using Bayesian sampling. The middle level enumerates candidate transformation directions algorithmically, while the low level uses the LLM to resolve the remaining degrees of freedom.

accuracy). At initialization, $\mathcal{D}_A$ contains at least one base architecture, which serves as the starting point of the evolution process.

At evolution step t, we sample a base architecture: $A_t^{base} \sim \mathcal{D}_A$, and determine a tree transformation

$$T_t \sim p_{cf}\left(T \mid A_t^{base}, \mathcal{D}_A, \mathcal{D}_M, \mathcal{D}_H, \mathcal{LLM}\right), \tag{1}$$

where $\mathcal{D}_H = \left\{(T_\tau, A_\tau^{base}, e_\tau - e_\tau^{base})\right\}_{\tau=1}^{t-1}$ is a history database that stores previous experimental results with tree transformation T_τ and metric improvement $e_\tau - e_\tau^{base}$. The p_{cf} represents the coarse-to-fine decision making discussed later. The decided transformation is applied to A_t^{base} as $A_t = T_t(A_t^{base})$. The resulting architecture A_t is evaluated by a feasibility function

$$\mathcal{E}(A_t^{base}, A_t, T_t) = \mathbf{1}_{\text{exec}(A_t)} \cdot \mathbf{1}_{\text{const}(A_t)} \cdot \mathbf{1}_{\text{intend}(A_t^{base}, A_t, T_t)}, \tag{2}$$

where $\mathbf{1}_{\text{exec}(A_t)}$, $\mathbf{1}_{\text{const}(A_t)}$, and $\mathbf{1}_{\text{intend}(A_t^{base}, A_t, T_t)}$ verify whether A_t is executable, A_t satisfies target constraints such as FLOPs and parameter budgets, and the intended transformation T_t is applied correctly. If all feasibility checks are verified, we instantiate the architecture tree into an actual DNN, conduct training and evaluation, and then store the results in $\mathcal{D}_A$ and $\mathcal{D}_H$ as

$$\mathcal{D}_A \leftarrow \mathcal{D}_A \cup \{(A_t, e_t)\}, \qquad \mathcal{D}_H \leftarrow \mathcal{D}_H \cup \{(T_t, A_t^{base}, e_t - e_t^{base})\}. \tag{3}$$

The overall evolution algorithm can be found in Algorithm 1.

In the following, we describe the details about the proposed evolution with tree transformation T and the coarse-to-fine decision making method p_{cf}.

Tree Transformation as an Evolution Step

We formulate architecture evolution as a tree transformation rather than source-code generation. This representation is compact, directly executable, and supports open-ended modifications. We define five operations for tree transformation:

- **Change Hyperparameter**: modify a leaf value, such as channel width or kernel size.
- **Swap Module**: replace a sub-tree with another module from $\mathcal{D}_M$.
- **Insert/Delete Module**: insert or remove a module at a list-structured position.
- **Create Module**: construct a composite sub-tree from multiple existing sub-trees.
- **Repeat Previous**: repeat a previously successful type of transformation.

The first four operations are the basic transformations $o \in \mathcal{O}_{basic}$, while **Repeat Previous** is a meta-operation that exploits previously successful directions. Given an operation, a concrete transformation requires determining where to edit and which module or value to use. The hierarchical

tree makes these choices explicit and allows many of them to be handled algorithmically, while the LLM resolves the remaining semantic decisions. The detailed algorithms of each operation are provided in Sec. D.2.

Coarse-to-Fine Decision Making using LLM as a Tool

At each evolution step, we must determine *what* operation to apply, *where* to apply it, and *how* to instantiate it. Rather than delegating all decisions to the LLM, we factorize this process hierarchically and adopt a coarse-to-fine planning strategy combined with a rule-based algorithm as shown in Fig. 3. The rule-based component determines the high-level structure of the transformation, while the LLM is used as a tool to determine a concrete tree transformation within the remaining degrees of freedom. Specifically, our hierarchical sampling flow is as follows: We **first** sample a tree transformation operation $o \sim p_1(o)$ from the basic tree transformation operations $o \in \mathcal{O}_{basic}$. For each operation, we prepare three categories of prompt templates $c \in \mathcal{C}$ (as shown in Fig. 3): relying heavily on the prior knowledge of the LLM (*rely LLM*), exploring opposed to the knowledge (inv. LLM), and focusing on diversity with minimal reliance on LLM (*mini. LLM*). The **second** sampling picks up one of these three categories $c \sim p_2(c \mid o)$, and the **third** sampling picks up a specific prompt template $\tau \in \mathcal{T}_c$ from the selected category $\tau \sim p_3(\tau \mid c, o)$. As illustrated in Fig. 3, each prompt template contains placeholders such as $\{\mathrm{A}\}$ and $\{\mathrm{B}\}$. In the **fourth** sampling, to fill these placeholders, a rule-based algorithm retrieves all the corresponding node addresses or module names from the architecture tree A_t^{base} and the module database $\mathcal{D}_M$. Then, it samples them as $\phi \sim p_4\big(\phi \mid c, o, \tau, A_t^{base}, \mathcal{D}_M)\big)$ where $\phi = (\mathrm{A}, \mathrm{B}, ...)$. The p_1 to p_4 have determined the coarse-grained aspects of the transformation at multiple levels of granularity. We then use LLM as a tool to create a complete transformation that generates a valid and executable architecture. Specifically, we construct a complete prompt by combining the prompt template specified by p_1–p_4 with relevant metadata retrieved from the history and module databases. The resulting prompt is then provided to the LLM to generate the complete transformation T according to $T \sim p_5(T \mid c, o, \tau, \phi, A_t^{base}, \mathcal{D}_M, \mathcal{D}_H)$. The details of the prompt templates and the resulting overall prompts are described in Sec. D.4.

In short, our coarse-to-fine decision making is

$$p_{cf}(T \mid A_t^{base}, \mathcal{D}_M, \mathcal{D}_H) = \sum_{o,c,\tau,\phi} p_5(T \mid c, o, \tau, \phi, \cdot)\, p_4(\phi \mid c, o, \tau, \cdot)\, p_3(\tau \mid c, o)\, p_2(c \mid o)\, p_1(o). \tag{4}$$

This decomposition has another merit. Although the complete transformation space is too large and sparse for each concrete decision to accumulate meaningful history, by grouping decisions hierarchically, useful feedback can accumulate at coarse levels, enabling efficient exploration while retaining fine-grained flexibility.

We implement coarse decisions (p_1 and p_2) using Bayesian Thompson sampling (Russo et al., 2018). Since the magnitude of performance improvement can be noisy, we use only its sign: $r_t = \mathbf{1}[e_t - e_t^{base} > 0]$. For each operation o, we maintain a success probability $\theta_o \sim \mathrm{Beta}(\alpha_o, \beta_o)$, initialized with $\alpha_o = \beta_o = 1$. After observing r_t, the corresponding posterior is updated as

$$\alpha_{o_t} \leftarrow \alpha_{o_t} + r_t, \qquad \beta_{o_t} \leftarrow \beta_{o_t} + (1 - r_t). \tag{5}$$

Thompson sampling draws from each posterior to balance exploration and exploitation. Because the best operation may change as the architecture evolves, we further employ ϵ-greedy Thompson sampling (Do et al., 2024), reserving probability ϵ for uniform exploration. The same procedure is applied to $p_2(c \mid o)$.

Thus, LLMasTool combines algorithmic diversity at coarse levels with LLM flexibility at fine levels, while hierarchical historical feedback makes open-ended exploration progressively more efficient.

4 Experiments

4.1 Experimental Setup

We evaluate LLMasTool under two experimental settings: (1) a NAS-Bench-201 (Dong & Yang, 2020)-style setting, following its training protocol and architectural scale on CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009), and ImageNet16-120 (Chrabaszcz et al., 2017) dataset; and (2) a larger-scale image size setting on ImageNet-100 (Tian et al., 2020).

Table 1: Comparisons with state-of-the-art NAS methods on NAS-Bench-201 Dong & Yang (2020)-style setting. The ♢ indicates methods with parameter sharing, ♡ indicates methods without parameter sharing, and ★ are LLM-based methods. The ± in our results denotes the s.t.d. over three runs.

	Method	Search (archs)	CIFAR-10 validation	CIFAR-10 test	CIFAR-100 validation	CIFAR-100 test	ImageNet16-120 validation	ImageNet16-120 test
	ResNet (our base arch.)	-	90.83	93.97	70.42	70.86	44.53	43.63
	NAS-Bench-201 Optimal	-	91.61	94.37	73.49	73.51	46.77	47.31
hand crafted search space	♢ ENAS	-	37.51±3.19	53.89±0.58	13.37±2.35	13.96±2.33	15.06±1.95	14.84±2.10
	♢ DARTS	-	39.77±0.00	54.30±0.00	38.57±0.00	15.61±0.00	18.87±0.00	16.32±0.00
	♢ SETN	-	84.04±0.28	87.64±0.00	58.86±0.06	59.05±0.24	33.06±0.02	32.52±0.21
	♢ DSNAS	-	89.66±0.29	93.08±0.13	30.87±16.40	31.01±16.38	40.61±0.09	41.07±0.09
	♢ PC-DARTS	-	89.96±0.15	93.41±0.30	67.12±0.39	67.48±0.89	40.83±0.08	41.31±0.22
	♢ SNAS	-	90.10±1.04	92.77±0.83	69.69±2.39	69.34±1.98	42.84±1.79	43.16±2.64
	♢ iDARTS	-	89.86±0.60	93.58±0.32	70.57±0.24	70.83±0.48	40.38±0.59	40.89±0.68
	♢ GDAS	-	89.89±0.08	93.61±0.09	71.34±0.04	70.70±0.30	41.59±1.33	41.71±0.98
	♢ DRNAS	-	91.55±0.00	94.36±0.00	73.49±0.00	73.51±0.00	46.37±0.00	46.34±0.00
	♢ β-DARTS	-	91.55±0.00	94.36±0.00	73.49±0.00	73.51±0.00	46.37±0.00	46.34±0.00
	♢ Λ-DARTS	-	91.55±0.00	94.36±0.00	73.49±0.00	73.51±0.00	46.37±0.00	46.34±0.00
	♡ REA	500	91.19±0.31	93.92±0.30	71.81±1.12	71.84±0.99	45.15±0.89	45.54±1.03
	♡ RS	500	90.93±0.36	93.70±0.36	70.93±1.09	71.04±1.07	44.45±1.10	44.57±1.25
	♡ REINFORCE	500	91.09±0.37	93.85±0.37	71.61±1.12	71.71±1.09	45.05±1.02	45.24±1.18
	♡ BOHB	500	90.82±0.53	93.61±0.52	70.74±1.29	70.85±1.28	44.26±1.36	44.42±1.49
	★ GENIUS	10	91.07±0.20	93.79±0.09	70.96±0.33	70.91±0.72	45.29±0.81	44.96±1.02
	★ CoLLM-NAS	100	91.59±0.04	94.37±0.01	73.44±0.12	73.44±0.15	46.62±0.10	46.79±0.28
open-ended NAS	★ LLMatic	2000	-	94.26±0.13	-	71.62±1.73	-	45.87±0.96
	★ LeMo-NADe(Gemini)	30	82.94	81.76	52.12	52.96	30.34	31.02
	★ LeMo-NADe(GPT4)	30	90.90	89.41	63.38	67.90	27.05	27.70
	★ NADER	500	91.55	94.62	75.72	76.00	50.20	50.52
	★ LLMasTool	100	92.51±0.19	95.08±0.22	76.48±0.35	76.52±0.28	50.40±0.22	50.43±0.55
		500	**93.06**	**95.31**	**77.51**	**77.83**	**52.30**	**53.20**

For NAS-Bench-201-style setting, we follow the protocol of prior work (Yang et al., 2025b). Architectures are constrained to the maximum size of the benchmark search space, with a limit of 1.5M parameters and FLOP budgets of 0.2 GFLOPs for CIFAR-10/CIFAR-100 and 0.05 GFLOPs for ImageNet16-120. Although LLMasTool supports arbitrary initial architectures, we initialize the search from ResNet-32 (He et al., 2016) following the prior work (Yang et al., 2025b). Training follows the original NAS-Bench-201 protocol, including optimizer, epochs, and data splits. We report results after searching 100 and 500 architectures and use three runs when reporting standard errors.

For ImageNet-100, we use a budget of 2M parameters and 0.3 GFLOPs and primarily initialize from MobileNetV3-Small. Results with other initial architectures are provided in Appendix C.5.

For the module database, our algorithm extracts 167, 107, 467, and 681 modules from PyTorch v2.9.0 (PyTorch), MMCV v2.1.0 (OpenMMLab, b), MMPretrain v1.2.0 (OpenMMLab, c), and timm v1.0.19 (HuggingFace), respectively. We use Qwen3-8B (Yang et al., 2025a) as the LLM. Since our tree representation is similar to OpenMMLab configuration files (OpenMMLab, a), we serialize the trees in an OpenMMLab-style format when prompting the LLM. Further details are provided in Appendix B.

4.2 BENCHMARKING

Table 1 compares LLMasTool with representative NAS methods under the NAS-Bench-201-style setting. There are two categories of NAS methods: NAS methods operating on hand-crafted search spaces, including ENAS (Pham et al., 2018), DARTS (Liu et al., 2019b), SETN (Dong & Yang, 2019a), PC-DARTS (Xu et al., 2020), GDAS (Dong & Yang, 2019b), DRNAS (Chen et al., 2021c), REA (Real et al., 2019), random search (RS) (Bergstra & Bengio, 2012), REINFORCE (Williams, 1992), BOHB (Falkner et al., 2018), GENIUS (Zheng et al., 2023), and CoLLM-NAS (Li et al., 2025); and open-ended NAS methods, including LLMatic (Nasir et al., 2024), LeMo-NADe (Rahman & Chakraborty, 2024), and NADER (Yang et al., 2025b).

Hand-crafted search spaces restrict exploration to a narrow set of manually designed architectures. Although recent methods such as Λ-DARTS (Movahedi et al., 2022) can approach the benchmark optimum efficiently, their discoveries remain bounded by the predefined topology.

Open-ended search removes this restriction but substantially increases the search difficulty. Code-generation methods such as LLMatic (Nasir et al., 2024) and LeMo-NADe (Rahman & Chakraborty,

Table 2: Ablation Studies of the proposed components in the architecture evolution. We evolved from MobileNetV3-small0.5 to achieve better accuracy within one epoch training on ImageNet-100. The *paper insp.* denotes the setting where 10 modification ideas are randomly selected and utilized from the 5,411 ideas that NADER Yang et al. (2025b) gathered from papers as inspiration. The grayed components are our proposal.

evo. step	historical memory	planning	template category	operation	sampling	Acc.
MobileNetV3		-	-	-	-	24.21
code-gen		LLM's idea	-	-	uniform	28.62
config-gen		LLM's idea	-	-	uniform	28.88
tree-trans		LLM's idea	-	basic	uniform	33.44
tree-trans	✓	LLM's idea	-	basic	uniform	33.45
tree-trans	✓	paper inspi.	-	basic	uniform	34.98
tree-trans	✓	LLM's idea	rely LLM	basic	uniform	33.26
tree-trans	✓	c-to-f	(rely+inv.) LLM	basic	uniform	34.00
tree-trans	✓	c-to-f	(rely+inv.+w/o) LLM	basic	uniform	36.18
tree-trans	✓	c-to-f	(rely+inv.+w/o) LLM	basic+repeat	uniform	38.86
tree-trans	✓	c-to-f	(rely+inv.+w/o) LLM	basic+repeat	Thompson	33.22
tree-trans	✓	c-to-f	(rely+inv.+w/o) LLM	basic+repeat	H-Thompson	**40.78**

2024) can generate diverse architectures but may suffer from instability and executability issues. NADER (Yang et al., 2025b) improves the reliability of LLM-based search by generating data-flow graphs and incorporating ideas from recent papers. However, its search remains dependent on the knowledge and representations available to the LLM.

LLMasTool substantially outperforms previous methods in both categories. From the hand-crafted setting, it improves test accuracy by 4.39 and 6.41 points on CIFAR-100 and ImageNet16-120, respectively. In open-ended NAS, it surpasses NADER by 1.83 and 2.68 points on the same datasets. These results demonstrate the effectiveness of the proposed formulation.

The improvement comes from how we formulate architecture evolution. Unlike methods that generate source code (Nasir et al., 2024; Rahman & Chakraborty, 2024) or data-flow graphs (Yang et al., 2025b), LLMasTool generates neither the complete architecture nor an unrestricted graph. Instead, it performs explicit tree transformations. This allows coarse decisions to be diversified algorithmically, enables direct reuse of existing modules, and provides a structured representation for constructing complex architectures while maintaining executability.

4.3 Ablation Studies

We perform ablation studies on ImageNet-100 to analyze the contribution of each component. Unless otherwise specified, we generate 100 architectures and train each for one epoch. The experimental setup and the longer-training experiments are provided in Appendices B.3 and C.7.

Effectiveness of Module Mining. We first compare our mined module database $\mathcal{D}_M$ with a primitive module database. Specifically, we compare (1) transformations using only MobileNetV3 modules and primitive modules implemented in the PyTorch library, and (2) transformations using the diverse modules automatically mined from existing source code.

Table 3: Effectiveness of module mining

modules	Acc.
primitive	33.21
mined (ours)	**40.78**

As shown in Table 3, module mining substantially improves performance by enabling direct reuse of components developed in previous research. Since the procedure operates on arbitrary PyTorch source code, new implementations can also be incorporated without manual intervention.

Effect of Tree Representation. Next, we analyze the impact of representing architectures as a deploy-friendly hierarchical tree by comparing three architecture representations: **code-gen**, which directly generates or modifies source code; **config-gen**, which directly generates or modifies OpenMMLab-style configuration text; and **tree-trans (ours)**, which performs explicit transformations on the hierarchical tree. To isolate the effect of representation, all variants use the same instruction, "please improve architecture", together with the corresponding output format.

As can be seen by comparing the second to fourth rows of Table 2, code generation frequently produces familiar architectural patterns or introduces undesirable structural behavior, resulting in limited gains. Configuration generation still requires the LLM to determine the complete architecture, resulting in similar gains. In contrast, tree-trans explicitly specifies the transformation space and only asks the LLM to resolve the required local decisions, improving performance by 4.82 points over code-gen. More importantly, the tree representation enables systematic control over modification locations and transformation types, which is essential for the coarse-to-fine planning proposal evaluated in the next section.

Effect of Coarse-to-Fine Planning. To evaluate the role of our coarse-to-fine decision making, we compare several planning strategies. As shown in Table 2, *planning with LLM's idea*, where the LLM directly determines the entire transformation without hierarchical decomposition, tends to plateau beyond a certain performance level (The evolutionary process is visualized in Fig. 4). While methods such as NADER (Yang et al., 2025b) improve exploration diversity by extracting inspiration from multiple papers, such ideas remain bounded by the LLM's prior knowledge, limiting truly diverse exploration.

In contrast, our coarse-to-fine framework algorithmically enumerates all modifiable nodes, modification types, and available modules, then determines a general direction of change before using the LLM to resolve the remaining degrees of freedom. This enables diverse directional exploration while leveraging the LLM's prior knowledge to make effective and executable decisions within each direction.

For the template category, mini. LLM, which assigns minimal decision-making to the LLM, achieves the largest improvement. The result highlights the importance of our coarse-to-fine framework, which uses the LLM as a tool rather than an agentic planner.

Effect of Meta-Level Operation. We evaluate the **Repeat Previous** meta-operation. As shown in Table 2, adding this operation drastically improves the performance (The evolutionary lineage in Fig. 5 shows that this operation is effectively used to generate the best architecture.). Coarse-to-fine exploration guarantees diversity in the search process, but at the cost of substantially expanding the search space. Once a promising direction for improvement is identified, repeatedly exploring similar directions enhances the efficiency of the search.

Effect of Hierarchical Thompson Sampling. In Table 2, we analyze the importance of hierarchical Thompson (H-Thompson) sampling, in which Thompson sampling is applied to both p_1 and p_2 under a hierarchical categorization scheme. We replace it with (1) uniform sampling, or (2) normal Thompson sampling. Uniform sampling lacks efficiency under the vast search space, while normal Thompson sampling fails due to too sparse historical data. Our H-Thompson sampling provides a balance between exploration and exploitation, yielding both faster improvement (See also Fig. 4) and better final architectures.

Summary of Findings. Overall, the ablation studies demonstrate that the performance gain of LLMasTool does not arise from the LLM alone, but from the synergy between (1) rule-based module mining, (2) deploy-friendly tree representation, and (3) diversity-guided coarse-to-fine planning with Bayesian sampling. These results validate the necessity of each design choice in enabling high-performance open-ended exploration.

5 Conclusion

We introduced LLMasTool, a NAS framework for open-ended architecture search. LLMasTool combines three key ideas: algorithmic module mining from existing source code, a deploy-friendly hierarchical tree representation, and coarse-to-fine evolution in which an LLM serves as a tool for fine-grained decisions rather than an autonomous architecture generator. The hierarchical decision process further enables Bayesian Thompson sampling to accumulate feedback at meaningful levels of abstraction. This combination balances the rule-based algorithm and LLM flexibility, achieving state-of-the-art performance among the compared open-ended NAS methods.

AI USE STATEMENT

In this work, we used generative AI tools for translation, grammar correction, and language polishing during manuscript preparation. We have not used generative AI tools to generate research ideas, develop the methodology, design experiments, analyze results, or draw scientific conclusions, and the remaining required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to help implement a small portion of the source code. The majority of the source code was written by the authors, and all AI-assisted code was reviewed and understood by the authors, and the correctness was verified as part of the development and testing process. We reviewed all AI-assisted work. The authors assume full responsibility for the final content of this work, including all text, claims, methodology, experimental results, and software artifacts produced with the aid of generative AI.

REFERENCES

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for All: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.

Angelica Chen, David M Dohan, and David R So. EvoPrompting: language models for code-level neural architecture search. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pp. 7787–7817, 2023.

Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 12270–12280, 2021a.

Minghao Chen, Kan Wu, Bolin Ni, Houwen Peng, Bei Liu, Jianlong Fu, Hongyang Chao, and Haibin Ling. Searching the search space of vision transformer. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, pp. 8714–8726, 2021b.

Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Dr{nas}: Dirichlet neural architecture search. In *International Conference on Learning Representations*, 2021c.

Junyan Cheng, Peter Clark, and Kyle Richardson. Language modeling by language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pp. 702–703, 2020.

Bach Do, Taiwo Adebiyi, and Ruda Zhang. Epsilon-greedy thompson sampling to bayesian optimization. *Journal of Computing and Information Science in Engineering*, 24(12):121006, 2024.

Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 3681–3690, 2019a.

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1761–1770, 2019b.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International conference on machine learning*, pp. 1437–1446. PMLR, 2018.

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*, pp. 770–778, 2016.

HuggingFace. GitHub - huggingface/pytorch-image-models — github.com. `https://github.com/huggingface/pytorch-image-models`. [Accessed 24-09-2026].

Jeimin Jeon, Youngmin Oh, Junghyup Lee, Donghyeon Baek, Dohyung Kim, Chanho Eom, and Bumsub Ham. Subnet-aware dynamic supernet training for neural architecture search. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 30137–30146, 2025.

Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 1946–1956. ACM, 2019.

Haidong Kang, Lihong Lin, and Hanling Wang. Revolutionizing training-free NAS: Towards efficient automatic proxy discovery via large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Brett Koonce. Mobilenetv3. In *Convolutional neural networks with swift for tensorflow: image recognition and dataset categorization*, pp. 125–144. Springer, 2021.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Master's thesis, University of Toronto*, 2009.

Zhe Li, Zhiwei Lin, and Yongtao Wang. CoLLM-NAS: Collaborative large language models for efficient knowledge-guided neural architecture search. *arXiv preprint arXiv:2509.26037*, 2025.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 19–34, 2018.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L. Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019a.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *7th International Conference on Learning Representations (ICLR 2019)*. OpenReview.net, 2019b.

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 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021.

Shun Lu, Yu Hu, Longxing Yang, Zihao Sun, Jilin Mei, Jianchao Tan, and Chengru Song. PA&DA: Jointly sampling path and data for consistent NAS. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11940–11949, 2023.

Sachin Mehta and Mohammad Rastegari. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*, 2021.

Keith G Mills, Di Niu, Mohammad Salameh, Weichen Qiu, Fred X Han, Puyuan Liu, Jialin Zhang, Wei Lu, and Shangling Jui. AIO-P: Expanding neural performance predictors beyond image classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 9180–9189, 2023.

Keith G Mills, Fred X Han, Mohammad Salameh, Shengyao Lu, Chunhua Zhou, Jiao He, Fengyu Sun, and Di Niu. Building optimal neural architectures using interpretable knowledge. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5726–5735, 2024.

Sajad Movahedi, Melika Adabinejad, Ayyoob Imani, Arezou Keshavarz, Mostafa Dehghani, Azadeh Shakery, and Babak N Araabi. Λ-DARTS: Mitigating performance collapse by harmonizing operation selection among cells. *arXiv preprint arXiv:2210.07998*, 2022.

Muhammad Umair Nasir, Sam Earle, Julian Togelius, Steven James, and Christopher Cleghorn. LLMatic: Neural architecture search via large language models and quality diversity optimization. In *proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1110–1118, 2024.

Youngmin Oh, Hyunju Lee, and Bumsub Ham. Efficient few-shot neural architecture search by counting the number of nonlinear functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 19740–19748, 2025.

OpenMMLab. GitHub - openmmlab. `https://github.com/open-mmlab`, a. [Accessed 24-09-2026].

OpenMMLab. GitHub - open-mmlab/mmcv: OpenMMLab Computer Vision Foundation — github.com. `https://github.com/open-mmlab/mmcv`, b. [Accessed 24-09-2026].

OpenMMLab. GitHub - open-mmlab/mmpretrain: OpenMMLab Pre-training Toolbox and Benchmark — github.com. `https://github.com/open-mmlab/mmpretrain`, c. [Accessed 24-09-2026].

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4095–4104. PMLR, 2018.

PyTorch. PyTorch — pytorch.org. `https://pytorch.org/`. [Accessed 24-09-2026].

Jincheng Qian, Shuxin Zheng, Yang Li, Jianguo Zhang, Zewei Lin, Jiashen Liu, Bin Wang, and Yue Gao. When NAS meets trees: An efficient algorithm for neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2022.

Md Hafizur Rahman and Prabuddha Chakraborty. Lemo-nade: Multi-parameter neural architecture discovery with llms. *arXiv preprint arXiv:2402.18443*, 2024.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2902–2911. PMLR, 2017.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Daniel J. Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on thompson sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96, 2018.

Yuiko Sakuma, Masakazu Yoshimura, Junji Otsuka, Atsushi Irie, and Takeshi Ohashi. Mixed-precision supernet training from vision foundation models using low rank adapter. *arXiv preprint arXiv:2403.20080*, 2024.

Mohammad Salameh, Keith Mills, Negar Hassanpour, Fred Han, Shuting Zhang, Wei Lu, Shangling Jui, Chunhua Zhou, Fengyu Sun, and Di Niu. AutoGO: Automated computation graph optimization for neural network evolution. *Advances in Neural Information Processing Systems*, 36: 74455–74477, 2023.

Yuheng Shi, Minjing Dong, and Chang Xu. Multi-scale vmamba: Hierarchy in hierarchy visual state space model. *Advances in Neural Information Processing Systems*, 37:25687–25708, 2024.

David So, Quoc Le, and Chen Liang. The evolved transformer. In *International conference on machine learning*, pp. 5877–5886, 2019.

Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path NAS: Device-aware efficient convnet design. In *Joint Workshop on On-Device Machine Learning & Compact Deep Neural Network Representations with Industrial Applications (ODML-CDNNRIA) in Conjunction with International Conference on Machine Learning*, 2019.

Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the genetic and evolutionary computation conference*, pp. 497–504, 2017.

Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *European conference on computer vision*, pp. 776–794. Springer, 2020.

Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, volume 48 of *Proceedings of Machine Learning Research*, pp. 564–572. PMLR, 2016.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10734–10742, 2019.

Chaodong Xiao, Minghan Li, Zhengqiang Zhang, Deyu Meng, and Lei Zhang. Spatial-mamba: Effective visual state space models via structure-aware state fusion. In *International Conference on Learning Representations (ICLR)*, 2025.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Zekang Yang, Jingzhe Shi, Haokui Zhang, Xinghao Chen, Yifei Zhang, Yu Zhang, Jiahui Huang, Shuai Yang, Dapeng Liu, Lei Chen, and Wei Chen. NADER: Neural architecture design via multi-agent collaboration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2025b.

Masakazu Yoshimura, Teruaki Hayashi, and Yota Maeda. Mambapeft: Exploring parameter-efficient fine-tuning for mamba. In *The Thirteenth International Conference on Learning Representations*, 2025.

Masakazu Yoshimura, Teruaki Hayashi, Yuki Hoshino, Wei-Yao Wang, and Takeshi Ohashi. Sf-mamba: Rethinking state space model for vision. *arXiv preprint arXiv:2603.16423*, 2026.

Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. BigNAS: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision*, pp. 702–717, 2020.

Buang Zhang, Xinle Wu, Hao Miao, Chenjuan Guo, and Bin Yang. Dependency-aware differentiable neural architecture search. In *European Conference on Computer Vision*, pp. 219–236, 2024.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856, 2018.

Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. Few-shot neural architecture search. In *International Conference on Machine Learning*, pp. 12707–12718, 2021.

Mingkai Zheng, Xiu Su, Shan You, Fei Wang, Chen Qian, Chang Xu, and Samuel Albanie. Can gpt-4 perform neural architecture search? *arXiv preprint arXiv:2304.10970*, 2023.

Xun Zhou, Liang Feng, Xingyu Wu, Zhichao Lu, and Kay Chen Tan. Design principle transfer in neural architecture search via large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 2025a.

Xun Zhou, Xingyu Wu, Liang Feng, Zhichao Lu, and Kay Chen Tan. Design principle transfer in neural architecture search via large language models. In *39th Annual AAAI Conference on Artificial Intelligence (AAAI 2025)*, pp. 23000–23008, 2025b.

Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. Vision mamba: Efficient visual representation learning with bidirectional state space model. In *ICML*. OpenReview.net, 2024.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations (ICLR 2017), Conference Track Proceedings*. OpenReview.net, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8697–8710, June 2018.

A Limitation and Future Works

Various evaluations demonstrated that LLMasTool achieves significant performance improvements over previous work by enabling open-ended exploration without being constrained by handcrafted search spaces or the prior knowledge embedded in LLMs. However, several limitations remain.

In this work, we conducted proof-of-concept experiments using packages that include many implementations, such as PyTorch (PyTorch), MMPretrain (OpenMMLab, c), MMCV (OpenMMLab, b), and timm (HuggingFace). However, in recent years, other architectures, such as Mamba-based vision models (Zhu et al., 2024; Shi et al., 2024; Xiao et al., 2025; Yoshimura et al., 2026; 2025), have been proposed that are not included in these libraries. These models often achieve improvements by implementing custom CUDA kernels, and seamlessly integrating them into our framework would require a more sophisticated implementation that builds and manages multiple such CUDA kernels simultaneously.

Another limitation is the time required for the search process. The time required for LLM generation itself is about one minute, while training takes approximately 60 to 150 minutes for 200-epoch training on CIFAR-100 or ImageNet16-120. Therefore, compared to conventional NAS approaches based on evolutionary algorithms (Suganuma et al., 2017; Liu et al., 2018; So et al., 2019) or reinforcement learning (Zoph & Le, 2017; Zoph et al., 2018; Pham et al., 2018), the overall time overhead is minimal. However, because the training cost for each candidate architecture is still high, searching over larger-scale model structures would require substantially more time. To discover large-scale architectures, future work would benefit from zero-shot NAS methods that can reliably evaluate the relative quality of architectures without full training. The challenge of zero-shot NAS for open-ended exploration should be to accurately evaluate fundamentally different structures.

B EXPERIMENTAL SETUP DETAILS

B.1 COMMON SETUP

Here, we describe the detailed experimental settings. In our experiments, we conducted the architecture search with a budget of either 100 or 500 architecture trials. To accelerate the experiments, the evolution process is not executed sequentially, but we run four evolution processes in parallel.

To balance the trade-off between exploration and exploitation, a promising base architecture is sampled from the architecture database $\mathcal{D}_A$ as $A_t^{base} \sim \mathcal{D}_A$. Specifically, in the 100-trial setting, the base architecture is sampled from the top-5 architectures ranked by accuracy. In the 500-trial setting, it is sampled from the top-25 architectures. In terms of the coarse-to-fine decision making process, the ϵ for the ϵ-greedy Thompson sampling (Do et al., 2024) is set to 0.5.

B.2 NAS-BENCH-201

Table 4: The training hyperparemeter settings for CIFAR-10 and CIFAR-100

optimizer	SGD	initial LR	0.1
Nesterov	✓	ending LR	0
momentum	0.9	LR schedule	cosine
weight decay	0.0005	epoch	200
batch size	256	normalization	✓
random flip	p=0.5	random crop	size=32, padding=4

Following the experimental setup of NAS-Bench-201, we use the same partitions for the training, validation, and test datasets, as well as the same training hyperparameters. The hyperparameters for CIFAR-10 and CIFAR-100 are summarized in Table 4. For ImageNet16-120, we apply the same settings, except for a random crop size of 16 and a padding of 2 following the previous convention (Yang et al., 2025b; Liu et al., 2019b).

B.3 IMAGENET-100

Table 5: The training hyperparemeter settings for ImageNet-100

	1 epoch setup	100 epoch setup
batch size (total)	64	128
optimizer	AdamW	AdamW
optimizer momentum (β_1, β_2)	(0.9, 0.99)	(0.9, 0.99)
learning rate	1e-3	5e-3
minimal learning rate	1e-5	5e-5
learning rate schedule	cosine	cosine
weight decay	0.05	0.05
Warmup epochs	0	5
random crop size	224	224
RandAug Cubuk et al. (2020)	-	rand-m9-mstd0.5
random erasing prob.	-	0.25
mixup alpha	-	0.8
cutmix alpha	-	1.0
label smoothing	-	0.1
mixed precision (AMP)	-	✓

For ImageNet-100, we use the predefined class subset and the train/validation split in (Tian et al., 2020). We mainly follow the common Swin Transformer's hyperparameter settings for ImageNet-1K (Liu et al., 2021) but slightly adapted to the dataset size and the number of training epochs. In detail, we evaluated with one and 100 training epochs. For one epoch training setup, we use batch size of 64 and learning rate of 1e-3 without image augmentation. For 100 epoch training setup, we use batch size of 128, learning rate of 5e-4. More details can be found in Table 5.

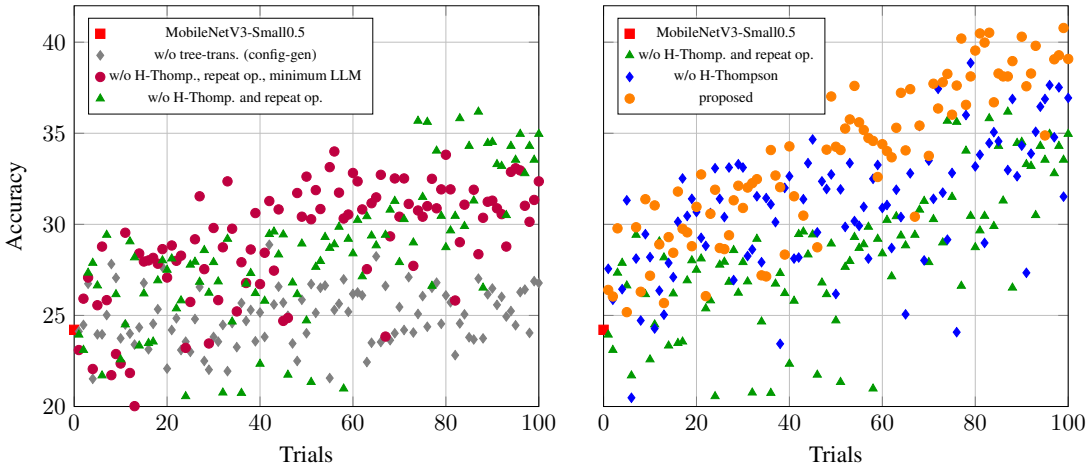

Figure 4: Evolutionary process on ImageNet-100 dataset with one epoch training.

C Additional Experiments

C.1 Analysis of the Evolutionary Process

Fig. 4 shows the evolutionary process of the ablation study on ImageNet-100. First, when we simply let the LLM improve the OpenMMLab configuration file—which serves as the input and output format for our LLM—through a configuration generation approach (config-gen), the performance improvement quickly stagnates. In contrast, by adopting tree transformation as an evolutionary process while still allowing the LLM to perform substantial planning, the performance improves significantly.

Furthermore, by introducing prompt categories that minimize the use of the LLM in the planning stage, the initial speed of performance improvement becomes slower; however, the search is no longer constrained by the prior bias of the LLM, resulting in better performance gains in the later stages. In addition, introducing the repeat operation, which performs a similar transformation again when a performance improvement is observed, substantially accelerates the speed of performance improvement.

Finally, by employing hierarchical Thompson sampling, performance improvements accelerate in the later stages once a meaningful history has been accumulated. Although the process is highly stochastic and noisy, these results demonstrate that each proposed component contributes as intended.

Fig. 5 shows the evolutionary lineage of 500 architectures explored on CIFAR-100 dataset. The thick lines and large circles indicate the path that leads to the best-performing architecture. The lineage shows that performance is improved through a combination of multiple operations. It also reveals that, once a promising direction of evolution is identified, the *Repeat Previous* operation is applied multiple times to repeatedly perform transformations in the same direction, further improving performance.

C.2 Relying LLM or Minimum LLM Use

We investigate how much we should rely on LLMs in *swap module* operation. To swap a module, it is necessary to determine where to replace, which module to use, and what hyperparameters to use. Since hyperparameters can only be determined by the LLM, we conduct an investigation to decide whether the other two decisions—where to replace and which module to use—should be handled by random sampling or by the LLM's decision.

Interestingly, Table 6 shows that performance improves more when random sampling is used instead of relying on prior knowledge of LLMs. Of course, in practical settings where the swap module operation is not executed 100 times as in this experiment, the number of times the swap

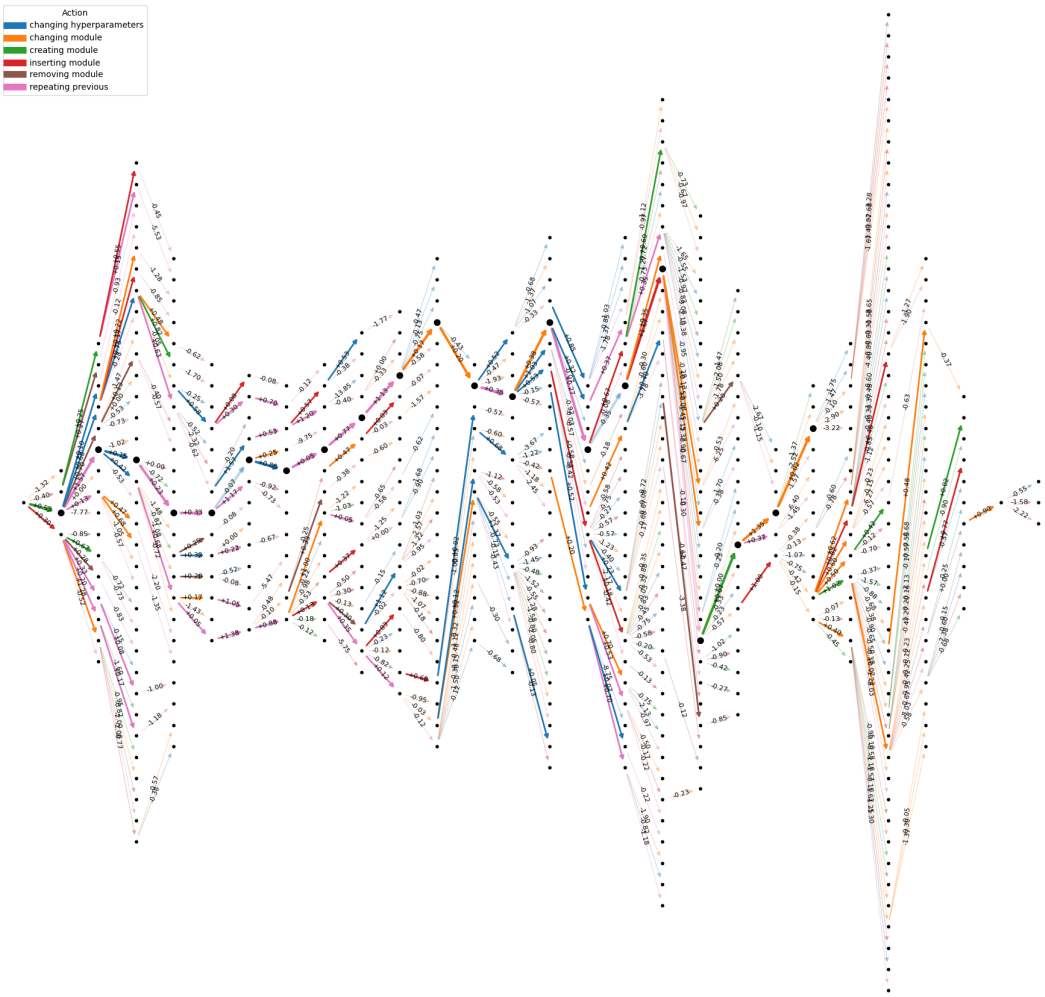

Figure 5: The evolutionary lineage of 500 architectures explored on CIFAR-100 dataset. The thick lines and large circles indicate the path that leads to the best-performing architecture. The numbers indicate the performance improvement in percentage points. The thick and thin lines indicate transformations that improved and degraded performance, respectively.

Table 6: We investigate how much should we rely LLMs in *swap module* operation. We conduct an investigation whether the two decisions—where to replace and which module to use—should be handled by random sampling or by the LLM's decision. In this experiment, we only use the *swap module* operation in 100 architecture evolution.

where to swap	which module to use	Acc.
LLM	sample 8 $\rightarrow$ LLM	29.98
sample 5 $\rightarrow$ LLM	sample 8 $\rightarrow$ LLM	30.14
random sample	sample 8 $\rightarrow$ LLM	34.34
random sample	random sample	**35.58**

module operation is used becomes smaller, which makes it more necessary to leverage prior knowledge to improve efficiency. However, if NAS relies too heavily on prior knowledge of LLMs , the improvement tends to stagnate during the search process.

Furthermore, the results in Table 9 indicate that this trend remains unchanged even when a larger LLM is used.

C.3 Detailed Ablation Studies about Hierarchical Thompson Sampling

Table 7: Ablation studies of the exploration-exploitation trade-off in our hierarchical Thompson sampling.

ϵ-greedy Thompson	uniform	hierarchical	ϵ	Acc.
	p_1,p_2, p_3, p_4	-	-	38.86
p_1	p_2, p_3, p_4	-	0.5	40.21
p_1,p_2,p_3	p_4	w/o	0.5	36.21
p_1,p_2,p_3	p_4	w/	0.5	36.38
p_1,p_2	p_3, p_4	w/o	0.5	36.88
p_1,p_2	p_3, p_4	w/	0.0	35.10
p_1,p_2	p_3, p_4	w/	0.2	34.72
p_1,p_2	p_3, p_4	w/	0.5	**40.78**
p_1,p_2	p_3, p_4	w/	0.8	39.97
p_1,p_2	p_3, p_4	w/	1.0	38.86

In this section, we conduct a detailed ablation study of **hierarchical Thompson sampling**, which combines Bayesian sampling with coarse-to-fine hierarchical decision making. In the proposed method, ϵ-greedy Thompson sampling (Do et al., 2024) is applied hierarchically to p_1 and p_2 in Equation (4) defined in the main text (the decision of the operation and the prompt category). We compare this with several alternatives: applying ϵ-greedy Thompson sampling up to p_3, removing the hierarchical structure, and varying the value of ϵ, which controls the probability of using uniform sampling instead of Thompson sampling.

As shown in Table 7, when the decision process is not hierarchical, the meaningful history cannot be accumulated within 100 architecture trials, resulting in a degraded performance. Even with hierarchical decision making, applying Bayesian exploitation up to p_3 makes the history too sparse because the number of effective trials is only around 60–70 after excluding the *repeat previous* operation. Therefore, a meaningful history can only be obtained when the number of combinations is limited to approximately operation (5) $\times$ prompt category (3) ≈ 15.

We also find that $\epsilon \approx 0.5$ yields the best performance. Because the optimal strategy changes as the architecture evolves during the search, exploration should be emphasized more than exploitation through pure Thompson sampling.

C.4 Evaluation of Each Operation

Table 8: The performance of each operation on ImageNet-100 one epoch setting.

	Acc.
MobileNet-V3	24.21
Change hyperparameter	31.48
Swap module	32.94
Insert module and Delete module	31.86
Create module	31.04
All basic operations	34.82
+ Repeat previous	40.78

The performance of each operation is reported in Table 8. In this experiment, we evaluate each operation independently by conducting searches that use only a single type of operation. As shown in the results, each operation can improve performance. However, relying on a single operation is insufficient; combining all operations to explore diverse directions leads to substantially better performance.

Furthermore, because the search is open-ended, the *repeat previous operation* significantly accelerates the exploration process. Once a direction that improves performance is discovered, repeating the same directional changes allows the search to efficiently exploit that improvement trajectory.

Table 9: Analysis of the impact of LLM performance differences. We report the results of searching 100 architectures on ImageNet-100 with one epoch training.

	Qwen3-4B	Qwen3-8B	Qwen3-32B
LLMasTool w/o minimum LLM category	35.52	36.75	34.52
LLMasTool	**41.26**	40.78	39.97

C.5 Sensitivity to the Performance of LLM

Table 9 presents the results obtained using LLMs of different sizes. As can be seen, the proposed method achieves strong performance regardless of the capability of the LLM. This is likely because the evolution step is defined as a tree transformation, which is easier than code generation, allowing the method to function effectively even when the LLM is not highly capable.

It may be specific to these three models, but in our experiments the smaller LLMs show slightly better performance. Since our experiments consistently improve by prioritizing diversity, one possible explanation is that stronger LLMs memorize the information in the training data more faithfully, resulting in a stronger prior bias that may hinder performance in this setting.

Furthermore, in the coarse-to-fine decision making process, prompt categories that minimally rely on LLMs are effective even when using high-performance LLMs. This suggests that, regardless of the LLM's capability, the prior bias is largely determined by the data on which the LLM was trained. Therefore, to achieve performance beyond the existing models, it is important to conduct a search process that does not rely solely on the LLM but instead uses diversifying algorithmic exploration to avoid being constrained by the LLM's prior bias.

C.6 Sensitivity to the Base Architectures

Table 10: Investigation of the robustness of the proposed method with respect to the base architecture. Architectures are searched on ImageNet-100 to maximize performance after 1 epoch of training, under the constraints of 300M FLOPs and fewer than 2M parameters. 100 architectures are searched.

Base arch.	Params [M]	MFLOPs	Acc.
ShuffleNet Zhang et al. (2018)	1.25	158	23.86
+ LLMasTool	1.52	186	**39.22**
MobileNetV3-Small0.5 Koonce (2021)	0.97	57	24.21
+ LLMasTool	0.95	202	**40.78**
MobileViT-XXS Mehta & Rastegari (2021)	0.98	254	22.00
+ LLMasTool	0.90	239	**31.17**

We investigate the robustness of the proposed method with respect to the base models used as the initial data for the architecture database $\mathcal{D}_A$. In addition to MobileNetV3-Small0.5, we consider three base models of comparable scale: ShuffleNet (Zhang et al., 2018), an earlier lightweight architecture, and MobileViT-XXS (Mehta & Rastegari, 2021), which incorporates attention mechanisms. We search for 100 architectures on ImageNet-100 (Tian et al., 2020), aiming to maximize performance after one epoch of training under the constraints of 300M FLOPs and fewer than 2M parameters.

As shown in Table 10, the performance can be substantially improved from all base models. However, when the search is limited to only 100 explored architectures, the results are still affected by the choice of the base model, which can lead to lower performance. In particular, when training for only one epoch, the MobileViT-based models perform poorly because attention modules tend to converge slowly. Nevertheless, in the best discovered architecture, all attention modules are successfully removed. This indicates that the method is not constrained by the prior assumption that attention mechanisms are inherently more advanced than CNNs, and instead can evolve architectures that are better suited to the target task setting.

C.7 IMAGENET-100 WITH LONGER EPOCHS

Table 11: Ablation Studies of the proposed components in the architecture evolution. This time, we maximize the accuracy with 100 epoch training on ImageNet-100. We report results after generating 100 architectures. The grayed components are our proposal.

evo. step	planning	sampling	Acc.
MobileNetV3	-	-	73.00
code-gen	LLM's idea	uniform	74.96
tree-trans	c-to-f	uniform	81.00
tree-trans	c-to-f	H-Thompson	**81.72**

In the ablation study in the main paper, we searched for models that achieve high accuracy after only one epoch of training. Demonstrating large performance improvements under this setting suggests that LLMasTool is effective even in practical scenarios where training must be performed with limited data, as well as in settings that deviate from commonly used training protocols.

On the other hand, it is also important to examine whether, under more conventional training settings that align well with the information contained in LLM training data, it is still preferable to use the LLM as a tool through coarse-to-fine decision making, rather than delegating all decisions to the LLM in an agentic manner. To investigate this, Table 11 also presents an ablation study where 100 architectures are searched on ImageNet-100 with 100 training epochs. Detailed experimental settings are described in Sec. B.3.

As shown in the table, coarse-to-fine decision making remains important even under standard training settings, indicating that exploration that is not constrained by the prior biases of the LLM is crucial. Furthermore, hierarchical Thompson sampling is also shown to be effective.

D IMPLEMENTATION DETAILS

D.1 OVERALL EVOLUTION FLOW

An overview of the evolution process described in the main text is shown in Algorithm 1 . The general framework of the evolution process follows a standard evolutionary procedure.

Algorithm 1 presents the proposed coarse-to-fine decision-making algorithm. Because the architecture is represented in a tree structure that is easy for the algorithm to manipulate, the algorithm can explicitly determine the direction of architectural modifications. We introduce a gradation in how much responsibility is assigned to the LLM. Some prompt categories allow the LLM to decide many of the decisions, while others restrict the LLM to making only minimal decisions. The details are described in the next section.

D.2 TREE TRANSFORMATION OPERATIONS

The proposed tree transformation T is realized through a set of tools that operate on the architecture tree. These tools consist of (i) algorithm-based tools that deterministically analyze and manipulate the tree structure, and (ii) LLM-based tools that enable transformations beyond the capability of predefined algorithms.

When processed by algorithmic tools, an architecture tree A is represented as a nested Python dictionary object that may contain list structures. Formally, we denote this representation as

$$A \in \mathcal{T}_{\text{dict}}, \tag{6}$$

where $\mathcal{T}_{\text{dict}}$ denotes the space of nested dictionary-based tree structures. In contrast, when interacting with the LLM, the architecture is converted into a textual configuration format similar to the OpenMMLab configuration file, denoted as

$$A_{\text{text}} \in \mathcal{T}_{\text{text}}. \tag{7}$$

We first introduce the algorithm-based tools used to analyze and manipulate the tree structure.

Algorithm 1 Architecture Evolution

Require: Module database $\mathcal{D}_M$, initial architecture database $\mathcal{D}_A$
1: Initialize history database $\mathcal{D}_H \leftarrow \emptyset$
2: **while** search budget not exhausted **do**
3: Sample base architecture

$$A_t^{base} \sim \mathcal{D}_A$$

4: Sample tree transformation using Algorithm 2

$$T_t \sim p_{cf}\Big(T \mid A_t^{base}, \mathcal{D}_A, \mathcal{D}_M, \mathcal{D}_H, \mathcal{LLM}\Big)$$

5: Apply transformation

$$A_t \leftarrow T_t(A_t^{base})$$

6: Check feasibility (executability, size constraint, intended transformation)

$$\epsilon_t = \mathcal{E}(A_t^{base}, A_t, T_t)$$

7: **if** $\epsilon_t = 1$ **then**
8: Instantiate A_t and train/evaluate to obtain a metric e_t
9: Update architecture database

$$\mathcal{D}_A \leftarrow \mathcal{D}_A \cup \{(A_t, e_t)\}$$

10: Update history database

$$\mathcal{D}_H \leftarrow \mathcal{D}_H \cup \{(T_t, A_t^{base}, e_t - e_t^{base})\}$$

11: **end if**
12: **end while**

Algorithm 2 Coarse-to-Fine Decision making p_{cf}

Require: Base architecture A_t^{base}, module database $\mathcal{D}_M$, history database $\mathcal{D}_H$
1: Sample operation $o \sim p_1(o)$ via ϵ-greedy Thompson sampling using $\mathcal{D}_H$
2: Sample prompt category conditioned on operation $c \sim p_2(c \mid o)$ via ϵ-greedy Thompson sampling using $\mathcal{D}_H$
3: Sample prompt template $\tau \sim p_3(\tau \mid c, o)$ uniformly
4: Retrieve candidate nodes/modules from A_t^{base} and $\mathcal{D}_M$
5: Sample placeholder assignments $\phi \sim p_4(\phi \mid c, o, \tau, A_t^{base}, \mathcal{D}_M)$
6: Construct prompt using (o, c, τ, ϕ) and history $\mathcal{D}_H$
7: Use LLM to generate concrete tree transformation
8: $T \sim p_5(T \mid c, o, \tau, \phi, A_t^{base}, \mathcal{D}_M, \mathcal{D}_H)$
9: **return** T

- **Module Attribute Extraction** A function that retrieves all module attributes and their corresponding module names appearing in the tree structure:

$$\mathrm{Attr}(A) = \{(a_1, m_1), (a_2, m_2), \ldots, (a_K, m_K)\}, \tag{8}$$

where a_k denotes the hierarchical attribute of the module (e.g., *model.backbone.module1*) and m_k denotes the corresponding module name.

- **Sequential Module Attribute Extraction** A function that extracts module attributes located inside the list structures of the tree:

$$\mathrm{Attr}_{\mathrm{list}}(A) \subseteq \mathrm{Attr}(A). \tag{9}$$

In most architectures, list structures correspond to sequential module compositions.

- **Sub-tree Retrieval** A function that retrieves the sub-tree (module) located at a specified attribute a from the architecture A:

$$S = \mathrm{GetSubTree}(A, a), \tag{10}$$

where a denotes the hierarchical attribute of a module in A, and S denotes the corresponding sub-tree.

- **Sub-tree Replacement** A function that replaces a sub-tree (module) located at a specified attribute a with a new sub-tree S:

$$A' = \text{Replace}(A, a, S). \tag{11}$$

- **Sub-tree Deletion from List Structure** A function that removes the sub-tree (module) corresponding to a specified attribute a within a list structure:

$$A' = \text{Delete}_{\text{list}}(A, a). \tag{12}$$

- **Sub-tree Insertion into List Structure** A function that inserts a new sub-tree (module) S immediately after the attribute a within a list structure:

$$A' = \text{Insert}_{\text{list}}(A, a, S). \tag{13}$$

- **Module Source Code Retrieval** A function that retrieves the source code of specified one or several modules from the module database $\mathcal{D}_M$:

$$C = \text{GetCode}(\{m\}, \mathcal{D}_M), \tag{14}$$

  where m denotes the module name and C denotes the corresponding source code retrieved from the database.

- **Default Parameter Retrieval** A function that retrieves the default initialization parameters of a specified module from the module database $\mathcal{D}_M$:

$$\theta = \text{GetDefault}(m, \mathcal{D}_M), \tag{15}$$

  where m denotes the module name and θ denotes the set of default initialization parameters associated with the module.

- **Tree-to-Text Conversion** A function that converts a nested Python dictionary object into a textual configuration format:

$$A_{text} = \text{Text}(A). \tag{16}$$

- **Text-to-Tree Conversion** A function that converts the textual configuration format back into a nested Python dictionary object:

$$A = \text{Dict}(A_{text}). \tag{17}$$

- **Module Retrieval from Database** This function takes one or more modules as input and retrieves candidate modules from the module database $\mathcal{D}_M$ whose forward interfaces are compatible with the input modules. Specifically, the retrieved modules must have a forward function whose numbers of input and output variables can match those of at least one input module.

$$\mathcal{M} = \text{Retrieve}(\mathcal{S}, \mathcal{D}_M), \tag{18}$$

  where $\mathcal{S} = \{m_1, \ldots, m_k\}$ denotes the set of input module names and $\mathcal{M} \subseteq \mathcal{D}_M$ denotes the set of retrieved candidate module names.

The following operations are executed by using the LLM as a tool in combination with the above algorithm-based tools.

D.2.1 Change Hyperparameter Operation.

As shown in the prompt of Sec. D.4.2, hyperparameter modifications are not handled primarily through dedicated tools; instead, the algorithm only provides coarse guidance, and the LLM directly outputs the resulting architecture. This choice is because changing only one hyperparameter is not efficient.

D.2.2 Swap Module Operation.

We swap modules as shown in Algorithm 3 using the above algorithm-based tools in conjunction with the LLM.

Algorithm 3 Swap Module Operation

Require: Base architecture A_t^{base}, module database $\mathcal{D}_M$, history database $\mathcal{D}_H$, prompt category c, prompt template P

$\mathcal{A} = \{(a_1, m_1), \ldots, (a_K, m_K)\} \leftarrow \mathrm{Attr}(A_t^{base})$ ▷ Extract all module attributes
if $c \in \{\text{rely LLM}, \text{involve LLM}\}$ **then**
  if $P \in \mathcal{P}_{restrict}$ **then**
    Randomly sample a subset $\tilde{\mathcal{A}} \subset \mathcal{A}$ and replace $\mathcal{A} \leftarrow \tilde{\mathcal{A}}$ ▷ We sampled 5
  end if
  $\mathcal{M} \leftarrow \mathrm{Retrieve}(\{m_1, \ldots, m_K\}, \mathcal{D}_M)$ ▷ Candidate module names
  Randomly sample candidate modules $\tilde{\mathcal{M}} \subset \mathcal{M}$ ▷ We sampled 8
  $P' = P.\mathrm{format}(\mathrm{Text}(A_t^{base}), \{a_1, \ldots, a_K\}, \mathrm{GetCode}(\tilde{\mathcal{M}}, \mathcal{D}_M), \mathcal{D}_H)$
  ▷ Fill blanks in the prompt template
  $(a_d, m') = \mathcal{LLM}(P')$ ▷ a_d: swap location, m': new module
else ▷ minimum LLM
  Randomly sample (a_d, m_d) from $\mathcal{A}$
  $\mathcal{M} \leftarrow \mathrm{Retrieve}(\{m_d\}, \mathcal{D}_M)$
  if P requires LLM **then**
    Randomly sample candidate modules $\tilde{\mathcal{M}} \subset \mathcal{M}$
    $P' = P.\mathrm{format}(\mathrm{Text}(A_t^{base}), a_d, \mathrm{GetCode}(\tilde{\mathcal{M}}, \mathcal{D}_M), \mathcal{D}_H)$
    ▷ Fill blanks in the prompt template
    $m' = \mathcal{LLM}(P')$ ▷ Decide m' from $\tilde{\mathcal{M}}$
  else
    m' is randomly sampled from $\mathcal{M}$
  end if
end if
$u = \mathrm{Text}(\mathrm{GetSubTree}(A_t^{base}, a_d))$ ▷ Get parameters used in the previous module
Let P_2 be a prompt template to determine the hyperparameters of m'
$P_2' = P_2.\mathrm{format}(\mathrm{GetCode}(\{m_d, m'\}, \mathcal{D}_M), u, \mathrm{GetDefault}(m', \mathcal{D}_M))$
$S' = \mathrm{Dict}(\mathcal{LLM}(P_2'))$ ▷ Generated sub-tree with the new module
$A_t = \mathrm{Replace}(A_t^{base}, a_d, S')$ ▷ Replace the module at a_d with S'
return transformed architecture A_t

D.2.3 Insert Module Operation.

We insert modules as shown in Algorithm 4.

D.2.4 Remove Module Operation.

We remove modules as shown in Algorithm 5.

D.2.5 Create Module Operation.

We create composite modules as shown in Algorithm 6.

D.3 History Database

The history database is used both to improve efficiency through Thompson sampling in coarse-to-fine decision making and to provide contextual information to the LLM. Here, we describe the details of the textualized history that is passed to the LLM.

First, the operations *"Swap Module," "Insert Module,"* and *"Remove Module"* explicitly track the exact transformation that has been performed. Therefore, we generate accurate history entries using rule-based templates in the following formats:

- `"Change {module_name_pre} at {attribute} into {module_name_new}"`
- `"Insert {module_name_new} at {attribute}"`
- `"Remove {module_name_pre} at {attribute}"`

Algorithm 4 Insert Module Operation

Require: Base architecture A_t^{base}, module database $\mathcal{D}_M$, history database $\mathcal{D}_H$, prompt category c, prompt template P

$\mathcal{A} = \{(a_1, m_1), \ldots, (a_K, m_K)\} \leftarrow \mathrm{Attr}_{\mathrm{list}}(A_t^{base})$
▷ Extract all module attributes in list structures
if $c \in \{\text{rely LLM}, \text{involve LLM}\}$ **then**
  Randomly sample a subset $\tilde{\mathcal{A}} \subset \mathcal{A}$ and replace $\mathcal{A} \leftarrow \tilde{\mathcal{A}}$ ▷ We sampled 5
  $\mathcal{M} \leftarrow \mathrm{Retrieve}(\{m_1, \ldots, m_K\}, \mathcal{D}_M)$ ▷ Candidate module names
  Randomly sample candidate modules $\tilde{\mathcal{M}} \subset \mathcal{M}$ ▷ We sampled 8
  $P' = P.\mathrm{format}(\mathrm{Text}(A_t^{base}), \{a_1, \ldots, a_K\}, \mathrm{GetCode}(\tilde{\mathcal{M}}, \mathcal{D}_M), \mathcal{D}_H)$
  ▷ Fill blanks in the prompt template
  $(a_d, m') = \mathcal{LLM}(P')$ ▷ a_d: insertion location, m': new module
else ▷ minimum LLM
  Randomly sample (a_d, m_d) from $\mathcal{A}$
  $\mathcal{M} \leftarrow \mathrm{Retrieve}(\{m_d\}, \mathcal{D}_M)$
  if P requires LLM **then**
    Randomly sample candidate modules $\tilde{\mathcal{M}} \subset \mathcal{M}$
    $P' = P.\mathrm{format}(\mathrm{Text}(A_t^{base}), a_d, \mathrm{GetCode}(\tilde{\mathcal{M}}, \mathcal{D}_M), \mathcal{D}_H)$ ▷ Fill blanks in the prompt template
    $m' = \mathcal{LLM}(P')$ ▷ Decide m' from $\tilde{\mathcal{M}}$
  else
    m' is randomly sampled from $\mathcal{M}$
  end if
end if
$u = \mathrm{Text}(\mathrm{GetSubTree}(A_t^{base}, a_d))$ ▷ Get parameters used in the previous module
Let P_2 be a prompt template to determine the hyperparameters of m'
$P_2' = P_2.\mathrm{format}(\mathrm{GetCode}(\{m_d, m'\}, \mathcal{D}_M), u, \mathrm{GetDefault}(m', \mathcal{D}_M))$
$S' = \mathrm{Dict}(\mathcal{LLM}(P_2'))$ ▷ Generated sub-tree with the new module
$A_t = \mathrm{Insert}_{\mathrm{list}}(A_t^{base}, a_d, S')$ ▷ Insert S' just after attribute a_d
return transformed architecture A_t

Algorithm 5 Remove Module Operation

Require: Base architecture A_t^{base}, module database $\mathcal{D}_M$, history database $\mathcal{D}_H$, prompt category c, prompt template P

$\mathcal{A} = \{(a_1, m_1), \ldots, (a_K, m_K)\} \leftarrow \mathrm{AttrList}(A_t^{base})$ ▷ Extract all module attributes and corresponding module names
if $c \in \{\text{rely LLM}, \text{inv. LLM}\}$ **then**
  Randomly sample a subset $\tilde{\mathcal{A}} \subset \mathcal{A}$ ▷ We sampled 5 candidates
  $\mathcal{A} \leftarrow \tilde{\mathcal{A}}$
  $P' = P.\mathrm{format}\big(\mathrm{Text}(A_t^{base}), \{a_1, \ldots, a_K\}, \mathrm{GetCode}(\{m_1, \ldots, m_K\}, \mathcal{D}_M), \mathcal{D}_H\big)$
  ▷ Fill blanks in the prompt template
  $a_d = \mathcal{LLM}(P')$ ▷ Attribute to remove, chosen from $\{a_1, \ldots, a_K\}$
else ▷ minimum LLM
  Randomly sample (a_d, m_d) from $\mathcal{A}$
end if
$u = \mathrm{Text}(\mathrm{GetSubTree}(A_t^{base}, a_d))$ ▷ Used parameters of the module to be removed
Let P_2 be a prompt template to generate the full arch. after removing the module
$P_2' = P_2.\mathrm{format}\big(\mathrm{Text}(A_t^{base}), u, \mathrm{GetCode}(\{m_d\}, \mathcal{D}_M)\big)$
$A_t = \mathrm{Dict}(\mathcal{LLM}(P_2'))$ ▷ Generate the architecture after removal
return transformed architecture A_t

On the other hand, the operations *"Change Hyperparameter," "Create Module,"* and *"Repeat Previous"* may involve diverse and complex modifications, making them difficult to handle with rule-based algorithms. Therefore, after the transformation is completed, we prompt the LLM with the instruction: *"Please describe the transformation in short one sentence."* The LLM then generates the textualized history entry.

When presenting the history to the LLM, the entries are shown in a format illustrated in the *Instruction Prompt (Turn 1)* in Sec. D.4.1.

Algorithm 6 Create Module Operation

Require: Base architecture A_t^{base}, module database $\mathcal{D}_M$, history database $\mathcal{D}_H$, prompt category c, prompt template P

Let primitive modules and special modules be $\mathcal{M}_{pytorch}$ and $\mathcal{M}_{special}$
$b_{merge} \leftarrow (P \in \mathcal{P}_{merge})$ ▷ Whether the prompt requires merging multiple modules
if b_{merge} **then** ▷ Merge several modules into a composite module
  $\mathcal{A} \leftarrow \mathrm{Attr}_{\mathrm{list}}(A_t^{base})$ ▷ Extract module attributes in list structures
  Randomly sample (a_d, m_d) from $\mathcal{A}$
  Randomly sample the number of merging modules n
  if $P \in \mathcal{P}_{restrict}$ **then**
    $\mathcal{M}_{candidate} \leftarrow \{m_d, \ldots, m_{d+n}\} \cup \mathcal{M}_{special}$
  else
    Sample custom modules $\mathcal{M}_{custom}$ from $\mathcal{D}_M$
    $\mathcal{M}_{candidate} \leftarrow \{m_d, \ldots, m_{d+n}\} \cup \mathcal{M}_{special} \cup \mathcal{M}_{custom}$
  end if
else ▷ Replace a module with a composite module
  $\mathcal{A} \leftarrow \mathrm{Attr}(A_t^{base})$ ▷ Extract all module attributes
  Randomly sample (a_d, m_d) from $\mathcal{A}$
  if $P \in \mathcal{P}_{restrict}$ **then**
    $\mathcal{M}_{candidate} \leftarrow \{m_d\} \cup \mathcal{M}_{special}$
  else
    Sample custom modules $\mathcal{M}_{custom}$ from $\mathcal{D}_M$
    $\mathcal{M}_{candidate} \leftarrow \{m_d\} \cup \mathcal{M}_{special} \cup \mathcal{M}_{custom}$
  end if
end if
if $c \in \{\text{inv. LLM, minimum LLM}\}$ **then**
  Randomly sample $\tilde{\mathcal{M}}_{pytorch} \subset \mathcal{M}_{pytorch}$
  $\mathcal{M}_{pytorch} \leftarrow \tilde{\mathcal{M}}_{pytorch}$
end if
if $c \in \{\text{rely LLM, inv. LLM}\}$ **then**
  $C' \leftarrow$ None ▷ No constraint
else
  Let C be a constraint template
  Sample $m_s \sim \mathcal{M}_{special}$
  $C' \leftarrow C$.format(m_d, m_s) ▷ A constraint to use at least m_d and m_s
end if
$P' \leftarrow P$.format$(\mathrm{Text}(A_t^{base}), \mathrm{GetCode}(\mathcal{M}_{candidate}, \mathcal{D}_M), \mathrm{GetDefault}(\mathcal{M}_{special}, \mathcal{D}_M),$
  $\mathrm{GetSubTree}(A_t^{base}, a_d), \mathcal{D}_H, C')$
$S_{text} \leftarrow \mathcal{LLM}(P')$ ▷ Ask LLM to create a new module using $\mathcal{M}_{candidate}$ and $\mathcal{M}_{special}$ with constraint C'

$S' \leftarrow \mathrm{Dict}(S_{text})$ ▷ Convert LLM output into a sub-tree
$A_t \leftarrow \mathrm{Replace}(A_t^{base}, a_d, S')$
if b_{merge} **then**
  for $i = n$ **down to** 1 **do**
    $A_t \leftarrow \mathrm{Remove}_{list}(A_t, a_{d+i})$
  end for
end if
return A_t

D.3.1 Repeat Previous Operation.

As shown in the prompt of Sec. D.4.7, hyperparameter modifications are not handled primarily through dedicated tools; instead, the algorithm only provides coarse guidance, and the LLM directly outputs the resulting architecture. This choice is because it is difficult for rule-based algorithm to extract what is the similar directional change to previous transformation.

D.4 Prompts

D.4.1 Overall Pipeline.

In our framework, the final tree transformation is determined through an algorithm-guided instruction process combined with a multi-turn interaction with the LLM.

First, the LLM is provided with the following *system prompt*.

System Prompt

You are a helpful assistant to create a Python-based config file of the MMPretrain. Your mission is to create an innovative model architecture which outperforms previous works. Take account the information you will be provided and create a great model config. Don't talk about dataloader, augmentation, optimizer, and learning rate. You just need to create the model config. You are not allowed to use pretrained weight.

In the first turn, we additionally input the **Instruction Prompt (Turn 1)** shown below. In this prompt, one of the prompt templates described in the following sections is used based on which operation to use. Based on the proposed coarse-to-fine decision making, the algorithm selects which prompt category and prompt template to use. The placeholders in the selected template are then filled using algorithm-based analysis of the tree structure, and the resulting prompt is provided to the LLM.

Afterward, depending on the operation, the algorithm and the LLM interact for several turns, and the final transformation is determined through this dialogue process.

In the next section, we present examples of the prompt templates and multi-turn conversations for each operation.

Instruction Prompt (Turn 1)

Please improve the config: {pre_cfg}

Below is a summary of previous experiments. Analyze what could improve performance and consider what to do next.

Previous experiments
{HISTORY}

Tips for utilizing previous experiments:
- Reverting to previous modules/hyperparameters is not interesting because we already know the performance of the previous model. Do not do that!
- Changing the same positions repeatedly is inefficient. Extract the essence and apply it to other positions.
- We want an innovative model, so trying new things that have never been done is important.

{Operation Specific Prompt Template}

Example of {pre_cfg}

```python
model = dict(
    type='ImageClassifier',
    backbone = ... # See Appendix E for an example.
    neck = ...
    head = ...
)
```

Example of {HISTORY}

Changes: Remove BasicBlock at model.backbone.layer_cfgs[4] Performance: Deteriorated
---
Changes: Added a third ConvFFN at layer_cfgs[31] with 256 internal channels and 0.4 drop path for enhanced feature refinement. Performance: Improved
---
...

D.4.2 CHANGE HYPERPARAMETERS.

We prepare the following prompt templates for *Change Hyperparameters* operation. One of these templates is used as the last part of the **Instruction Prompt (Turn 1)**. In this operation, the algorithm does not explicitly decide which specific parameters to modify because changing only a single parameter at a time is inefficient. The algorithm narrows down a set of candidate locations and instructs the LLM to modify several hyperparameters at once. The LLM then generates the entire tree architecture accordingly.

Prompt Templates for Change Hyperparameters

```
# Category: Rely LLM.
Four templates with following variations.
{GUIDANCE} = [
    "changing a few hyperparameters in the config",
    "changing some important hyperparameters in the config",
    "changing hyperparameters that are likely to affect model performance",
    "modifying hyperparameters which you think weird in the config"
]
 # Category: Inv. LLM.
Three templates with following variations.
{GUIDANCE} = [
    "changing hyperparameters that have not been changed until now",
    "changing hyperparameters to values not tried before",
    "changing hyperparameters to unexpected values"
]
 # Category: Mini. LLM.
14 templates with following variations.
{GUIDANCE} = [
    "slightly increasing the channel width of some layers",
    "slightly decreasing the channel width of some layers",
    "slightly increasing the channel width of the initial layers",
    "slightly decreasing the channel width of the initial layers",
    "slightly increasing the channel width of the middle layers",
    "slightly decreasing the channel width of the middle layers",
    "slightly increasing the channel width of the final layers",
    "slightly decreasing the channel width of the final layers",
    "changing some boolean hyperparameters in the config",
    "changing some float hyperparameters in the config",
    "changing some integer hyperparameters in the config",
    "changing some string hyperparameters in the config",
    "changing the receptive field of some layers",
    "changing the stride of some layers"
]
```

This time, please improve the model's performance by {GUIDANCE}. Change only hyperparameters. Do not add additional hyperparameters. Just change the values of existing hyperparameters. Also, do not change the architecture or modules. In other words, do not change the value of each "type". Please provide a fully modified model config.

D.4.3 SWAP MODULE.

We prepare the following prompt templates for *Swap Module* operation. One of these templates is used as the last part of the **Instruction Prompt (Turn 1)**. As the prompt indicates, in the first turn of the conversation, the LLM determines which part should be replaced with which module if the algorithm has not decided yet.

Prompt Templates for Swap Module

Category: Rely LLM.
2 × 2 = 4 templates with following variations.
{GUIDANCE} = [
"Positions with Some Issues",
"Positions Previously Not Changed Much"
]
{CANDIDATE_ATTRIBUTES} = [
{all_module_attributes},
{sampled_module_attributes}
]

This time, let's try to improve the model's performance by using one of the following modules:
{candidate_module_codes}
Analyze the source codes, especially how they work and what the input and output tensor shapes are, to replace previous modules.
Then, which module should be used and where should it be replaced in the original config?
I will ask later about appropriate hyperparameters, so please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
{GUIDANCE} (More Improvement Needed Positions): XXX
Pros and Cons of the New Candidate Modules: XXX
Input and Output Shape Compatibility of the New Candidate Modules against Previous Modules: XXX
New Module Name to Use: YYY
Where to be Used: ZZZ
##########
Fill XXX based on your analysis.
YYY can be chosen from {candidate_module_names}.
Select ZZZ from {CANDIDATE_ATTRIBUTES}.

Category: Inv. LLM.
Two templates.

This time, let's try to improve the model's performance by using one of the following modules:
{candidate_module_codes},
but I want to try an unexpected module this time. Analyze the source codes, especially how they work and what the input and output tensor shapes are, to replace previous modules.
Then, which module should be used and where should it be replaced in the original config? I will ask later about appropriate hyperparameters, so please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
Positions Previously Not Changed Much (More Improvement Needed Positions): XXX
New Candidate Modules that few researchers would consider (This time, I want to try novel changes): XXX
Input and Output Shape Compatibility of the New Candidate Modules against Previous Modules: XXX
New Module Name to Use: YYY
Where to be Used: ZZZ
##########
Fill XXX based on your analysis.
YYY can be chosen from {candidate_module_names}.
Select ZZZ from {sampled_module_attributes}.

This time, let's try to improve the model's performance by using one of the following modules:
{candidate_module_codes}
Analyze the source codes, especially how they work and what the input and output tensor shapes are, to replace previous modules.
Then, which module should be used and where should it be replaced in the original config?
I will ask later about appropriate hyperparameters, so please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
Positions rarely modified (This time, I want to try novel changes): XXX
Pros and Cons of the New Candidate Modules: XXX
Input and Output Shape Compatibility of the New Candidate Modules against Previous Modules: XXX
New Module Name to Use: YYY
Where to be Used: ZZZ
##########
Fill XXX based on your analysis.
YYY can be chosen from {candidate_module_names}.
Select ZZZ from {sampled_module_attributes}.

Category: Mini. LLM.
Two templates but one of them skips to ask to LLM by randomly deciding where to swap and which module to use.

This time, let's try to improve the model's performance by replacing the module in {decided_module_attribute} with one of the following modules:
{candidate_module_codes}

Analyze the source codes, especially about how they work and what are the input and output tensor shapes to replace with the previous module.
Then, answer which module should be used.
I'll ask later about appropriate hyperparameters, so please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
Pros and Cons of the New Candidate Modules: XXX
Input and Output Shape Compatibility of the New Candidate Modules against Previous Modules: XXX
New Module Name to Use: YYY
##########
Fill XXX based on your analysis.
YYY can be chosen from {sampled_module_names}.

(SKIP)

In the second turn of the conversation, appropriate hyperparameters of the new module are determined using the following instruction prompt. The obtained hyperparameters are then extracted, and the tree structure is actually transformed in an algorithmic manner.

Instruction Prompt for Swap Module Operation (Turn 2)

Now, I want to ask how to replace. The original module, {original_module_name}, is
{original_module_code},
and the used parameters of __init__ function are
{used_parameters}.
The new module, {decided_module_name}, is
{decided_module_code},
and the default parameters of __init__ function are
{decided_module_default_param}.
To replace the original module with the new module, modifications to the default parameters are needed to ensure the input and output tensor shapes match those of the original module.
If there is a "¡TODO¿" in the default parameters, it must be replaced with an appropriate parameter. Moreover, further modification of the default parameters might be desired to improve the performance. However, adding new parameters (keys of the dict) are not allowed.
Considering above, please modify the default parameters of
{decided_module_default_param}.
Note that you need to modify the values of the dict and don't need to modify the keys of the dict.

D.4.4 INSERT MODULE.

We prepare the following prompt templates for *Insert Module* operation. One of these templates is used as the latter part of the **Instruction Prompt (Turn 1)**. If the algorithm has not yet decided, the LLM determines which module should be inserted and where it should be placed.

Prompt Templates for Insert Module

Category: Rely LLM.
Two templates.

This time, let's try to improve the model's performance by inserting an additional module after one of the following positions: {sampled_module_attributes}. We can use one of the following modules:
{candidate_module_codes}
Analyze the source codes, especially about how they work and what are the input and output tensor shapes. Then, answer which module should be used and where it should be inserted. I'll ask later about appropriate hyperparameters, so please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
Positions Previously Not Inserted Much (More Improvement Might be Possible): XXX
Pros and Cons of the Candidate Modules: XXX
Analysis of the Candidate Inserting Positions: XXX
Input and Output Shape Compatibility of the Candidate Modules to Insert: XXX
New Module Name to Use: YYY
Where to be Inserted: ZZZ
##########
Fill XXX based on your analysis.
YYY can be chosen from {candidate_module_names}.
ZZZ can be chosen from {sampled_module_attributes}.

This time, let's try to improve the model's performance by inserting an additional module after one of the following positions: {sampled_module_attributes}. We can use one of the following modules:
{candidate_module_codes}
Analyze the source codes, especially about how they work and what are the input and output tensor shapes. Then, answer which module should be used and where it should be inserted. I'll ask later about appropriate hyperparameters, so please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
Pros and Cons of the Candidate Modules: XXX
Positions Previously Not Inserted Much (More Improvement Might be Possible): XXX
Analysis about Where Lacks the Expressiveness: XXX
Input and Output Shape Compatibility of the Candidate Modules to Insert: XXX
New Module Name to Use: YYY
Where to be Inserted: ZZZ
##########
Fill XXX based on your analysis.
YYY can be chosen from {candidate_module_names}.
ZZZ can be chosen from {sampled_module_attributes}.

Category: Inv. LLM.
Two templates.

This time, let's try to improve the model's performance by inserting an additional module after one of the following positions: {sampled_module_attributes}. We can use one of the following modules:
{candidate_module_codes},
but I want to try an unexpected module this time.
Analyze the source codes, especially about how they work and what are the input and output tensor shapes. Then, answer which module should be used and where it should be inserted. I'll ask later about appropriate hyperparameters, so please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
Positions Previously Not Inserted Much (More Improvement Might be Possible): XXX
New Candidate Modules that few researchers would consider (This time, I want to try novel changes): XXX
Analysis of the Candidate Inserting Positions: XXX
Input and Output Shape Compatibility of the Candidate Modules to Insert: XXX
New Module Name to Use: YYY
Where to be Inserted: ZZZ
##########
Fill XXX based on your analysis.
YYY can be chosen from {candidate_module_names}.
ZZZ can be chosen from {sampled_module_attributes}.

This time, let's try to improve the model's performance by inserting an additional module after one of the following positions: {sampled_module_attributes}, especially focusing on novel changes. We can use one of the following modules:
{candidate_module_codes}
Analyze the source codes, especially about how they work and what are the input and output tensor shapes. Then, answer which module should be used and where it should be inserted. I'll ask later about appropriate hyperparameters, so please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
Pros and Cons of the Candidate Modules: XXX
Positions where Nobody will Insert (This time, I want to try novel changes): XXX
Input and Output Shape Compatibility of the Candidate Modules to Insert: XXX
New Module Name to Use: YYY
Where to be Inserted: ZZZ
##########
Fill XXX based on your analysis.
YYY can be chosen from {candidate_module_names}.
ZZZ can be chosen from {sampled_module_attributes}.

Category: Mini. LLM.
Two templates but one of them skips to ask to LLM by randomly deciding where to replace and which module to use.

This time, let's try to improve the model's performance by inserting an additional module after {decided_module_attribute}. We can use one of the following modules:
{candidate_module_codes}
Analyze the source codes, especially about how they work and what are the input and output tensor shapes to insert after {decided_module_attribute}. Then, answer which module should be used. I'll ask later about appropriate hyperparameters, so please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
Pros and Cons of the Candidate Modules: XXX
Input and Output Shape Compatibility of the Candidate Modules to Insert After {decided_module_attribute}: XXX
New Module Name to Use: YYY

##########
Fill XXX based on your analysis.
YYY can be chosen from {candidate_module_names}.

(SKIP)

In the second turn of the conversation, appropriate hyperparameters of the new module are determined using the following instruction prompt. The obtained hyperparameters are then extracted, and the tree structure is actually transformed in an algorithmic manner.

Instruction Prompt for Insert Module Operation (Turn 2)

Now, I want to ask how to insert it. The previous module, {original_module_name}, is
{original_module_code},
and the used parameters of __init__ function are
{used_parameters}.
The new module, {decided_module_name}, is
{decided_module_code},
and the default parameters of __init__ function are
{decided_module_default_param}.
To insert it, modifications to the default parameters are needed to ensure that the input shape of the new module matches the output shape of the preceding module and that its output shape is compatible with the subsequent module.
If there is a "¡TODO¿" in the default parameters, it must be replaced with an appropriate parameter. Moreover, further modification of the default parameters might be desired to improve the performance. However, adding new parameters (keys of the dict) are not allowed.
Considering above, please modify the default parameters of
{decided_module_default_param}.
Note that you need to modify the values of the dict and don't need to modify the keys of the dict.

D.4.5 REMOVE MODULE.

We prepare the following prompt templates for *Remove Module* operation. One of these templates is used as the latter part of the **Instruction Prompt (Turn 1)**. If the algorithm has not yet decided, the LLM determines which module should be removed.

Prompt Templates for Remove Module

Category: Rely LLM.
Two templates.

This time, let's try to improve the model's performance by removing a module from {sampled_module_attributes}. The source codes of the modules are as follows:
{candidate_module_codes}
Analyze the source codes. Then, answer which module should be removed. I'll ask later about appropriate hyperparameters, so please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
Positions Previously Not Removed Much (More Improvement Might be Possible): XXX
Positions Where You Think Weird: XXX
Where to be Removed: YYY
##########
Fill XXX based on your analysis.
YYY can be chosen from {sampled_module_attributes}.

This time, let's try to improve the model's efficiency by removing a module from {sampled_module_attributes}. The source codes of the modules are as follows:
{candidate_module_codes}
Analyze the source codes. Then, answer which module should be removed. I'll ask later about appropriate hyperparameters, so please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
Positions Previously Not Removed Much (More Improvement Might be Possible): XXX
Where to be Removed: YYY
##########
Fill XXX based on your analysis.
YYY can be chosen from {sampled_module_attributes}.

Category: Inv. LLM.
Two templates.

This time, let's try to improve the model's performance by removing a module from {sampled_module_attributes}, especially focusing on unexpected positions. The source codes of the modules are as follows:
{candidate_module_codes}
Analyze the source codes. Then, answer which module should be removed. I'll ask later about appropriate hyperparameters, so please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
Rarely considered removal locations (This time, I want to try novel changes): XXX
Where to be Removed: YYY
##########
Fill XXX based on your analysis.
YYY can be chosen from {sampled_module_attributes}.

This time, let's try to improve the model's efficiency by removing a module from {sampled_module_attributes}, especially focusing on novel changes. The source codes of the modules are as follows:
{candidate_module_codes}
Analyze the source codes. Then, answer which module should be removed. I'll ask later about appropriate hyperparameters, so please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
Positions where You think Important (This time, I want to try novel changes): XXX
Where to be Removed: YYY
##########
Fill XXX based on your analysis.
YYY can be chosen from {sampled_module_attributes}.

Category: Mini. LLM.
One template which skips to ask to LLM by randomly deciding where to remove.

(SKIP)

In the second turn of the conversation, the complete configuration after removing the decided module is requested. Although we request full architecture generation, the following verification process check whether the intended module is removed.

Instruction Prompt for Remove Module Operation (Turn 2)

Now, I want to ask how to remove the module at {decided_module_attribute}. I want you to create a config without this module. You might need to modify hyperparameters for surrounding modules at {surrounding_module_attributes} to make the model work well without the removed module.
First, refer to the following source codes of relevant modules:
{decided_module_code}
{surrounding_modules_code}
Then, please provide the complete model config after removing the module at {decided_module_attribute} and modifying hyperparameters of surrounding modules as needed.

D.4.6 CREATE MODULE.

We prepare the following prompt templates for *Create Module* operation. One of these templates is used as the latter part of the **Instruction Prompt (Turn 1)**. In this operation, a composite module is created to replace one module or several modules. This operation obtains the final output within a single turn of conversation, and the algorithm replaces the corresponding part of the tree structure with the composite module.

Prompt Templates for Create Module

```
# Category: Rely LLM.
2 × 2 = 4 templates with following variations.
{HOW_AND_WHERE} = [
    "replacing {decided_module_attribute} with",
    "merging {num} modules at {decided_sequential_attributes} into"
]
{CUSTOM_MODULES} = [
```

```
    Original module:{original_module_code},
    Original modules and Custom modules:{original_module_code} + {custom_modules_code}
]
```

This time, let's try to improve the model's performance by {HOW_AND_WHERE} a new module created by combining some of the following modules:
PyTorch modules (I only show the __init__ parameters since you know well):{pytorch_modules_dict}
{CUSTOM_MODULES}
Special modules which flexibly combine these modules with config dictionaries:{special_modules_code}
For example, you can create a module like as follows:

```python
dict(type='ParallelWithConfig',
    module_cfg1=dict(
        type='SequentialWithConfig',
        module_cfgs=[
            dict(type='Conv2d', in_channels=32, out_channels=64,
                kernel_size=1, stride=1, padding=0, dilation=1, groups=1,
                bias=True, padding_mode='zeros', device=None, dtype=None),
            dict(type='GELU', approximate='tanh'),
            dict(type='Conv2d', in_channels=64, out_channels=32,
                kernel_size=3, stride=1, padding=1, dilation=1, groups=1,
                bias=True, padding_mode='zeros', device=None, dtype=None),
            dict(type='Sigmoid')
        ]
    ),
    module_cfg2=dict(type='Identity'),
    merge_operation='mul',
)
```

Note that we have to make the input and output tensor shapes compatible with the previous module(s) with the following parameters:{used_parameters}
Please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
Input and Output Shape of the Previous Module(s): XXX
New Module Configuration:

```python
XXX
```

##########
Fill XXX based on your analysis.

```
# Category: Inv. LLM.
2 × 2 = 4 templates with following variations.
{HOW_AND_WHERE} = [
    "replacing {decided_module_attribute} with",
    "merging {num} modules at {decided_sequential_attributes} into"
]
 {CUSTOM_MODULES} = [
    Original module:{original_module_code},
    Original modules and Custom modules:{original_module_code} + {custom_modules_code}
]
```

This time, let's try to improve the model's performance by {HOW_AND_WHERE} a new module created by combining some of the following modules:
PyTorch modules (I only show the __init__ parameters since you know well):{pytorch_modules_dict}
{CUSTOM_MODULES}
Special modules which flexibly combine these modules with config dictionaries:{special_modules_code}
For example, you can create a module like as follows:

```python
dict(type='ParallelWithConfig',
    module_cfg1=dict(
        type='SequentialWithConfig',
        module_cfgs=[
            dict(type='Conv2d', in_channels=32, out_channels=64,
                kernel_size=1, stride=1, padding=0, dilation=1, groups=1,
                bias=True, padding_mode='zeros', device=None, dtype=None),
            dict(type='GELU', approximate='tanh'),
            dict(type='Conv2d', in_channels=64, out_channels=32,
                kernel_size=3, stride=1, padding=1, dilation=1, groups=1,
                bias=True, padding_mode='zeros', device=None, dtype=None),
```

dict(type='Sigmoid')
]
),
module_cfg2=dict(type='Identity'),
merge_operation='mul',
)
```
Note that we have to make the input and output tensor shapes compatible with the previous module(s) with the following parameters:{used_parameters}
**I expect you to create a novel module configuration that nobody has tried before, so please try to combine the modules in a unique way.**
Please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
Input and Output Shape of the Previous Module: XXX
New Module Configuration:
```python
XXX
```
##########
Fill XXX based on your analysis.

Category: Mini. LLM.
2 × 2 = 4 templates with following variations.
{HOW_AND_WHERE} = [
"replacing {decided_module_attribute} with",
"merging {num} modules at {decided_sequential_attributes} into"
]
{CUSTOM_MODULES} = [
Original module:{original_module_code},
Original modules and Custom modules:{original_module_code} + {custom_modules_code}
]

This time, let's try to improve the model's performance by {HOW_AND_WHERE} a new module created by combining some of the following modules:
PyTorch modules (I only show the __init__ parameters since you know well):{pytorch_modules_dict}
{CUSTOM_MODULES}
Special modules which flexibly combine these modules with config dictionaries:{special_modules_code}
For example, you can create a module like as follows:
```python
dict(type='ParallelWithConfig',
module_cfg1=dict(
type='SequentialWithConfig',
module_cfgs=[
dict(type='Conv2d', in_channels=32, out_channels=64,
kernel_size=1, stride=1, padding=0, dilation=1, groups=1,
bias=True, padding_mode='zeros', device=None, dtype=None),
dict(type='GELU', approximate='tanh'),
dict(type='Conv2d', in_channels=64, out_channels=32,
kernel_size=3, stride=1, padding=1, dilation=1, groups=1,
bias=True, padding_mode='zeros', device=None, dtype=None),
dict(type='Sigmoid')
]
),
module_cfg2=dict(type='Identity'),
merge_operation='mul',
)
```
Note that we have to make the input and output tensor shapes compatible with the previous module(s) with the following parameters:{used_parameters}
This time, please use the {original_module_name} and {random_special_module_name} at least.
Creating a previously unexplored module is important.
Please answer in the following format.
##########
Knowledge from Previous Experiments: XXX
Input and Output Shape of the Previous Module: XXX
New Module Configuration:
```python
XXX
```
##########
```

Fill XXX based on your analysis.

D.4.7 REPEAT PREVIOUS.

We prepare the following prompt for *Repeat Previous* operation. Note that the prompt, **Instruction Prompt (Turn 1)**, is not used. Instead, the following prompts are used to directly generate the modified architecture. Because the history is sparse, we uniformly sample from the predefined prompt variations.

Prompt for Repeat Previous

```
# For all previous operations
{INTRODUCTION} = [
    "From the previous config: {pre_pre_cfg} we created a following config: {pre_cfg} by "{pre_transform}". Then, the accuracy
    was improved from {pre_pre_acc}% to {pre_acc}%.",

    "The previous config was {pre_pre_cfg}. By "{pre_transform}", we successfully achieved better accuracy with the following
    config: {pre_cfg}.",

    "The previous config was {pre_pre_cfg}. From this config, performance is improved by "{pre_transform}". The config is
    {pre_cfg}.",

    "The previous config (Accuracy: {pre_pre_acc}%) was {pre_pre_cfg}. From this config, we improved the accuracy to
    {pre_acc}% by "{pre_transform}". The improved config is {pre_cfg}.",

    "The config: {pre_cfg} is created from the previous config: {pre_pre_cfg} by "{pre_transform}", resulting in an accuracy
    improvement from {pre_pre_acc}% to {pre_acc}%.",

    "By "{pre_transform}", we successfully created a better config: {pre_cfg}.",

    "The config: {pre_cfg} is created by "{pre_transform}"."
]

# Restrict prompt is prepared for each operation
# Change Hyperparameters
{RESTRICTION} = [
    "extracting the essence and refining it",

    "extracting the essence and improving the new config furthermore",

    "making the same change but with a slightly larger magnitude",

    "slightly increasing the strength of the change",

    "slightly increasing the strength of the change in the same positions",

    "applying the same or similar changes to other hyperparameters in the same modules",

    "applying the same or similar changes to other hyperparameters in the surrounding modules",

    "applying the same or similar changes to other hyperparameters in the far away modules",

    "applying the same or similar changes to other hyperparameters in one different module"
]
 # Swap Module
{RESTRICTION} = [
    "extracting the essence and improving the new config furthermore",

    "replacing one or two other positions into {module_new}",

    "replacing a module around {location} into {module_new}",

    "replacing a module in the far away positions from {location} into {module_new}",
]
 # Insert Module
{RESTRICTION} = [
    "extracting the essence and improving the new config furthermore",

    "inserting one more {module_new} before the {location}",

    "inserting one more {module_new} after the {location}",

    "inserting one more {module_new} in a far away position from {location}",

    "inserting one more {module_new} at {random_location}"
]
 # Remove Module
{RESTRICTION} = [
    "extracting the essence and improving the new config furthermore",

    "removing a module at {location} from the new config",

    "removing a module at {location} from the new config",

    "removing a module working similar to {module_pre}",
```

```
    "removing a similar module to {module_pre} around {location}",
    "removing a similar module to {module_pre} far away from {location}"
]
# Create Module and Repeat Previous
{RESTRICTION} = [
    "extracting the essence and improving the new config furthermore",
    "doing something similar to the new config",
    "making a further change in the same direction to the new config"
]

--------------------------------------------------------------------------------
{INTRODUCTION}
I think the change, "{pre_transform}", is a good idea. So, this time, I want you to further improve the performance by {RESTRICTION}.
You can refer the source code of relevant modules below: {relevant_source_code}
Please provide the improved complete config.
--------------------------------------------------------------------------------
```

E DISCOVERED ARCHITECTURES

An example of the discovered architecture is shown below. The tree representation is converted into an OpenMMLab (OpenMMLab, a)-style configuration format, and this format is presented to the LLM as *pre_cfg* in the prompt.

F IMAGENET16-120

Discovered architecture for ImageNet16-120
(Acc. 53.20%, 37.3 MFLOPs, 0.63 #Params[M])

```
model = dict(
    # mmpretrain.models.classifiers.image.ImageClassifier
    type='ImageClassifier',
    backbone=dict(
        # nas_special_modules.NAS_Backbone
        type='NAS_Backbone',
        layer_cfgs=[
            dict(
                # mmcv.cnn.bricks.conv_module.ConvModule
                type='ConvModule',
                in_channels=3,
                out_channels=16,
                kernel_size=3,
                stride=1,
                padding=1,
                dilation=1,
                groups=1,
                bias=False,
                with_spectral_norm=False,
                order=(
                    'conv',
                    'norm',
                    'act',
                ),
                norm_cfg=dict(
                    # torch.nn.BatchNorm2d
                    type='BatchNorm2d',
                    eps=1e-05,
                    momentum=0.1),
                act_cfg=dict(
                    # torch.nn.SiLU
                    type='SiLU',
                    inplace=False)),
            dict(
                # mmpretrain.models.backbones.resnet.BasicBlock
                type='BasicBlock',
                in_channels=16,
                out_channels=16,
```

```
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=1e-05,
                        momentum=0.1),
                    drop_path_rate=0.15,
                    act_cfg=dict(
                        type='SiLU',
                        inplace=False)),
                dict(
                    # nas_special_modules.ParallelWithConfig
                    type='ParallelWithConfig',
                    module_cfg1=dict(
                        # nas_special_modules.SequentialWithConfig
                        type='SequentialWithConfig',
                        module_cfgs=[
                            dict(
                                # torch.nn.Conv2d
                                type='Conv2d',
                                in_channels=16,
                                out_channels=16,
                                kernel_size=3,
                                stride=1,
                                padding=1,
                                dilation=1,
                                groups=1,
                                bias=True,
                                padding_mode='zeros',
                                device=None,
                                dtype=None),
                            dict(
                                type='BatchNorm2d',
                                num_features=16,
                                eps=0.001,
                                momentum=0.1,
                                affine=True,
                                track_running_stats=True,
                                device=None,
                                dtype=None),
                            dict(
                                type='SiLU',
                                inplace=False),
                            dict(
                                # nas_special_modules.MyReshape
                                type='MyReshape',
                                shape=[
                                    -1,
                                    16,
                                    16,
                                    16
                                ]),
                            dict(
                                # timm.layers.bottleneck_attn.BottleneckAttn
                                type='BottleneckAttn',
                                dim=16,
                                dim_out=16,
                                feat_size=(
                                    16,
                                    16,
                                ),
                                stride=1,
                                num_heads=4,
                                dim_head=None,
                                qk_ratio=1.0,
                                qkv_bias=False,
                                scale_pos_embed=True),
                            dict(
                                type='MyReshape',
                                shape=[
                                    -1,
                                    16,
                                    16,
```

```
                                16
                            ]),
                        dict(
                            type='Conv2d',
                            in_channels=16,
                            out_channels=16,
                            kernel_size=3,
                            stride=1,
                            padding=1,
                            dilation=1,
                            groups=1,
                            bias=True,
                            padding_mode='zeros',
                            device=None,
                            dtype=None),
                        dict(
                            type='BatchNorm2d',
                            num_features=16,
                            eps=0.001,
                            momentum=0.1,
                            affine=True,
                            track_running_stats=True,
                            device=None,
                            dtype=None),
                        dict(
                            type='SiLU',
                            inplace=False),
                    ]),
                module_cfg2=dict(
                    type='SequentialWithConfig',
                    module_cfgs=[
                        dict(
                            type='Conv2d',
                            in_channels=16,
                            out_channels=16,
                            kernel_size=1,
                            stride=1,
                            padding=0,
                            dilation=1,
                            groups=1,
                            bias=True,
                            padding_mode='zeros',
                            device=None,
                            dtype=None),
                        dict(
                            type='BatchNorm2d',
                            num_features=16,
                            eps=0.001,
                            momentum=0.1,
                            affine=True,
                            track_running_stats=True,
                            device=None,
                            dtype=None),
                        dict(
                            type='SiLU',
                            inplace=False),
                    ]),
                merge_operation='add',
                concat_dim=1),
            dict(
                type='BasicBlock',
                in_channels=16,
                out_channels=16,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
                conv_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=1e-05,
                    momentum=0.1),
                drop_path_rate=0.15,
                act_cfg=dict(
                    type='SiLU',
                    inplace=False)),
```

```
            dict(
                type='BottleneckAttn',
                dim=16,
                dim_out=None,
                feat_size=(
                    16,
                    16,
                ),
                stride=1,
                num_heads=4,
                dim_head=None,
                qk_ratio=1.0,
                qkv_bias=False,
                scale_pos_embed=True),
            dict(
                type='ParallelWithConfig',
                module_cfg1=dict(
                    type='SequentialWithConfig',
                    module_cfgs=[
                        dict(
                            type='Conv2d',
                            in_channels=16,
                            out_channels=32,
                            kernel_size=3,
                            stride=1,
                            padding=1,
                            dilation=1,
                            groups=1,
                            bias=True,
                            padding_mode='zeros',
                            device=None,
                            dtype=None),
                        dict(
                            type='BatchNorm2d',
                            num_features=32,
                            eps=1e-05,
                            momentum=0.1,
                            affine=True,
                            track_running_stats=True,
                            device=None,
                            dtype=None),
                        dict(
                            type='SiLU',
                            inplace=False),
                        dict(
                            # timm.layers.mlp.ConvMlp
                            type='ConvMlp',
                            in_features=32,
                            hidden_features=64,
                            out_features=32,
                            norm_cfg=dict(
                                type='BatchNorm2d',
                                eps=0.001,
                                momentum=0.1),
                            act_cfg=dict(
                                # torch.nn.GELU
                                type='GELU'),
                            drop=0.1,
                            init_cfg=None),
                        dict(
                            type='BatchNorm2d',
                            num_features=32,
                            eps=1e-05,
                            momentum=0.1,
                            affine=True,
                            track_running_stats=True,
                            device=None,
                            dtype=None),
                        dict(
                            # timm.models.repvit.ConvNorm
                            type='ConvNorm_4',
                            in_dim=32,
                            out_dim=32,
                            ks=3,
                            stride=1,
                            pad=1,
                            dilation=1,
                            groups=1,
```

```
                            bn_weight_init=1),
                        dict(
                            type='ConvNorm_4',
                            in_dim=32,
                            out_dim=32,
                            ks=3,
                            stride=1,
                            pad=1,
                            dilation=1,
                            groups=1,
                            bn_weight_init=1),
                        dict(
                            type='SiLU',
                            inplace=False),
                        dict(
                            type='Conv2d',
                            in_channels=32,
                            out_channels=16,
                            kernel_size=3,
                            stride=1,
                            padding=1,
                            dilation=1,
                            groups=1,
                            bias=True,
                            padding_mode='zeros',
                            device=None,
                            dtype=None),
                        dict(
                            type='BatchNorm2d',
                            num_features=16,
                            eps=1e-05,
                            momentum=0.1,
                            affine=True,
                            track_running_stats=True,
                            device=None,
                            dtype=None),
                        dict(
                            type='SiLU',
                            inplace=False),
                    ]),
                module_cfg2=dict(
                    type='SequentialWithConfig',
                    module_cfgs=[
                        dict(
                            type='Conv2d',
                            in_channels=16,
                            out_channels=16,
                            kernel_size=1,
                            stride=1,
                            padding=0,
                            dilation=1,
                            groups=1,
                            bias=True,
                            padding_mode='zeros',
                            device=None,
                            dtype=None),
                        dict(
                            type='BatchNorm2d',
                            num_features=16,
                            eps=1e-05,
                            momentum=0.1,
                            affine=True,
                            track_running_stats=True,
                            device=None,
                            dtype=None),
                        dict(
                            type='SiLU',
                            inplace=False),
                    ]),
                merge_operation='add',
                concat_dim=1),
            dict(
                type='BasicBlock',
                in_channels=16,
                out_channels=16,
                expansion=1,
                stride=1,
                dilation=1,
```

```
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=1e-05,
                        momentum=0.1),
                    drop_path_rate=0.15,
                    act_cfg=dict(
                        type='SiLU',
                        inplace=False)),
                dict(
                    # mmpretrain.models.utils.inverted_residual.InvertedResidual
                    type='InvertedResidual_3',
                    in_channels=16,
                    out_channels=32,
                    mid_channels=256,
                    kernel_size=3,
                    stride=2,
                    drop_path_rate=0.2,
                    se_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=1e-05,
                        momentum=0.1),
                    act_cfg=dict(type='SiLU')),
                dict(
                    type='BasicBlock',
                    in_channels=32,
                    out_channels=32,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=1e-05,
                        momentum=0.1),
                    drop_path_rate=0.15,
                    act_cfg=dict(
                        type='SiLU',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=32,
                    out_channels=32,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=1e-05,
                        momentum=0.1),
                    drop_path_rate=0.15,
                    act_cfg=dict(
                        type='SiLU',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=32,
                    out_channels=32,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
```

```
                    eps=1e-05,
                    momentum=0.1),
                drop_path_rate=0.15,
                act_cfg=dict(
                    type='SiLU',
                    inplace=False)),
            dict(
                type='BasicBlock',
                in_channels=32,
                out_channels=32,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
                conv_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=1e-05,
                    momentum=0.1),
                drop_path_rate=0.15,
                act_cfg=dict(
                    type='SiLU',
                    inplace=False)),
            dict(
                type='BottleneckAttn',
                dim=32,
                dim_out=None,
                feat_size=(
                    8,
                    8,
                ),
                stride=1,
                num_heads=4,
                dim_head=None,
                qk_ratio=1.0,
                qkv_bias=False,
                scale_pos_embed=True),
            dict(
                type='BasicBlock',
                in_channels=32,
                out_channels=32,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
                conv_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=1e-05,
                    momentum=0.1),
                drop_path_rate=0.15,
                act_cfg=dict(
                    type='SiLU',
                    inplace=False)),
            dict(
                type='BasicBlock',
                in_channels=32,
                out_channels=32,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
                conv_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=1e-05,
                    momentum=0.1),
                drop_path_rate=0.15,
                act_cfg=dict(
                    type='SiLU',
                    inplace=False)),
            dict(
```

```
                type='BasicBlock',
                in_channels=32,
                out_channels=32,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
                conv_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=1e-05,
                    momentum=0.1),
                drop_path_rate=0.15,
                act_cfg=dict(
                    type='SiLU',
                    inplace=False)),
            dict(
                type='BasicBlock',
                in_channels=32,
                out_channels=32,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
                conv_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=1e-05,
                    momentum=0.1),
                drop_path_rate=0.15,
                act_cfg=dict(
                    type='SiLU',
                    inplace=False)),
            dict(
                type='InvertedResidual_3',
                in_channels=32,
                out_channels=64,
                mid_channels=320,
                kernel_size=3,
                stride=2,
                drop_path_rate=0.2,
                se_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=1e-05,
                    momentum=0.1),
                act_cfg=dict(type='SiLU')),
            dict(
                type='BasicBlock',
                in_channels=64,
                out_channels=64,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
                conv_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=1e-05,
                    momentum=0.1),
                drop_path_rate=0.15,
                act_cfg=dict(
                    type='SiLU',
                    inplace=False)),
            dict(
                type='BasicBlock',
                in_channels=64,
                out_channels=64,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
```

```
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=1e-05,
                        momentum=0.1),
                    drop_path_rate=0.15,
                    act_cfg=dict(
                        type='SiLU',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=64,
                    out_channels=64,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=1e-05,
                        momentum=0.1),
                    drop_path_rate=0.15,
                    act_cfg=dict(
                        type='SiLU',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=64,
                    out_channels=64,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=1e-05,
                        momentum=0.1),
                    drop_path_rate=0.15,
                    act_cfg=dict(
                        type='SiLU',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=64,
                    out_channels=64,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=1e-05,
                        momentum=0.1),
                    drop_path_rate=0.15,
                    act_cfg=dict(
                        type='SiLU',
                        inplace=False)),
            ]),
        neck=dict(
            # mmpretrain.models.necks.gap.GlobalAveragePooling
            type='GlobalAveragePooling'),
        head=dict(
            # mmpretrain.models.heads.linear_head.LinearClsHead
            type='LinearClsHead',
            in_channels=64,
            loss=dict(
                # mmpretrain.models.losses.cross_entropy_loss.CrossEntropyLoss
```

```
            type='CrossEntropyLoss_2',
            loss_weight=1.0),
        num_classes=120,
        topk=(
            1,
            5,
        )))
```

G CIFAR-10

Discovered architecture for CIFAR-10
(Acc. 95.31%, 186 MFLOPs, 1.21 #Params[M])

```
model = dict(
    # mmpretrain.models.classifiers.image.ImageClassifier
    type='ImageClassifier',
    backbone=dict(
        # nas_special_modules.NAS_Backbone
        type='NAS_Backbone',
        layer_cfgs=[
            dict(
                # mmcv.cnn.bricks.conv_module.ConvModule
                type='ConvModule',
                in_channels=3,
                out_channels=16,
                kernel_size=7,
                stride=1,
                padding=3,
                dilation=1,
                groups=1,
                bias=False,
                with_spectral_norm=False,
                order=(
                    'conv',
                    'norm',
                    'act',
                ),
                norm_cfg=dict(
                    # torch.nn.BatchNorm2d
                    type='BatchNorm2d',
                    eps=0.001,
                    momentum=0.1),
                act_cfg=dict(
                    # timm.layers.activations_me.SwishMe
                    type='SwishMe',
                    inplace=False)),
            dict(
                # nas_special_modules.ParallelWithConfig
                type='ParallelWithConfig',
                module_cfg1=dict(
                    # nas_special_modules.SequentialWithConfig
                    type='SequentialWithConfig',
                    module_cfgs=[
                        dict(
                            # torch.nn.Conv2d
                            type='Conv2d',
                            in_channels=16,
                            out_channels=32,
                            kernel_size=1,
                            stride=1,
                            padding=0,
                            dilation=1,
                            groups=1,
                            bias=True,
                            padding_mode='zeros',
                            device=None,
                            dtype=None),
                        dict(
                            # torch.nn.GELU
                            type='GELU',
                            approximate='tanh'),
                        dict(
                            # timm.models.swiftformer.
                            # EfficientAdditiveAttention
```

```
                            type='EfficientAdditiveAttention',
                            in_dims=32,
                            token_dim=16,
                            num_heads=1),
                    ]),
                module_cfg2=dict(
                    type='SequentialWithConfig',
                    module_cfgs=[
                        dict(
                            type='Conv2d',
                            in_channels=16,
                            out_channels=16,
                            kernel_size=3,
                            stride=1,
                            padding=1,
                            dilation=1,
                            groups=1,
                            bias=True,
                            padding_mode='zeros',
                            device=None,
                            dtype=None),
                        dict(
                            type='GELU',
                            approximate='tanh'),
                        dict(
                            type='Conv2d',
                            in_channels=16,
                            out_channels=16,
                            kernel_size=1,
                            stride=1,
                            padding=0,
                            dilation=1,
                            groups=1,
                            bias=True,
                            padding_mode='zeros',
                            device=None,
                            dtype=None),
                    ]),
                merge_operation='mul',
                concat_dim=1),
            dict(
                # mmcv.cnn.bricks.context_block.ContextBlock
                type='ContextBlock',
                in_channels=16,
                ratio=0.5,
                pooling_type='att',
                fusion_types=[
                    'channel_add',
                    'channel_mul',
                ]),
            dict(
                # mmpretrain.models.backbones.resnet.BasicBlock
                type='BasicBlock',
                in_channels=16,
                out_channels=16,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
                conv_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=0.001,
                    momentum=0.1),
                drop_path_rate=0.15,
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                type='BasicBlock',
                in_channels=16,
                out_channels=16,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
```

```
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.15,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=16,
                    out_channels=16,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.15,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=16,
                    out_channels=16,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.15,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    # mmpretrain.models.utils.inverted_residual.
                    # InvertedResidual
                    type='InvertedResidual_3',
                    in_channels=16,
                    out_channels=32,
                    mid_channels=1536,
                    kernel_size=3,
                    stride=2,
                    drop_path_rate=0.3,
                    se_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=32,
                    out_channels=32,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
```

```
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=0.001,
                    momentum=0.1),
                drop_path_rate=0.2,
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                type='BasicBlock',
                in_channels=32,
                out_channels=32,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
                conv_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=0.001,
                    momentum=0.1),
                drop_path_rate=0.2,
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                type='BasicBlock',
                in_channels=32,
                out_channels=32,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
                conv_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=0.001,
                    momentum=0.1),
                drop_path_rate=0.2,
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                type='BasicBlock',
                in_channels=32,
                out_channels=32,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
                conv_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=0.001,
                    momentum=0.1),
                drop_path_rate=0.2,
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                type='InvertedResidual_3',
                in_channels=32,
                out_channels=64,
                mid_channels=2048,
                kernel_size=3,
                stride=2,
                drop_path_rate=0.4,
                se_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=0.001,
                    momentum=0.1),
```

```
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                type='BasicBlock',
                in_channels=64,
                out_channels=64,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
                conv_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=0.001,
                    momentum=0.1),
                drop_path_rate=0.25,
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                type='BasicBlock',
                in_channels=64,
                out_channels=64,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
                conv_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=0.001,
                    momentum=0.1),
                drop_path_rate=0.25,
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                # mmpretrain.models.backbones.vig.FFN
                type='FFN',
                in_features=64,
                hidden_features=1024,
                out_features=64,
                act_cfg=dict(type='SwishMe', inplace=False),
                drop_path=0.25),
            dict(
                type='BasicBlock',
                in_channels=64,
                out_channels=64,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
                conv_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=0.001,
                    momentum=0.1),
                drop_path_rate=0.25,
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                type='BasicBlock',
                in_channels=64,
                out_channels=64,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
```

```
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.25,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=64,
                    out_channels=64,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.25,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    # mmpretrain.models.backbones.tinyvit.MBConvBlock
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=2,
                    drop_path=0.25,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=2,
                    drop_path=0.25,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=4,
                    drop_path=0.3,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=4,
                    drop_path=0.3,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=4,
                    drop_path=0.3,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
```

```
                    expand_ratio=6,
                    drop_path=0.35,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=6,
                    drop_path=0.35,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=6,
                    drop_path=0.35,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=8,
                    drop_path=0.5,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=8,
                    drop_path=0.5,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=8,
                    drop_path=0.5,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=8,
                    drop_path=0.5,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
            ]),
        neck=dict(
            # mmpretrain.models.necks.gap.GlobalAveragePooling
            type='GlobalAveragePooling'
        ),
        head=dict(
            # mmpretrain.models.heads.linear_head.LinearClsHead
            type='LinearClsHead',
            in_channels=64,
            loss=dict(
                # mmpretrain.models.losses.cross_entropy_loss.
                # CrossEntropyLoss
                type='CrossEntropyLoss_2',
                loss_weight=1.0),
            num_classes=100,
            topk=(
                1,
                5,
            )))
```

H CIFAR-100

Discovered architecture for CIFAR-100
(Acc. 77.83%, 184 MFLOPs, 1.45 #Params[M])

```
model = dict(
    # mmpretrain.models.classifiers.image.ImageClassifier
    type='ImageClassifier',
    backbone=dict(
        # nas_special_modules.NAS_Backbone
        type='NAS_Backbone',
        layer_cfgs=[
            dict(
                # mmcv.cnn.bricks.conv_module.ConvModule
                type='ConvModule',
                in_channels=3,
                out_channels=16,
                kernel_size=7,
                stride=1,
                padding=3,
                dilation=1,
                groups=1,
                bias=False,
                with_spectral_norm=False,
                order=(
                    'conv',
                    'norm',
                    'act',
                ),
                norm_cfg=dict(
                    # torch.nn.BatchNorm2d
                    type='BatchNorm2d',
                    eps=0.001,
                    momentum=0.1),
                act_cfg=dict(
                    # timm.layers.activations_me.SwishMe
                    type='SwishMe',
                    inplace=False)),
            dict(
                # nas_special_modules.ParallelWithConfig
                type='ParallelWithConfig',
                module_cfg1=dict(
                    # nas_special_modules.SequentialWithConfig
                    type='SequentialWithConfig',
                    module_cfgs=[
                        dict(
                            # torch.nn.Conv2d
                            type='Conv2d',
                            in_channels=16,
                            out_channels=32,
                            kernel_size=1,
                            stride=1,
                            padding=0,
                            dilation=1,
                            groups=1,
                            bias=True,
                            padding_mode='zeros',
                            device=None,
                            dtype=None),
                        dict(
                            # torch.nn.GELU
                            type='GELU',
                            approximate='tanh'),
                        dict(
                            # timm.models.swiftformer.
                            # EfficientAdditiveAttention
                            type='EfficientAdditiveAttention',
                            in_dims=32,
                            token_dim=16,
                            num_heads=1),
                    ]),
                module_cfg2=dict(
                    type='SequentialWithConfig',
                    module_cfgs=[
                        dict(
                            type='Conv2d',
                            in_channels=16,
```

```
                                out_channels=16,
                                kernel_size=3,
                                stride=1,
                                padding=1,
                                dilation=1,
                                groups=1,
                                bias=True,
                                padding_mode='zeros',
                                device=None,
                                dtype=None),
                            dict(
                                type='GELU',
                                approximate='tanh'),
                            dict(
                                type='Conv2d',
                                in_channels=16,
                                out_channels=16,
                                kernel_size=1,
                                stride=1,
                                padding=0,
                                dilation=1,
                                groups=1,
                                bias=True,
                                padding_mode='zeros',
                                device=None,
                                dtype=None),
                        ]),
                    merge_operation='mul',
                    concat_dim=1),
                dict(
                    # mmcv.cnn.bricks.context_block.ContextBlock
                    type='ContextBlock',
                    in_channels=16,
                    ratio=0.5,
                    pooling_type='att',
                    fusion_types=[
                        'channel_add',
                        'channel_mul',
                    ]),
                dict(
                    # mmpretrain.models.backbones.resnet.BasicBlock
                    type='BasicBlock',
                    in_channels=16,
                    out_channels=16,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.15,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=16,
                    out_channels=16,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.15,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
```

```
                dict(
                    type='BasicBlock',
                    in_channels=16,
                    out_channels=16,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.15,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=16,
                    out_channels=16,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.15,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    # mmpretrain.models.utils.inverted_residual.
                    # InvertedResidual
                    type='InvertedResidual_3',
                    in_channels=16,
                    out_channels=32,
                    mid_channels=1536,
                    kernel_size=3,
                    stride=2,
                    drop_path_rate=0.3,
                    se_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=32,
                    out_channels=32,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.2,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=32,
```

```
                    out_channels=32,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.2,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=32,
                    out_channels=32,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.2,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=32,
                    out_channels=32,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.2,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='InvertedResidual_3',
                    in_channels=32,
                    out_channels=64,
                    mid_channels=2048,
                    kernel_size=3,
                    stride=2,
                    drop_path_rate=0.4,
                    se_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=64,
                    out_channels=64,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
```

```
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.25,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=64,
                    out_channels=64,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.25,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    # mmpretrain.models.backbones.vig.FFN
                    type='FFN',
                    in_features=64,
                    hidden_features=1024,
                    out_features=64,
                    act_cfg=dict(type='SwishMe', inplace=False),
                    drop_path=0.25),
                dict(
                    type='BasicBlock',
                    in_channels=64,
                    out_channels=64,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.25,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='BasicBlock',
                    in_channels=64,
                    out_channels=64,
                    expansion=1,
                    stride=1,
                    dilation=1,
                    downsample=None,
                    style='pytorch',
                    with_cp=False,
                    conv_cfg=None,
                    norm_cfg=dict(
                        type='BatchNorm2d',
                        eps=0.001,
                        momentum=0.1),
                    drop_path_rate=0.25,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='BasicBlock',
```

```
                in_channels=64,
                out_channels=64,
                expansion=1,
                stride=1,
                dilation=1,
                downsample=None,
                style='pytorch',
                with_cp=False,
                conv_cfg=None,
                norm_cfg=dict(
                    type='BatchNorm2d',
                    eps=0.001,
                    momentum=0.1),
                drop_path_rate=0.25,
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                # mmpretrain.models.backbones.tinyvit.MBConvBlock
                type='MBConvBlock',
                in_channels=64,
                out_channels=64,
                expand_ratio=2,
                drop_path=0.25,
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                type='MBConvBlock',
                in_channels=64,
                out_channels=64,
                expand_ratio=2,
                drop_path=0.25,
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                type='MBConvBlock',
                in_channels=64,
                out_channels=64,
                expand_ratio=4,
                drop_path=0.3,
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                type='MBConvBlock',
                in_channels=64,
                out_channels=64,
                expand_ratio=4,
                drop_path=0.3,
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                type='MBConvBlock',
                in_channels=64,
                out_channels=64,
                expand_ratio=4,
                drop_path=0.3,
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                type='MBConvBlock',
                in_channels=64,
                out_channels=64,
                expand_ratio=6,
                drop_path=0.35,
                act_cfg=dict(
                    type='SwishMe',
                    inplace=False)),
            dict(
                type='MBConvBlock',
                in_channels=64,
                out_channels=64,
                expand_ratio=6,
                drop_path=0.35,
```

```
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=6,
                    drop_path=0.35,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=8,
                    drop_path=0.5,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=8,
                    drop_path=0.5,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=8,
                    drop_path=0.5,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
                dict(
                    type='MBConvBlock',
                    in_channels=64,
                    out_channels=64,
                    expand_ratio=8,
                    drop_path=0.5,
                    act_cfg=dict(
                        type='SwishMe',
                        inplace=False)),
            ]),
        neck=dict(
            # mmpretrain.models.necks.gap.GlobalAveragePooling
            type='GlobalAveragePooling'
        ),
        head=dict(
            # mmpretrain.models.heads.linear_head.LinearClsHead
            type='LinearClsHead',
            in_channels=64,
            loss=dict(
                # mmpretrain.models.losses.cross_entropy_loss.
                # CrossEntropyLoss
                type='CrossEntropyLoss_2',
                loss_weight=1.0),
            num_classes=100,
            topk=(
                1,
                5,
            )))
```