Structuring Open-Ended NAS: Semi-Automated Design Knowledge Structuring with LLMs for Efficient Neural Architecture Search

Yuiko Sakuma[1] **Masakazu Yoshimura**[1] **Marcel Gröpl**[2]
Zitang Sun[1] **Junji Otsuka**[1] **Atsushi Irie**[1] **Takeshi Ohashi**[1]
[1]Sony Group Corporation, Tokyo, Japan [2]ETH Zürich
Yuiko.Sakuma@sony.com

Abstract

Current neural architecture search (NAS) methods are often limited by their predefined, restrictive search spaces. While recent large language model (LLM)-assisted NAS methods enable open-ended search spaces, they often suffer from inefficient exploration due to biased or low-quality design ideas. To address these issues, we propose to semi-automatically structure model design knowledge to guide the search process. Our approach first defines a high-level structural template of architectural attributes. An LLM then populates this template by analyzing papers, creating a rich and diverse search space that embodies this structured design knowledge. To efficiently explore this vast space, we introduce FairNAD, using a multi-type mutation that enables broad exploration through mutation with fair idea sampling, Pareto-aware mutation, LLM-driven iterative mutation, and a fine-grained feedback loop. We demonstrate the effectiveness of FairNAD in discovering high-performing architectures that yield 0.84, 2.17, and 2.35 points improvement on CIFAR-10, CIFAR-100, and ImageNet16-120, respectively, compared to current state-of-the-art methods.

1 Introduction

Designing high-performance neural network architectures remains a challenge, as manual architecture design heavily relies on an extensive trial-and-error process conducted by experts. This process is not only time-consuming but also confines the search to the scope of the designer's expertise, which limits the performance improvements of the resulting models. To address these issues, Neural Architecture Search (NAS) has been extensively investigated as an approach to automate the design process, aiming to reduce search costs while enabling a broader search [29, 62]. However, many conventional NAS methods depend on a closed search space, predefined by human experts. This means the search is restricted to combinations of existing operators and blocks, fundamentally limiting the ability to generate innovative architectures.

In recent years, a new paradigm has emerged to overcome this limitation: leveraging Large Language Models (LLMs) for NAS. LLM-based NAS adopts an approach of directly generating the source code of a neural network model, either entirely or partially [6, 37, 60, 56]. This enables an "open-ended" search, removing the limitations imposed by the predefined search space in traditional methods. As a result, LLM-based NAS facilitates the discovery of novel operations, modules, and network topologies, potentially enabling the design of high-performing architectures. Moreover, recent studies [56] have shown that external design knowledge, such as architectural knowledge extracted from papers, can further enrich this open-ended search process.

However, open-endedness alone does not guarantee efficient exploration. The effectiveness of knowledge-guided open-ended NAS critically depends on how the underlying design knowledge

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

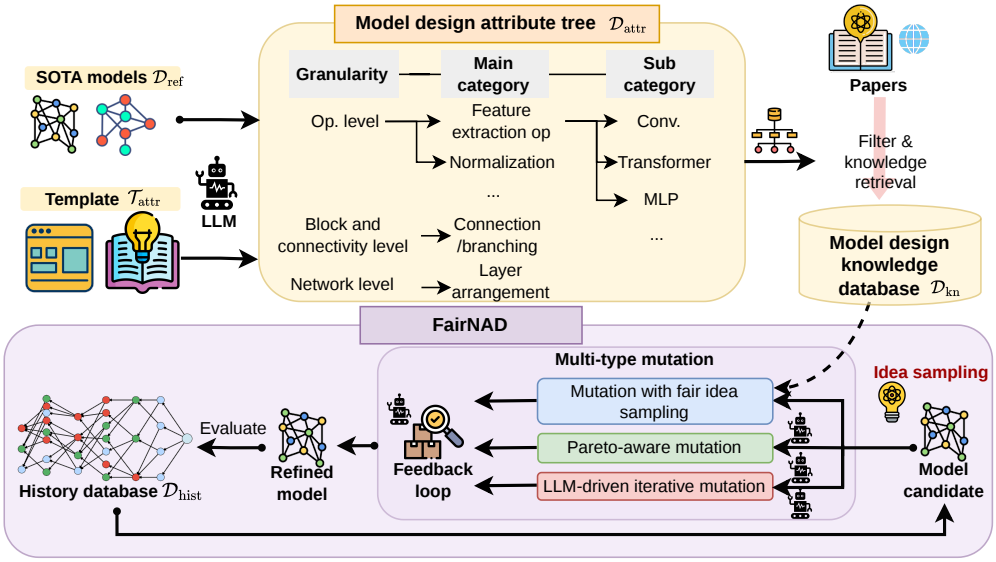

Figure 1: *Overview of the proposed NAS.* (**Top**) The model design attribute tree is generated from state-of-the-art (SOTA) models and a structured template. This tree is used to extract high-quality, fine-grained model design knowledge. (**Bottom**) FairNAD, a LLM-driven framework, then searches for high-performing models using mutation with fair idea sampling, Pareto-aware mutation, LLM-driven iterative mutation, and feedback loop.

is collected, organized, and sampled. In existing methods, design knowledge is directly obtained from LLM priors [60], network architectures [61], papers [56], or web resources [10], and is used in a relatively flat and unstructured manner. These methods retrieve and utilize design knowledge without considering its scope of application (e.g., whether it pertains to an operation, connectivity or the entire network) and its attribute coverage (e.g., neural operators may fall into categories such as convolutions, Transformers, multi-layer perceptrons, and pooling). As a result, knowledge with different levels of abstraction, quality, and applicability are mixed together, making it difficult to effectively retrieve and utilize useful knowledge. Consequently, the search space may include noisy, low-quality, or overly domain-specific knowledge, and may also inherit biases from LLM corpora and research trends. For example, if the knowledge pool is largely constructed from papers published during a period dominated by Transformer-based improvements, the resulting search process may overemphasize that direction and narrow the scope of exploration.

In this work, we address the inefficiency of open-ended NAS by proposing to structure the design knowledge for open-ended NAS. In contrast to prior open-ended methods such as [56], which draws design knowledge from an unstructured knowledge pool, we curate the knowledge using a structured model design attribute template. This structured representation enables filtering of overly domain-specific knowledge, collection of higher-quality design ideas into a model design knowledge database, and mitigation of biases inherited from LLM priors and research trends. Further, in a vast open-ended search space, unstructured and performance-driven search strategies remain inefficient. In particular, strategies that sample design ideas from a flat knowledge pool, select parent architectures mainly according to validation performance, and discard candidates that violate HW constraints may suffer from three limitations: (1) they overlook diverse model candidates, particularly failing to select compact yet high-performing models for their size; (2) they lack intensive exploration of promising models; and (3) diversity is reduced by discarding models that exceed hardware (HW) constraints. Thus, building on the structured knowledge space, we propose FairNAD, a framework which employs a multi-stage search process for efficient and effective open-ended neural architecture design (NAD). First, FairNAD introduces the *mutation with fair idea sampling* stage which enables a category-balanced and uniform exploration of a vast design space by retrieving design ideas based on the structured model knowledge pool. In particular, the idea is sampled according to its attributes, referred to as fair idea sampling. Second, the *Pareto-aware mutation* stage performs Pareto-aware sampling under HW constraints, together with upscaling and hyperparameter adjustment, so that

compact yet promising models are not overlooked during the evolution. Third, the *LLM-driven iterative mutation* stage intensively refines the most promising models to accelerate convergence. Furthermore, the *feedback loop* is incorporated to downsize any models that exceed HW constraints, thereby encouraging the diversity of candidate models by retaining models that would otherwise be discarded.

We demonstrate the effectiveness of FairNAD in designing high-performing architectures by exploring the open-ended search space efficiently and effectively. The experiments on NAS-Bench-201 achieves state-of-the-art performance over previous methods. Our contributions are summarized as follows:

- We introduce a semi-automated model design attribute structuring method that organizes design knowledge into a hierarchical attribute tree. Based on this structured representation, we curate and organize diverse model design ideas from papers.
- We introduce FairNAD, using a multi-type mutation designed to efficiently explore an open-ended search space. First, the *mutation with fair idea sampling* stage uniformly explores a wide range of model design ideas based on their design attribute. Second, the *Pareto-aware mutation* stage scales up compact yet high-performing models. Third, the *LLM-driven iterative mutation* stage intensively refines the most promising models. This multi-stage approach enhances both the diversity of candidate models and search efficiency. Furthermore, each model generation process incorporates a *feedback loop*. This loop verifies and refines generated models. In particular, models exceeding the HW constraints are scaled down which further promotes architectural diversity.

2 Related Works

2.1 Neural Architecture Search (NAS)

Neural Architecture Search (NAS) aims to automatically explore optimal neural network architectures. Early NAS work demonstrated that reinforcement-learning controllers can generate competitive architectures [38, 62, 63], and that evolutionary strategies can discover strong image classifiers at scale [28, 43, 45]. Supernet-based NAS typically adopts a weight-sharing strategy to avoid training each subnet from scratch [3, 4, 12, 29, 44, 50, 58]. Prior work has also extended supernet-based NAS to Transformer architectures [7, 8]. Recently, GNN-based methods have been proposed for efficient optimization [34, 35, 41]. However, these methods are constrained by their search spaces, which limit their ability to discover novel architectures.

2.2 LLM-Based NAS

Recent works propose to use LLMs in evolutionary-guided NAS by directly prompting them to modify the architecture. Initial attempts [6, 27, 37, 60, 39] rely on predefined prompts or the LLM's internal knowledge, resulting in a narrow search space or an inherent dependence on the LLM's pretrained knowledge. To broaden this, some methods [10, 56, 60, 61], incorporate external sources for exploration. For example, [61] transfers the design principles of a predefined set of SOTA architectures to initialize the search process. Going further, [10, 56] mine current papers or web sources, achieving broader architectural diversity. However, this grounding in recent publications introduces a bias toward research trends, potentially limiting discoverable architectures.

3 Problem Statement

This paper addresses search efficiency in open-ended NAS, a paradigm that moves beyond predefined, hand-crafted search spaces. Our open-ended search is enabled by directly generating source code of the architectures (e.g., in PyTorch) using an LLM. Unlike conventional hand-crafted NAS, which composes architectures from a predefined set of blocks (e.g., 3x3/5x5 convolutions, Transformer layers) and adheres to fixed topologies (e.g., ResNet-based), an LLM-based method enables the design of new blocks and modifications to the fundamental dynamics. We term this exploration process within open-ended NAS as Neural Architecture Design (NAD). Our approach contrasts with prior LLM-based code generation methods that are ideally NAD, such as GENIUS [60] and LLMatic [37]. These operate on a limited set of prompts and thus represent a more constrained problem setting.

Our method, following prior work such as NADER [56], constructs the search space from diverse external sources like papers, enabling code generation from a wide variety of prompts.

The scope of this research focuses on discovering backbones for vision models. We do not address the design of the neck and head architectures for downstream tasks. Architecture design is defined as a search problem to find a valid model of optimal architecture a^* that maximizes a metric subject to HW constraints such as floating-point operations per second (FLOPs) or parameter size. A valid model is defined as a syntactically correct, executable (e.g., can be initialized, differentiable, and forward and backward defined), and multi-layer PyTorch model code.

4 Method

We propose an LLM-based NAS method as illustrated in Fig. 1. The overall procedure is as follows. First, the source code of the existing state-of-the-art models is analyzed using an LLM to construct a model design attribute tree ($\mathcal{D}_{\text{attr}}$). This process identifies the attributes that describe how existing architectures are composed and what elements they use, and enumerates potential operations for improvement. Second, papers are analyzed using an LLM combined with $\mathcal{D}_{\text{attr}}$ to generate diverse model design ideas. At the same time, papers that are irrelevant to architectural design are filtered out, resulting in the construction of a curated and structured model design knowledge database ($\mathcal{D}_{\text{kn}}$). Third, in the evolutionary search phase, a parent model that has been previously generated and evaluated is sampled from the evolution tree, while modification ideas are sampled from $\mathcal{D}_{\text{kn}}$. Based on these ideas, multi-type mutations are performed.

In this paper, there are two main proposals over previous work: (1) enhancing the idea collection phase through the semi-automatic structuring of model design knowledge and (2) introducing a FairNAD framework that integrates (a) mutation with fair idea sampling, (b) Pareto-aware mutation, (c) an LLM-driven iterative mutation, and (d) combining the search process with a feedback loop.

4.1 Semi-automated Model Design Attribute Structuring

Extracting model design knowledge from external sources, such as papers, by simply prompting an LLM with a general query like "extract model design ideas from this paper" can lead to outputs heavily biased by the LLM's internal knowledge and research trends. To illustrate this, we analyzed the frequency of specific keywords in the attributes of ideas within $\mathcal{D}_{\text{kn}}$ constructed from the sources used in the prior work [56]. These attributes were tagged by the method described later in this section. As shown in Figure 2(**top**), approximately 70% of the ideas for feature extraction operations are related to attention. Similarly, Figure 2(**bottom**) indicates that most ideas about blocks and connectivity are related to adaptive feature recalibration. Furthermore, as illustrated in Figure 3 (**(middle)**), some ideas are irrelevant to architecture design (e.g., including terms like "loss") or overly task- or domain-specific (e.g., including terms like "3D").

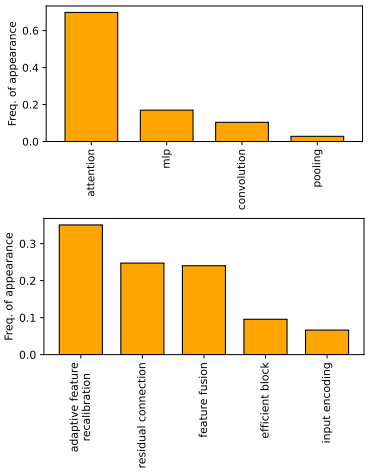

Figure 2: Example of frequency of model design ideas for (**top**) feature extracting operations and (**bottom**) block and connectivity.

To address this challenge and facilitate the extraction of broad and comprehensive knowledge relevant to architecture design, we employ a structured model design attribute tree, $\mathcal{D}_{\text{attr}}$, as an example when querying the LLM. The creation of this tree presents a challenge; a purely hand-crafted design would be constrained by the engineer's knowledge, whereas a fully automated generation using an LLM would be biased. Therefore, we propose a semi-automated approach. This method involves analyzing commonly used performant models as a reference with a predefined attribute template to systematically list model design attributes, striking a balance between manual design and full automation.

As illustrated in Figure 1 (**top**), the proposed model design attribute generator takes two inputs: an attribute template ($\mathcal{T}_{\text{attr}}$) and a set of reference model codes ($\mathcal{D}_{\text{ref}}$). The output is a model design

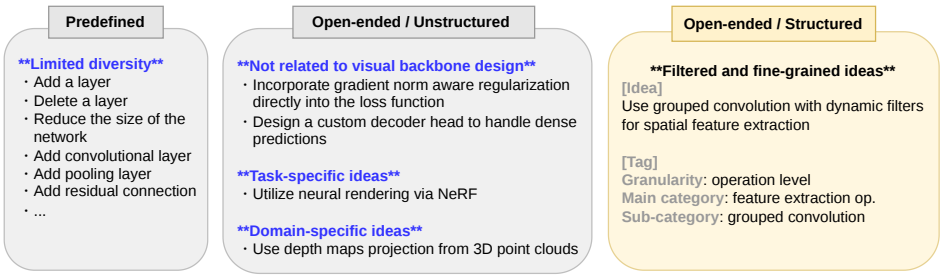

Figure 3: The difference of the model design ideas between (**left**) predefined [37], (**middle**) open-ended and unstructured [56], and (**right**) open-ended and structured methods.

attribute tree ($\mathcal{D}_{\text{attr}}$). We formulate a three-level tree structure within $\mathcal{T}_{\text{attr}}$. This hierarchy consists of granularity (i.e., operation, block and connectivity, network), main attributes (i.e., categories specific to each granularity, such as feature extraction operation at the operation level), and sub-attributes (i.e., finer-grained categories within each main attribute, such as grouped convolution). The top two levels (e.g., granularity and main category) were predefined based on expert knowledge, while the sub-attributes were generated by prompting an LLM. For each reference model $a^{\text{ref}} \in \mathcal{D}_{\text{ref}}$, the LLM is queried with $\mathcal{T}_{\text{attr}}$ to extract the corresponding sub-attributes. $\mathcal{T}_{\text{attr}}$ was designed for different targets, performance and efficiency improvements.

4.2 Knowledge Extraction Using Model Design Attribute Tree

To build a high-quality, broad, and diverse knowledge base, we utilize our attribute tree to distill design ideas from papers. The process is designed to filter overly domain- or task-specific and irrelevant ideas related to architectural design and to encourage the extraction of fine-grained and diverse ideas from papers. This module takes a paper pool and $\mathcal{D}_{\text{attr}}$ as input, and constructs a model design knowledge database, $\mathcal{D}_{\text{kn}}$. The procedure begins with a filtering stage. After scraping papers to structure the paper pool, the LLM is queried to classify each abstract against $\mathcal{D}_{\text{attr}}$. Papers that fail to align with any attribute are discarded. In the subsequent extraction stage, the full text of the filtered papers is analyzed. The LLM is queried to read the method section if the abstract refers to any $\mathcal{D}_{\text{attr}}$ element. The LLM is then queried to extract the associated model design ideas that can inspire architecture design. Potentially, multiple ideas are extracted from a single paper. Each idea is tagged with the corresponding element at each level of the hierarchy of $\mathcal{D}_{\text{attr}}$ (i.e., granularity, main category and sub-category). The resulting ideas are then stored in $\mathcal{D}_{\text{kn}}$.

The examples of model design ideas from different methods are illustrated in Figure 3. While predefined methods [37] rely on a limited set of design concepts, open-ended methods [56] explore a diverse range of ideas, offering the potential to explore a wider variety of architectures. However, a challenge with these open-ended methods is that some generated ideas can be excessively domain- or task-specific, or even irrelevant to visual backbone design. Our proposed method, which is both open-ended and structured, addresses this limitation by systematically curating these ideas. Furthermore, it tags each idea with specific attributes that are utilized later for effective sampling.

4.3 FairNAD

In open-ended NAS, simply mutating models using uniformly sampled ideas is inefficient for exploring the vast search space. To address this inefficiency, we propose FairNAD, using a multi-type mutation as illustrated in Figure 4. The core of our framework is an evolutionary search which iteratively modifies the parent model's code a'_{par} to generate the child model's code a_{cur} to discover a high-performing optimized architecture, a^*. Here, a_0 is initialized with a base model a^{base}. Specifically, a'_{par} is mutated into an intermediate model a'_{cur} with an idea op_t. Then, a'_{cur} is transformed by the feedback loop into the final model, a_{cur}. Finally, a_{cur} is checked for validity and, if valid, a_{cur} is trained and evaluated. The evaluation metric (e.g., accuracy $\text{acc}(\text{a}_{\text{cur}})$) is added in the history database $\mathcal{D}_{\text{hist}}$ with a'_{cur}, op_{cur}, and HW budgets $\text{budget}_{\text{i}}(a_{\text{cur}}), i \in \{1, ..., n\}$ as follows:

$$\mathcal{D}_{\text{hist}} \leftarrow \mathcal{D}_{\text{hist}} \cup \{(a_{\text{cur}}, \text{acc}(a_{\text{cur}}), \text{op}_{\text{cur}}, (\text{budget}_1(a_{\text{cur}}), \ldots, \text{budget}_n(a_{\text{cur}}))) \mid \text{is_valid}(a_{\text{cur}})\} \tag{1}$$

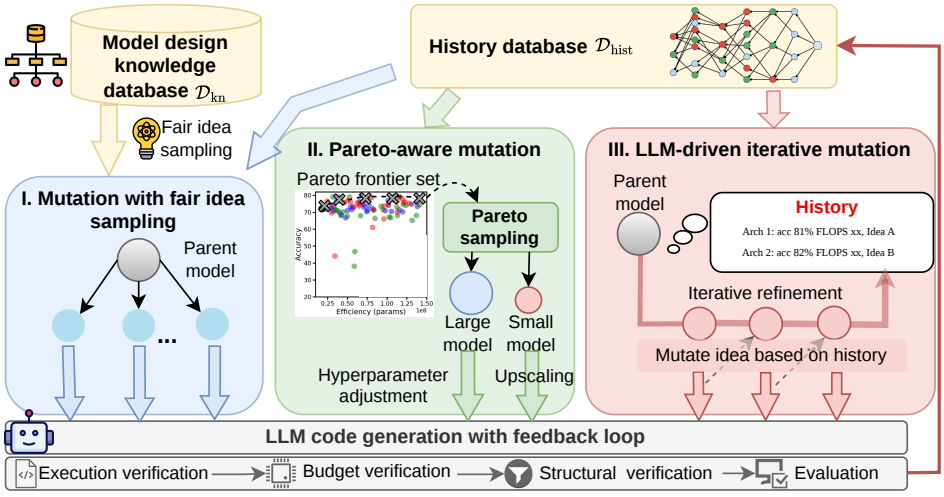

Figure 4: *FairNAD employs a multi-type mutation to balance exploration and exploitation.* (**I**) Model design ideas are uniformly sampled according to its attributes. (**II**) To explore models on the Pareto frontier, small models are scaled up, while large models undergo hyperparameter tuning. (**III**) An LLM agent then iteratively refines high-performing ideas and candidate models.

where n denotes the number of HW budgets to be optimized. The parent models were chosen based on the top-k strategy except for the Pareto-aware mutation.

Typical evolutionary searches perform crossover and mutation operations to balance exploration and exploitation. Instead of defining crossover as an operation that takes two parent models to generate a child model, we introduce a relaxed definition. We introduce a multi-type mutation to balance exploration and exploitation. Specifically, *mutation with fair idea sampling* ensures a diverse set of ideas is sampled equitably based on their attributes providing fair exploratory opportunities for mutation ideas. Next, *Pareto-aware mutation* improves a wide range of models from the Pareto frontier according to the accuracy and HW budgets, thereby exploring models that are performant relative to their size which often be ignored in the simple top-k sampling strategy. Then, *LLM-driven iterative mutation* speeds up the search by focusing on and repeatedly improving high-performing models and ideas. Each type of mutation is applied sequentially. Furthermore, a feedback loop downsizes models that exceed HW constraints. These strategies allow FairNAD to effectively preserve diverse architectures. The pseudo code of FairNAD is provided in Appendix F.1.

4.3.1 Mutation with Fair Idea Sampling

Diverse ideas from $\mathcal{D}_{\text{kn}}$ are evaluated (Figure 4 (**I**).) A mutation idea, op_{cur}, is sampled from $\mathcal{D}_{\text{kn}}$ based on its design attribute tag. We define "fair idea sampling" as the process of uniformly sampling the idea op_{cur} according to the attributes in $\mathcal{D}_{\text{attr}}$. In addition, the part of the code to be modified is specified according to the attribute tag of op_{cur}. Specifically, when the granularity level is tagged as "operation" or "block and connectivity", the idea op_{cur} is applied either to modify an existing module or to design a novel module, with this choice being uniformly sampled. When the tag is "network", op_{cur} is prompted to ensure that the modification is applied at the network-level. This helps to enhance the diversity and novelty of the generated architectures while also ensuring that op_{cur} is properly applied to the intended part of the model.

The sampling coverage can be formulated as follows. Suppose there are N sub-categories, and sub-category i is sampled with probability p_i. The probability that sub-category i is never sampled after n trials is $(1-p_i)^n$. In contrast to unstructured sampling, our fair sampling strategy makes p_i closer to $1/N$, yielding a coverage probability of $1-(1-1/N)^n$ for each sub-category. The fair sampling strategy improves the coverage of minor categories and reduces the number of samples needed to explore all design categories.

4.3.2 Pareto-Aware Mutation

Simply applying the top-k parent sampling strategy can be inefficient because it tends to overlook compact yet performant models on the Pareto frontier with low FLOPs or parameter sizes, thereby reducing the diversity of candidate models, potentially causing the search to stagnate. Thus, we introduce the Pareto-aware mutation (Figure 4 (**II**)). This mutation employs a straightforward strategy to improve accuracy: model refinement through upscaling or hyperparameter adjustment. First, the k parent models are sampled from the Pareto frontier using the method in NSGA-II [13]. Then, models below a certain budget threshold are upscaled (by increasing their width or depth), whereas models exceeding the threshold undergo hyperparameter tuning. Hyperparameters that are unrelated to width and depth, such as kernel size and stride, are adjusted. Several prompts are predefined for both upscaling and hyperparameter tuning and are uniformly sampled in each trial.

4.3.3 LLM-Driven Iterative Mutation

To accelerate the search process, an iterative refinement is introduced, which mutates high-performing ideas and models recorded in the history (Figure 4 (**III**).) The mutation record h_{t-1} for a'_{par} taken from $\mathcal{D}_{\mathrm{hist}}$ and $\mathcal{D}_{\mathrm{attr}}$ are prompted to the LLM to generate a mutation operation from a similar design attribute category. Specifically, the mutation prompt, p_t, is formatted with a template $\mathcal{T}_{\mathrm{idea}}$ as $p_t = \mathrm{format}(\mathcal{T}_{\mathrm{idea}}; a'_{\mathrm{par}}, h_{t-1}, \mathcal{D}_{\mathrm{attr}})$. This operation is repeated d times.

4.4 Feedback Loop

For each mutation process, the generated code a'_{cur} is verified and modified using a feedback loop with multiple verifiers to generate the final code a_{cur}. As a result, the chance of generating a valid code increases, which in turn improves the diversity of the generated architectures. Each verifier is described as follows:

- Execution verification: a'_{cur} is verified to determine if it is compilable and executable. If it fails, a **debug** operation is performed; the error message and a'_{cur} are queried into an LLM to fix a'_{cur}.
- Budget verification: The HW budgets (e.g., FLOPs and parameter size) are calculated for a'_{cur}. If it exceeds the budgets, **model downscaling** is performed; a'_{cur} and the model downscaling strategy (e.g., depth or width shrinking) are provided to an LLM. Model downscaling strategies are predefined and uniformly sampled for each trial.
- Structural verification: a'_{cur} is verified to ensure that the code has been modified from a'_{par} and retains the multi-layer architecture. If a'_{cur} is unmodified or consists of only a single layer, it is discarded, and the generation process is repeated with a new prompt.

5 Experimental Setup

$\mathcal{D}_{\mathrm{kn}}$, was created by scraping papers from [56], selecting 1,149 from 2,353 abstracts to extract 15,323 ideas. For $\mathcal{D}_{\mathrm{attr}}$, 15 main categories and 55 sub-categories were generated. $\mathcal{D}_{\mathrm{ref}}$ contains 14 reference architectures, including convolution-based (e.g., ResNet [21], MobileNetV2 [42], MobileNetV3 [22], EfficientNet [47], Inception [46], ConvNeXT [31], ResNeXT [52]), Transformer-based (e.g., Vision Transformer [18], Swin Transformer [30], EfficientViT [5]), MLP-based (e.g., MLP-Mixer [48]), and hybrid models (e.g., EfficientFormer [26], LeViT [20], MobileViT [33]).

FairNAD is evaluated on the widely-used NAS-Bench-201 benchmark [17], using the CIFAR-10, CIFAR-100 [25], and ImageNet16-120 datasets [11]. To ensure a fair comparison on NAS-Bench-201, our experimental setup follows that of prior work [56]. Specifically, the search is constrained to the maximum architecture size defined in the NAS-Bench-201 search space. The number of parameters is limited to 1.5M for all datasets, while FLOPs are constrained to 0.2 GFLOPs for CIFAR-10 and CIFAR-100, and 0.05 GFLOPs for ImageNet16-120. Although the base model can be any architecture, the results from a search starting with the CIFAR-10 version of ResNet-32 [21] are reported. Results with other base models are reported in Appendix B.6. The training process adheres to the NAS-Bench-201 settings, including its optimization settings, training epochs, and data splits. The search was conducted on four A100 GPUs with 40GB memory, using two parallel streams per GPU, resulting in eight model parallel evaluations. The experiments are implemented using the

Table 1: Benchmarking on NAS-Bench-201 [17]. The ♢ indicates methods with parameter sharing, ♡ indicates methods without parameter sharing, and ★ indicates methods based on LLMs. The ± 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 [21] (base model)	-	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 [38]	-	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 [29]	-	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 [15]	-	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 [23]	-	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 [54]	-	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 [53]	-	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 [59]	-	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 [16]	-	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 [9]	-	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 [57]	-	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 [36]	-	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 [40]	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 [2]	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 [49]	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 [19]	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 [60]	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 [27]	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 [37]	2000	-	94.26±0.13	-	71.62±1.73	-	45.87±0.96
	★ LeMo-NADe(Gemini) [39]	30	82.94	81.76	52.12	52.96	30.34	31.02
	★ LeMo-NADe(GPT4) [39]	30	90.90	89.41	63.38	67.90	27.05	27.70
	★ NADER [56]	500	91.55	94.62	75.72	76.00	50.20	50.52
	★ FairNAD	100	91.77±0.29	94.93±0.17	76.54±0.28	76.26±0.33	49.99±0.80	49.75±0.61
		500	**92.20±0.14**	**95.46±0.10**	**78.75±0.99**	**78.17±1.25**	**52.73±1.74**	**52.87±1.32**

OpenMMLab library [1]. Qwen3-8B [55] was used as the LLM. More details of the search settings and all prompts can be found in Appendix A and F.2, respectively.

6 Experimental Results

6.1 Comparison with State-of-the-Art Methods

The comparisons of FairNAD with state-of-the-art NAS methods, both hand-crafted and LLM-based methods, on NAS-Bench-201 are presented in Table 1. The results after searching for 100 and 500 new architectures are reported. The results are presented as the mean ± standard deviation over three runs. Code-based NAS, such as GENIUS [60] and LLMatic [37], enhances flexibility by removing predefined search spaces, yet their performance is constrained by the LLM's intrinsic knowledge, limiting significant gains over hand-crafted methods. In contrast, LLM-based NAS that uses paper knowledge, like NADER [56], surpasses the state-of-the-art (SOTA) of hand-crafted NAS by exploring broader design spaces derived from external sources like academic papers.

FairNAD outperforms NADER, the current SOTA, achieving test accuracy improvements of 0.84, 2.17, and 2.35 points on CIFAR-10, CIFAR-100, and ImageNet16-120, respectively. This substantial progress can be attributed to our method's ability to structure the open-ended space into a high-quality, expansive search domain, which is then effectively and efficiently explored by our proposed FairNAD framework. The architectural design of FairNAD provided advantages for incorporating innovative modules and flexible topologies, allowing multi-level modifications to its structure (e.g., operations, connectivity) and components (e.g., convolutions). See Appendix D.2 for details. The contribution of each component is further analyzed in our ablation study.

6.2 Ablation Studies

Effect of structured model design knowledge. FairNAD employs a structured model design knowledge attribute tree, D_{attr}, to extract ideas from papers. This process filters out overly domain-specific ideas and enables the extraction of more fine-grained and high-quality knowledge. This innovation potentially improves the performance of the searched model. To investigate the effect of structured representation, the ablation experiments were conducted using ideas generated by different methods, as shown in Table 2. The results after searching for 200 new architectures are reported. The first mutation stage only is conducted (denoted as "Idea-based mut."). Using paper-based inspiration,

Table 2: Ablation study on CIFAR-100 dataset. The grayed components are our proposal.

Planning	Mutation target	Sampling	Stages	Val / Test
- (Base net)	-	-	-	70.42 / 70.86
LLM's idea		Uniform	Idea-based mut.	72.48 / 72.31
Paper insp.		Uniform	Idea-based mut.	74.78 / 74.55
Structured insp.		Uniform	Idea-based mut.	75.25 / 74.92
Structured insp.	✓	Uniform	Idea-based mut.	75.56 / 75.24
Structured insp.	✓	Fair	Idea-based mut.	77.54 / 77.05
Structured insp.	✓	Fair	Idea-based + Pareto mut.	77.82 / 77.28
Structured insp.	✓	Fair	Idea-based + LLM mut.	75.86 / 75.64
Structured insp.	✓	Fair	Idea-based + Pareto + LLM mut.	**78.32 / 77.97**

which employs ideas from NADER [56], the test accuracy improves by 2.24 points compared to when LLM's ideas are used. This suggests that extracting diverse ideas from papers improves performance. Our proposed structured inspiration yields an additional 0.37-point improvement, demonstrating the effectiveness of using D_{attr} to curate higher-quality knowledge.

Effect of defining mutation target. Tagging each idea with its granularity and main and sub-category attributes using D_{attr} provides fine-grained control over its implementation. Specifically, the "granularity" tag was used to define the mutation target. This approach yields 0.32-point improvement in test accuracy, demonstrating its effectiveness in generating more diverse modules.

Effect of fair idea sampling. FairNAD employs a fairness constraint in idea sampling to facilitate an unbiased and uniform exploration of the open-ended search space. The introduction of the fair sampling mechanism (denoted as "fair" in Table 2) significantly improves test accuracy by 1.81 points, demonstrating the importance of broadly evaluating ideas with diverse attributes. Note that idea sampling is enabled by the proposed structured representation. Thus, the complete use of the structured representation yields a total improvement of 2.50 points over the unstructured knowledge pool. Further ablation experiments are presented in Appendix B.4.

Effect of multi-type mutation. Our proposed multi-type mutation is designed to efficiently explore the open-ended design space. The introduction of the Pareto-aware mutation stage (denoted as "Pareto mut." in Table 2) improves test accuracy by 0.23 points by facilitating the sampling of diverse parent models, which mitigates search stagnation. Further integrating the LLM-driven iterative mutation stage (denoted as "LLM mut.") yields an additional 0.69-point improvement. This stage accelerates the search by effectively exploring promising models and mutation operations. However, introducing the LLM-driven iterative mutation stage alone is detrimental, degrading test accuracy by 1.41 points. This is because LLM-driven iterative mutation by itself cannot generate innovative mutation ideas, making it inefficient when applied in isolation. It becomes effective when combined with the Pareto-aware mutation stage, as the latter introduces diversity in both parent models and mutation methods.

Effect of feedback loop. As shown in Table 3, eliminating (denoted as "w/o") the execution (denoted as "exec."), budget, and structural (denoted as struct.) verifiers degrades the test accuracy by 0.99, 1.34, and 1.38 points, respectively. This suggests that each stage of the feedback loop contributes to generating a valid model and improves the overall performance. Specifically, eliminating the budget verifier significantly degrades the pass rate (PR) which not only impairs performance but also increases the time required to generate valid models. The additional experiments and discussion are provided in Appendix B and E, respectively.

Table 3: Ablation studies of the feedback loop. The execution, budget, and structural verification improve the PR and performance.

Verifier	PR	CIFAR-100 validation	CIFAR-100 test
All	**0.56**	**78.32**	**77.97**
W/o exec.	0.52	77.40	76.98
W/o budget	0.04	77.04	76.63
W/o struct.	- (1.00)	77.14	76.59

7 Conclusions

We introduced a novel approach for effective and efficient exploration of the open-ended design space. First, we proposed a semi-automatic method to structure the model design knowledge attributes to

extract diverse and curated knowledge. Next, we introduced FairNAD, using a multi-type mutation to efficiently explore the vast open-ended design space. First, the model design ideas are uniformly sampled according to their attribute. Second, the Pareto-aware mutation stage explores compact yet performant models. Third, the LLM-driven iterative mutation accelerates the search. Furthermore, the feedback loop improves the architectural diversity by downscaling the oversized models. FairNAD achieves state-of-the-art performance, significantly outperforming existing open-ended NAS methods, and we believe it establishes a crucial foundation for future research in open-ended NAS.

References

[1] OpenMMLab — github.com. `https://github.com/open-mmlab`. [Accessed 27-04-2026].

[2] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[3] H. Cai, L. Zhu, and S. Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[4] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.

[5] H. Cai, J. Li, M. Hu, C. Gan, and S. Han. Efficientvit: Lightweight multi-scale attention for high-resolution dense prediction. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 17302–17313, 2023.

[6] A. Chen, D. Dohan, and D. So. Evoprompting: Language models for code-level neural architecture search. *Advances in neural information processing systems*, 36:7787–7817, 2023.

[7] M. Chen, H. Peng, J. Fu, and H. Ling. Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12270–12280, 2021.

[8] M. Chen, K. Wu, B. Ni, H. Peng, B. Liu, J. Fu, H. Chao, and H. Ling. Searching the search space of vision transformer. *Advances in Neural Information Processing Systems*, 34:8714–8726, 2021.

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

[10] J. Cheng, P. Clark, and K. Richardson. Language modeling by language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[11] P. Chrabaszcz, I. Loshchilov, and F. Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

[12] X. Chu, B. Zhang, and R. Xu. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. In *Proceedings of the IEEE/CVF International Conference on computer vision*, pages 12239–12248, 2021.

[13] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[14] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun. Repvgg: Making vgg-style convnets great again. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13728–13737. IEEE, 2021.

[15] X. Dong and Y. Yang. One-shot neural architecture search via self-evaluated template network. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3681–3690, 2019.

[16] X. Dong and Y. Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1761–1770, 2019.

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

[18] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[19] S. Falkner, A. Klein, and F. Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International conference on machine learning*, pages 1437–1446. PMLR, 2018.

[20] B. Graham, A. El-Nouby, H. Touvron, P. Stock, A. Joulin, H. Jégou, and M. Douze. Levit: a vision transformer in convnet's clothing for faster inference. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12259–12269, 2021.

[21] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[22] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[23] S. Hu, S. Xie, H. Zheng, C. Liu, J. Shi, X. Liu, and D. Lin. Dsnas: Direct neural architecture search without parameter retraining. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12084–12092, 2020.

[24] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Dang, et al. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[25] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. *Master's thesis, University of Toronto*, 2009.

[26] Y. Li, G. Yuan, Y. Wen, J. Hu, G. Evangelidis, S. Tulyakov, Y. Wang, and J. Ren. EfficientFormer: Vision transformers at mobilenet speed. *Advances in neural information processing systems*, 35:12934–12949, 2022.

[27] Z. Li, Z. Lin, and Y. Wang. CoLLM-NAS: Collaborative large language models for efficient knowledge-guided neural architecture search. *arXiv preprint arXiv:2509.26037*, 2025.

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

[29] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

[30] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin Transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[31] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A ConvNet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022.

[32] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. ShuffleNet V2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[33] S. Mehta and M. Rastegari. MobileVit: Light-weight, general-purpose, and mobile-friendly vision transformer. In *International Conference on Learning Representations*, 2022.

[34] K. G. Mills, D. Niu, M. Salameh, W. Qiu, F. X. Han, P. Liu, J. Zhang, W. Lu, and S. Jui. Aio-p: Expanding neural performance predictors beyond image classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 9180–9189, 2023.

[35] K. G. Mills, F. X. Han, M. Salameh, S. Lu, C. Zhou, J. He, F. Sun, and D. Niu. Building optimal neural architectures using interpretable knowledge. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5726–5735, 2024.

[36] S. Movahedi, M. Adabinejad, A. Imani, A. Keshavarz, M. Dehghani, A. Shakery, and B. N. Araabi. λ-darts: Mitigating performance collapse by harmonizing operation selection among cells. In *The Eleventh International Conference on Learning Representations*, 2023.

[37] M. U. Nasir, S. Earle, J. Togelius, S. James, and C. Cleghorn. LLMatic: neural architecture search via large language models and quality diversity optimization. In *proceedings of the Genetic and Evolutionary Computation Conference*, pages 1110–1118, 2024.

[38] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

[39] M. H. Rahman and P. Chakraborty. LeMo-NADe: Multi-parameter neural architecture discovery with llms. *arXiv preprint arXiv:2402.18443*, 2024.

[40] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[41] M. Salameh, K. Mills, N. Hassanpour, F. Han, S. Zhang, W. Lu, S. Jui, C. Zhou, F. Sun, and D. Niu. Autogo: Automated computation graph optimization for neural network evolution. *Advances in Neural Information Processing Systems*, 36:74455–74477, 2023.

[42] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[43] D. So, Q. Le, and C. Liang. The evolved transformer. In *International conference on machine learning*, pages 5877–5886. PMLR, 2019.

[44] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. 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.

[45] M. Suganuma, S. Shirakawa, and T. Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the genetic and evolutionary computation conference*, pages 497–504, 2017.

[46] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[47] M. Tan and Q. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[48] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, et al. MLP-Mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34:24261–24272, 2021.

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

[50] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10734–10742, 2019.

[51] J. Wu, Q. Zhang, and G. Xu. Tiny imagenet challenge. In *Technical report*. Stanford University, 2017.

[52] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[53] S. Xie, H. Zheng, C. Liu, and L. Lin. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019.

[54] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong. PC-DARTS: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020.

[55] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[56] Z. Yang, W. Zeng, S. Jin, C. Qian, P. Luo, and W. Liu. Nader: Neural architecture design via multi-agent collaboration. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 4452–4461, 2025.

[57] P. Ye, B. Li, Y. Li, T. Chen, J. Fan, and W. Ouyang. b-DARTS: Beta-decay regularization for differentiable architecture search. In *proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10874–10883, 2022.

[58] J. Yu, P. Jin, H. Liu, G. Bender, P.-J. Kindermans, M. Tan, T. Huang, X. Song, R. Pang, and Q. Le. BigNAS: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision*, pages 702–717. Springer, 2020.

[59] M. Zhang, S. W. Su, S. Pan, X. Chang, E. M. Abbasnejad, and R. Haffari. iDARTS: Differentiable architecture search with stochastic implicit gradients. In *International Conference on Machine Learning*, pages 12557–12566. PMLR, 2021.

[60] M. Zheng, X. Su, S. You, F. Wang, C. Qian, C. Xu, and S. Albanie. Can gpt-4 perform neural architecture search? *arXiv preprint arXiv:2304.10970*, 2023.

[61] X. Zhou, X. Wu, L. Feng, Z. Lu, and K. C. Tan. Design principle transfer in neural architecture search via large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 23000–23008, 2025.

[62] B. Zoph and Q. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

[63] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

A Experimental Setup Details **16**
A.1 NAS-Bench-201 Settings 16
A.2 Evolution Parameters 16
A.3 Additional Information on the Design Knowledge Collection 16
A.4 Experiment Compute Resources 16

B Additional Experimental Results **17**
B.1 Detailed Ablation Studies of Search Strategy and Knowledge Database 17
B.2 Detailed Analysis of Expert-Dependency of the Knowledge Database Taxonomy . 17
B.3 Sensitivity to Paper Selection 18
B.4 Detailed Ablation Studies of Idea Sampling 18
B.5 Sensitivity to Different LLMs 18
B.6 Performance with Different Base Models 19
B.7 Performance on Tiny Imagenet 19
B.8 Analysis of Wall-Clock Time and LLM Tokens 20

C Analysis of Model Design Attribute Structuring **20**
C.1 Details of Constructed Model Design Attribute Tree 20
C.2 Analysis of Failure Cases 20

D Analysis of FairNAD **22**
D.1 Analysis of Evolutionary Process 22
D.2 Discovered Architecture Details 22
D.3 Analysis of Failure Cases 29

E Discussion, Limitations, and Future Works **30**

F Implementation Details **31**
F.1 Pseudo Code 31
F.2 Prompts 32
F.2.1 Prompt for Model Design Attribute Structuring 32
F.2.2 Prompt for Knowledge Extraction 36
F.2.3 Prompt for Architectural Mutation 38
F.2.4 Prompt for Idea Mutation 40
F.2.5 Prompt for Feedback Loop 41

A Experimental Setup Details

A.1 NAS-Bench-201 Settings

Table 4: NAS-Bench-201 Hyperparameter Settings

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

We followed the experimental setup of NAS-Bench-201 [17] for the data splits (training, validation, and test) and training hyperparameters. The hyperparameters for CIFAR-10 and CIFAR-100 are summarized in Table 4. For ImageNet16-120, we applied the same settings, with the exception of the random crop size and padding, which were set to 16 and 2, respectively, following prior works [29, 56].

A.2 Evolution Parameters

Table 5: Evolution Parameters

Arch. per stage	
Stage I (Mutation with fair sampling):	8
Stage II (Pareto-aware mutation):	8
Stage III (LLM-driven iterative mutation):	4 with 2 iterative steps (d)
Thresholds	
To switch between scale-up and hyperparameter adjustment in stage II:	
$0.9 \times \tau_i \quad \forall i \in \{1, \ldots, n\}$	
Max Retries	
Debug: 2	Model reduction: 4

The hyperparameters for the evolution search are detailed in Table 5. The number of architectures per stage (denoted as "Arch. per stage") is set to eight. For the model refinement stage, the operations switch between scale-up and hyperparameter adjustment at a threshold of 90% of the maximum value for each computation budget (e.g., denoted as τ_i. We used parameter size and FLOPs.) Within the feedback loop, the debug operation (in the **execution** verification phase) is performed up to twice and the model downscaling operation (in the **budget** verification phase) is performed up to four times.

A.3 Additional Information on the Design Knowledge Collection

In this study, we analyzed academic papers published as open access to collect model design knowledge. The paper texts were scraped from the CVPR 2023 website (`https://openaccess.thecvf.com/CVPR2023`).The copyright for each paper belongs to the original author(s), and this research was conducted under Creative Commons licenses (primarily CC BY), ensuring that attribution to the authors was maintained. It should be noted that the full texts of the papers are not redistributed or published in this study; their use is strictly limited to analytical purposes.

A.4 Experiment Compute Resources

Evaluations were performed using four 40GB A100 GPUs. Benchmarking the search for 500 new architectures required 24.22, 26.56, and 30.06 GPU days for the CIFAR-10, CIFAR-100, and ImageNet16-120 datasets, respectively. The full research project required more than the experiments reported in this paper, including:

- Preliminary experiments: For instance, we assessed the feasibility of the proposed stages of FairNAD by utilizing a reduced number of training epochs (e.g., one epoch per candidate architecture) for evaluation, in contrast to the full training schedule (e.g., 200 epochs).

- Exploration of alternative approaches: We also evaluated various parent sampling methods other than top-k sampling. However, we found that applying Pareto sampling to all stages resulted in an inefficient search. Consequently, we concluded that Pareto sampling is best suited exclusively for the Pareto-aware mutation, while other mutations should be performed with top-k sampling for faster search. Moreover, the conventional crossover method, which references two parent models, did not perform effectively with our LLM-based approach.

B Additional Experimental Results

B.1 Detailed Ablation Studies of Search Strategy and Knowledge Database

Table 6: Detailed ablation studies of search strategy and knowledge database

Search strategy	Knowledge DB	CIFAR-100 validation	CIFAR-100 test
Multi-agent	Unstructured	73.58	73.07
Multi-agent	Structured	75.04	74.77
Multi-type mutation	Unstructured	77.14	77.05
Multi-type mutation	Structured	78.32	77.97

To cleanly separate FairNAD's contributions, multi-type mutation and structured knowledge database, we conducted a full 2×2 controlled experiment on CIFAR-100 that disentangles the knowledge representation (unstructured vs. structured) and the search strategy (NADER multi-agent vs. our multi-type mutation) under the same LLM setup. We report results of searching for 200 new architectures in Table 6. The results show that both components contribute to the performance. With the same unstructured knowledge pool, our multi-type mutation outperforms the NADER-style multi-agent search by 3.98 points in test accuracy, 77.05% vs. 73.07%. With the same structured knowledge pool, the improvement is 3.20 points, 77.97% vs. 74.77%. These results suggest that the proposed multi-type mutation is more effective than the NADER-style multi-agent search in our code-based model generation setting. We attribute this to the flexibility of multi-type mutation. The multi-agent search uses a reflector to guide the LLM based on past experience, which may constrain the search direction in a flexible code-based search space and reduce model diversity. In contrast, our method uses search history only in the LLM-driven iterative mutation, while mutation with fair sampling and Pareto-aware mutation can explore candidates without relying on history, leading to more diverse model generation.

The structured knowledge pool is also beneficial for both search strategies. It improves the multi-agent search from 73.07% to 74.77% (1.70 points improvement) and our multi-type mutation from 77.05% to 77.97% (0.92 points improvement) in test accuracy. Overall, the controlled comparison confirms that both the structured knowledge pool and the multi-type mutation independently contribute to the final performance.

B.2 Detailed Analysis of Expert-Dependency of the Knowledge Database Taxonomy

Table 7: Sensitivity to removing one category from each granularity on CIFAR-100

Removed category / granularity	CIFAR-100 validation	CIFAR-100 test
- (All categories)	78.32	77.97
Feature extraction operators / operation-level	77.00	76.65
Residual connections and multi-branch architectures / block and connectivity-level	77.74	78.19
Network structural patterns / network-level	78.12	77.75

To quantify the expert dependency of the taxonomy, we performed a sensitivity analysis by removing one main category at a time. Specifically, the most prominent category from each granularity (e.g., operation, block and connectivity, network) was removed. We report the results on CIFAR-100 within searching for 200 new architectures in Table 7. The results show that FairNAD remains effective under all three variants, although removing any main category slightly degrades performance compared

with the full taxonomy. Specifically, when the feature extraction operator is removed from operation level, test accuracy decreases by 1.32 points. For other granularity levels, test accuracy remains comparable. These results suggest that the taxonomy should include categories corresponding to important architectural components. However, the method appears relatively robust to small differences in fine-grained categorization. Thus, FairNAD is not overly dependent on any single expert-defined category.

B.3 Sensitivity to Paper Selection

Table 8: Sensitivity to the paper selection.

Venue-Year	Search (archs)	Val	Test
CVPR-2025	200	77.68	77.59
ICCV-2025	200	77.58	77.84

Table 8 presentsthe results obtained using papers from different venues and years.. The search performance remained largely consistent across different sources. Our fair idea sampling strategy encourages the search to cover diverse operators in the search space, thereby mitigating potential biases caused by trends specific to a particular venue or year.

B.4 Detailed Ablation Studies of Idea Sampling

Table 9: Detailed ablation studies of idea sampling on CIFAR-100 dataset.

Granularity	Main category	Sub-category	Val	Test
-	-	-	75.56	75.24
✓			76.64	75.96
	✓		77.54	77.05
		✓	73.68	73.06

Table 9 shows the results of considering different attribute levels for fairness in idea sampling. We report the results after searching for 200 architectures. The checkmark (✓) indicates that ideas are sampled uniformly at the corresponding level. A dash (-) indicates that the ideas are uniformly sampled from $\mathcal{D}_{\text{kn}}$ without considering any attributes. The search is performed only with the first stage, the mutation with fair idea sampling stage.

The test accuracy improves by 0.72 and 1.81 points when considering the *granularity* and *main category* attributes, respectively. Since the ideas are prompted by specifying the mutation target (i.e., which part of the code is to be edited), model diversity can be improved by uniformly sampling according to the granularity attribute. Utilizing the main category yields the best performance, as it enhances the diversity of the generated models. However, considering the *sub-category* significantly degrades the performance as the test accuracy drops by 2.18 points. This degradation is likely caused by overly fragmented categories. Specifically, while there were 15 main categories, there were 55 sub-categories. Using sub-categories for sampling creates an excessive number of groups, which undermines the significance of the sampling process. Furthermore, the number of sub-categories per main category is imbalanced. Thus, considering sub-categories introduces additional biases into sampling and degrades the performance.

B.5 Sensitivity to Different LLMs

Table 10 shows the results of FairNAD using different sizes of Qwen2.5-Coder [24] and Qwen3 [55]. We replaced the LLM for search and used Qwen3-8B for extracting model design knowledge. Since FairNAD relies on the coding ability of LLMs, the LLM's capability to implement the intent of the prompts affects search performance. However, the older variant of Qwen, Qwen2.5-Coder-7B-Instruct, which is less performant than GPT-4o [24], achieved 76.25% test accuracy on CIFAR-100 within searching for 200 new architectures. In contrast, the prior approach, NADER [56], reaches 76.00% test accuracy after 500 architecture searches. Similarly, the relatively lightweight variant of Qwen3, Qwen3-4B still produces high-performing models, yielding 76.59% test accuracy for CIFAR-100 within 200 architecture search while the prior approach, NADER [56] reaches 76.40%

Table 10: Sensitivity to the choice of LLMs on CIFAR-100 dataset.

Method	LLM	Search (archs)	Val	Test
NADER [56]	GPT4o	500	75.72	76.00
NADER [56]*	Qwen3-8B	500	75.84	76.40
FairNAD	Qwen2.5-Coder-7B-Instruct	200	76.62	76.25
FairNAD	Qwen2.5-Coder-32B-Instruct	200	78.60	78.83
FairNAD	Qwen3-4B	200	76.62	76.59
FairNAD	Qwen3-8B	200	78.32	77.97
FairNAD	Qwen3-32B	200	78.88	78.27

* We reproduced the official implementation.

test accuracy after 500 architecture searches with Qwen3-8B. This suggests that even if the LLM is not fully capable of implementing each prompt, the Pareto-aware parent sampling strategy and feedback loop provide variety in the searched models, and thus yield high-performing models. The larger variant of Qwen3, Qwen3-32B performs better than the smaller variants, yielding 78.27% test accuracy for CIFAR-100. Also, we observed that searches using Qwen3-32B find high-performing models at an earlier stage of the search than those with smaller variants. This suggests that larger LLMs may be more capable of finding better-performing models and may also accelerates the search.

B.6 Performance with Different Base Models

Table 11: Sensitivity to the base architecture on CIFAR-100 dataset.

Architecture	Search (archs)	Params [M]	GFLOPs	Val	Test
NAS-Bench-201 random	0	1.47	0.17	71.82	72.02
+ FairNAD	200	1.45	0.19	76.62	76.59
MobileNetV2-0.5 [42]	0	0.82	0.01	66.68	66.11
+ FairNAD	200	1.19	0.16	77.02	76.54
ShuffleNetV2-0.5 [32]	0	0.44	0.01	66.84	67.71
+ FairNAD	200	1.50	0.11	76.46	76.07
RepVGG-0.375	0	1.30	0.09	71.18	71.09
+ FairNAD	200	1.39	0.13	76.82	76.71

The results of searching from different base models are shown in Table 11. The search was conducted under the same HW constraints with NAS-Bench-201 [17] with the number of parameters was limited to 1.5M and FLOPs were constrained to 0.2G."NAS-Bench-201 random" denotes a model randomly selected from the NAS-Bench-201 search space. We used the implementation from the NADER [56] GitHub repository. For MobileNetV2 [42] and ShuffleNetV2 [32], the width multiplier was set to 0.5, denoted as "-0.5". For RepVGG [14], the width multiplier was set to 0.375, denoted as "0.375".

FairNAD consistently performs well on different base models, achieving a test accuracy of over 76%, which represents state-of-the-art performance. Furthermore, FairNAD is robust to base models with different modules (e.g., ResNet-based and MobileNet-based) and topologies (e.g., the multi-branch architecture in RepVGG.)

B.7 Performance on Tiny Imagenet

Table 12: Results on Tiny ImageNet

Model	Top-1	Top-5
Base model (MobileNetV2)	53.14	77.41
+ Unstructured insp.	55.57	79.53
+ FairNAD	**58.94**	**80.13**

To further evaluate scalability beyond the NAS-Bench-201 setting, FairNAD was evaluated on Tiny-ImageNet [51] using MobileNetV2 [42] as the base model, with a larger budget of 4.0M

parameters and 0.3 GFLOPs. Each architecture was evaluated for 30 epochs, and the model with the best validation accuracy was trained from scratch for 200 epochs. As shown in Table 12, our method improves over the already optimized MobileNetV2 baseline by 5.80 points in top-1 accuracy, suggesting that the proposed idea sampling and multi-type mutation remain beneficial beyond the NAS-Bench-201-scale setting.

B.8 Analysis of Wall-Clock Time and LLM Tokens

Table 13: LLM generation and evaluation wall-clock time for one architecture (in seconds)

LLM code generation	LLM structural verification	LLM debug (max trials: 2)	LLM model reduction (max trials: 2)	Evaluation
25	10	20	20	1005

Table 14: LLM tokens for generating one architecture

Category	LLM code generation	LLM structural verification	LLM debug (max trials: 2)	LLM model reduction (max trials: 2)
Input	800	1200	2200	1170
Output	4000	1100	2000	1500

We profiled our pipeline on a four-A100 setup to obtain the wall-clock time breakdown. As shown in Table 13, the evaluation stage is the dominant computational bottleneck, accounting for 90% of the total runtime, whereas the LLM generation stages account for the remaining portion at maximum.

From Table 14, to generate 500 architectures, our method uses at most 8,740k input tokens and 12,100k output tokens, compared with NADER's reported 5,371k input tokens and 987k output tokens. It is important to note that Tables 13 and 14 represent an upper bound, assuming that LLM debugging and model reduction are triggered at their maximum number of trials (2 each). In practice, these stages are invoked only when necessary; for example, if debugging and model reduction are each performed once per architecture, the input-token usage is reduced to approximately 5,370k, which is comparable to NADER. The higher output-token usage in our method is mainly attributed to Qwen's thinking mode.

C Analysis of Model Design Attribute Structuring

C.1 Details of Constructed Model Design Attribute Tree

Table 15 and Table 16 show details of the model design attribute tree, $\mathcal{D}_{\mathrm{attr}}$, for performance and efficiency targets, respectively. $\mathcal{D}_{\mathrm{attr}}$ was constructed using the method explained in Section 4.1. After automatically collecting the sub-category items using LLMs, the obvious misclassifications (e.g., a main category item appears as a sub-category) and noise artifacts (e.g., the erroneous inclusion of split keys "*main category*") were corrected. This is a rule-based post-processing step. The main categories were checked to ensure that they appear in the sub-categories and do not contain erroneous split keys.

Thanks to the structured attribute template, $\mathcal{T}_{\mathrm{attr}}$, fine-grained and comprehensive attributes were extracted as sub-categories. As the codes of state-of-the-art reference models (a^{ref}) were analyzed, the attributes include representative modules of modern architectures.

C.2 Analysis of Failure Cases

The major failure cases for model design attribute structuring are summarized as follows:

1. Overly specific attributes: The LLM often fails to follow the instruction to extract general attributes and collects modules existing only in specific models (e.g., "input subtraction pooling".)

Table 15: Details of model design attribute tree for performance.

Granularity	Main category	Sub-category
operation level	feature extraction operators	attention, mlp, token-mixing-mlp, grouped convolution, local convolution, convolution, input subtraction pooling, channel-mixing-mlp
	normalization	grn, batch normalization, layer normalization
	activation	siLU, gelu, relu, relu6, hardswish
block and connectivity level	input encoding	position embedding, cnn stem, patch embedding
	residual connections and multi-branch architectures	residual connections, multi-branch architecture, learnable layer scale
	feature fusion and aggregation	multi-scale feature fusion, element-wise addition, channel concatenation
	adaptive feature recalibration	channel attention (SE block), multi-scale attention, multi-scale feature extraction
network level	network structural patterns	local-global feature extraction, multi-stage hierarchical structure, hierarchical structure, stacked transformer blocks, hierarchical channel scaling, hybrid convolution and transformer architecture
	layer arrangement and order	pre-activation, layer normalization before attention

Table 16: Details of model design attribute tree for efficiency.

Granularity	Main category	Sub-category
operation level	efficient convolution	depthwise convolution, grouped convolution, depthwise separable convolution
	efficient transformer	windowed attention, reduced key dimension attention, lightweight multi-scale linear attention, patch-based transformer
block and connectivity level	efficient block structures	bottleneck block, grouped convolution blocks, pooling-based blocks, convnext block, local-global block structure, inverted residual block, swin block, stochastic depth
	dense connectivity for feature reuse	-
network level	network-wise scaling	hierarchical feature downsampling, compound scaling, channel width scaling
	dynamic computation	stochastic depth

2. Inconsistent categorization: The LLM classifies the same attribute into different categories when analyzing different reference models (e.g., "grouped convolution" appears in multiple categories.)

3. Missing attributes in specific categories: Although specific main categories exist in the manual design, no corresponding attributes exist when analyzing the reference models (e.g., no sub-categories are found for "dense connectivity for feature reuse" in Table 16.)

We attribute failures (1) and (2) primarily to the LLM's capability. Specifically, (1) is caused by the LLM's limitation in following the instructions and (2) occurs due to inconsistencies in the LLMs' output. These errors may be diminished by using more performant, larger LLMs. Failure (3) is caused by the limitations of using reference models. If specific design attributes do not exist in the set of reference models, they cannot be extracted. This issue can be addressed by adding more reference models (e.g., including models designed for different tasks or domains) or different types of reference representations such as texts. Using a hand-crafted NAS search space as a reference is another direction that could potentially be useful for extracting promising design principles.

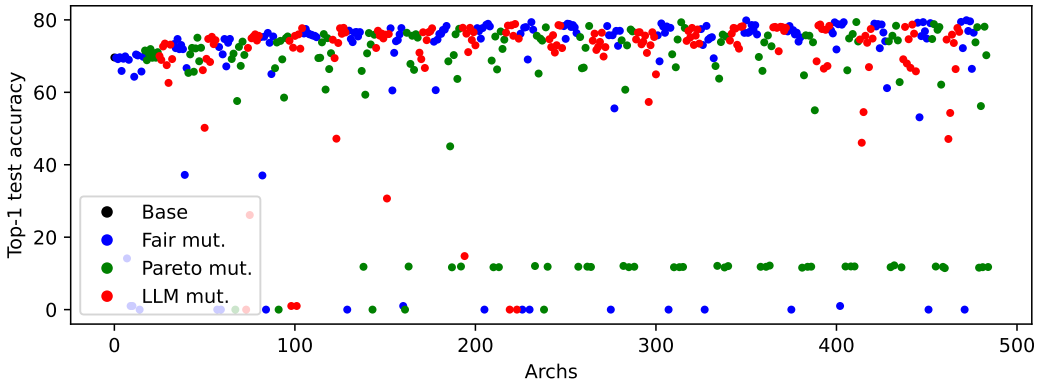

Figure 5: Evolutionary process on CIFAR-100 for searching 500 architectures.

D Analysis of FairNAD

D.1 Analysis of Evolutionary Process

Figure 5 shows the evolutionary process on the CIFAR-100 dataset. Each mutation opertion in FairNAD contributes to improving the performance. The performance gain is significant in the first iteration of the search; performance gains can be observed in mutation with fair idea sampling (denoted as "Fair mut."), Pareto-aware mutation (denoted as "Pareto mut."), and LLM-driven iterative mutation (denoted as "LLM mut.").

In the early stage of search, the performance gain is larger for the LLM-driven iterative mutation than in the other two operations. This suggests that a greater focus on promising models speeds up the search when the models have not been sufficiently optimized. In the later stages of search, performance gains can be observed more often in the mutation with fair idea sampling and Pareto-aware mutation. Mutation with fair idea sampling can introduce novel modules and prevent performance from stagnating. Similarly, the Pareto-aware mutation improves diverse parent models and can potentially lead to improvements even in the later stages of search.

D.2 Discovered Architecture Details

```
import torch
import torch.nn as nn

class DPCA(nn.Module):
    def __init__(self, channel, reduction=16):  # Reduced reduction for more
    expressive attention
        super().__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.max_pool = nn.AdaptiveMaxPool2d(1)
        self.fc = nn.Sequential(
            nn.Linear(channel * 2, channel // reduction, bias=False),
            nn.ReLU(inplace=True),
            nn.Linear(channel // reduction, channel, bias=False),
            nn.Sigmoid()
        )

    def forward(self, x):
        b, c, _, _ = x.size()
        y_avg = self.avg_pool(x).view(b, c)
        y_max = self.max_pool(x).view(b, c)
        y = torch.cat([y_avg, y_max], dim=1)
        y = self.fc(y).view(b, c, 1, 1)
        return x * y

class BaseBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
```

```
        super().__init__()
        self.se_block = DPCA(out_channels)
        self.op1 = nn.Conv2d(in_channels, in_channels, kernel_size=3, stride=stride, padding=1, bias=False, groups=in_channels)
        self.op2 = nn.BatchNorm2d(in_channels)
        self.op3 = nn.PReLU(num_parameters=in_channels)  # Restored PReLU for stable gradient flow
        self.op4 = nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=1, padding=0, bias=False)
        self.op5 = nn.BatchNorm2d(out_channels)
        self.downsample = (in_channels != out_channels) or (stride != 1)
        if self.downsample:
            self.proj = nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, padding=0, bias=False)
            self.bn = nn.BatchNorm2d(out_channels)
        self.op6 = nn.PReLU(num_parameters=out_channels)  # Restored PReLU for stable gradient flow

    def forward(self, x):
        x1 = self.op1(x)
        x1 = self.op2(x1)
        x1 = self.op3(x1)
        x1 = self.op4(x1)
        x1 = self.op5(x1)
        if self.downsample:
            x = self.proj(x)
            x = self.bn(x)
        x1 = x1 + x
        x1 = self.se_block(x1)
        x1 = self.op6(x1)
        return x1

class FeaturePool(nn.Module):
    def __init__(self):
        super().__init__()
        self.stored_features = None

    def forward(self, x):
        B, C, H, W = x.shape
        pooled = x.mean(dim=0)  # (C, H, W)
        self.stored_features = pooled
        return x

class Network(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.stem = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.PReLU(num_parameters=64)
        )
        layers = []
        for _ in range(5):  # Original block count preserved
            layers.append(BaseBlock(64, 64, stride=1))
        self.layers1 = nn.Sequential(*layers)
        self.feature_pool = FeaturePool()  # New module for dynamic feature reuse
        layers = []
        layers.append(BaseBlock(64, 192, stride=2))  # Original channel scaling preserved
        for _ in range(7):  # Original block count preserved
            layers.append(BaseBlock(192, 192))
        self.layers2 = nn.Sequential(*layers)
        layers = []
        layers.append(BaseBlock(192, 384, stride=2))  # Original channel scaling preserved
        for _ in range(5):  # Original block count preserved
            layers.append(BaseBlock(384, 384))
```

```
        self.layers3 = nn.Sequential(*layers)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(384, num_classes)

    def forward(self, x):
        h = self.stem(x)
        h = self.layers1(h)
        h = self.feature_pool(h)  # Store batch-averaged features
        # Retrieve stored features and expand to batch size
        stored_features = self.feature_pool.stored_features
        stored_features = stored_features.unsqueeze(0).expand(x.size(0), -1, -1, -1)
        h = h + stored_features  # Add stored features to current features
        h = self.layers2(h)
        h = self.layers3(h)
        h = self.avg_pool(h)
        h = h.view(h.size(0), -1)
        h = self.fc(h)
        return h
```

Code 1: Discovered Architecture for CIFAR10 (validation accuracy: 92.18%, test accuracy: 95.61%, parameter size: 1.48M, FLOPs: 0.16G)

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

class BandLimitedReLU(nn.Module):
    def __init__(self, c=1.0):
        super().__init__()
        self.c = nn.Parameter(torch.tensor(c), requires_grad=True)

    def forward(self, x):
        return x * (1 + torch.tanh(self.c * x)) / 2

class BinaryPrototypeModule(nn.Module):
    def __init__(self):
        super().__init__()
        self.tanh = nn.Tanh()

    def forward(self, x):
        return self.tanh(x)

class BaseBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        self.depthwise_conv1 = nn.Conv2d(
            in_channels, in_channels, kernel_size=3, stride=stride, padding=1, groups=in_channels, bias=False
        )
        self.pointwise_conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu1 = BandLimitedReLU()  # Replaced with BandLimitedReLU

        self.depthwise_conv2 = nn.Conv2d(
            out_channels, out_channels, kernel_size=3, stride=1, padding=1, groups=out_channels, bias=False
        )
        self.pointwise_conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.relu2 = BandLimitedReLU()  # Replaced with BandLimitedReLU

        # Dual-branch attention module for gradient descent and inertial terms
        self.attention = DualBranchAttention(out_channels)

```

```
        self.downsample = (in_channels != out_channels) or (stride != 1)
        if self.downsample:
            self.proj_conv = nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False)
            self.proj_bn = nn.BatchNorm2d(out_channels)
        self.scale = nn.Parameter(torch.tensor(1.0))  # Learnable layer scaling parameter

    def forward(self, x):
        x1 = self.depthwise_conv1(x)
        x1 = self.pointwise_conv1(x1)
        x1 = self.bn1(x1)
        x1 = self.relu1(x1)

        x1 = self.depthwise_conv2(x1)
        x1 = self.pointwise_conv2(x1)
        x1 = self.bn2(x1)
        x1 = self.relu2(x1)
        x1 = self.attention(x1)

        if self.downsample:
            x = self.proj_conv(x)
            x = self.proj_bn(x)
        x = x * self.scale  # Apply learnable layer scaling to the residual term
        x1 = x1 + x
        return x1

class DualBranchAttention(nn.Module):
    def __init__(self, in_channels):
        super().__init__()
        # ISCA (Inertial Scale-aware Context Aggregation) branch
        self.isca_branch = nn.Sequential(
            nn.GroupNorm(16, in_channels),
            nn.AdaptiveAvgPool2d(1),
            nn.Conv2d(in_channels, in_channels // 16, kernel_size=1),
            BandLimitedReLU(),
            nn.Conv2d(in_channels // 16, in_channels, kernel_size=1),
            nn.Sigmoid()
        )
        # PGCA (Position-Guided Context Aggregation) branch
        self.pgca_branch = nn.Sequential(
            nn.Conv2d(in_channels, 1, kernel_size=1, padding=0),
            nn.Sigmoid()
        )
        # Binary Prototype Module
        self.binary_prototype = BinaryPrototypeModule()

    def forward(self, x):
        isca = self.isca_branch(x)
        pgca = self.pgca_branch(x)
        combined = isca * pgca
        x = x * combined
        x = self.binary_prototype(x)
        return x

class AdaptiveFeatureAttention(nn.Module):
    def __init__(self, in_channels):
        super().__init__()
        self.norm = nn.GroupNorm(num_groups=16, num_channels=in_channels)
        self.channel_attention = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Conv2d(in_channels, in_channels // 16, kernel_size=1),
            BandLimitedReLU(),
            nn.Conv2d(in_channels // 16, in_channels, kernel_size=1),
            nn.Sigmoid()
        )
        self.spatial_attention = nn.Sequential(
```

```
            nn.Conv2d(in_channels, 1, kernel_size=1, padding=0),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.norm(x)
        x = x * self.channel_attention(x)
        x = x * self.spatial_attention(x)
        return x


class Network(nn.Module):
    def __init__(self, num_classes=100):
        super().__init__()
        # Increased stem channels for scale-up
        self.stem = nn.Sequential(
            nn.Conv2d(3, 128, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True)
        )

        # Reintroduced precompute layer with modified normalization
        self.precompute = nn.Sequential(
            nn.Conv2d(128, 128, kernel_size=1, padding=0, bias=False),
            nn.GroupNorm(16, 128),  # Changed from BatchNorm to GroupNorm
            BandLimitedReLU()
        )

        # Enhanced layers with adjusted depth and channel scaling
        layers1 = []
        for _ in range(2):  # Reduced from 3 to 2 blocks in layers1
            layers1.append(BaseBlock(128, 128, stride=1))
        self.layers1 = nn.Sequential(*layers1)
        self.dropout1 = nn.Dropout2d(0.1)  # Added dropout to prevent
    overfitting

        layers2 = []
        layers2.append(BaseBlock(128, 128, stride=2))  # Reduced from 256 to
    128
        for _ in range(3):  # Increased from 2 to 3 additional blocks in
    layers2
            layers2.append(BaseBlock(128, 128))
        self.layers2 = nn.Sequential(*layers2)
        self.dropout2 = nn.Dropout2d(0.15)  # Increased dropout rate for
    deeper layers

        layers3 = []
        layers3.append(BaseBlock(128, 512, stride=2))  # Doubled from 128 to
    512
        for _ in range(1):  # Reduced from 2 to 1 additional blocks in
    layers3
            layers3.append(BaseBlock(512, 512))
        self.layers3 = nn.Sequential(*layers3)
        self.dropout3 = nn.Dropout2d(0.2)  # Increased dropout rate for
    deeper layers

        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, num_classes)

    def forward(self, x):
        h = self.stem(x)
        h = self.precompute(h)
        h = self.layers1(h)
        h = self.dropout1(h)
        h = self.layers2(h)
        h = self.dropout2(h)
        h = self.layers3(h)
        h = self.dropout3(h)
```

```
        h = self.avg_pool(h)
        h = h.view(h.size(0), -1)
        h = self.fc(h)
        return h
```

Code 2: Discovered Architecture for CIFAR100 (validation accuracy: 79.90%, test accuracy: 79.60%, parameter size: 1.32M, FLOPs: 0.20G)

```
import torch
import torch.nn as nn

class PositionalEncoding(nn.Module):
    def __init__(self, d_model):
        super().__init__()
        self.d_model = d_model

    def forward(self, x):
        B, C, H, W = x.shape
        i = torch.arange(H, device=x.device).view(1, 1, H, 1)
        j = torch.arange(W, device=x.device).view(1, 1, 1, W)
        pe = torch.zeros(B, C, H, W, device=x.device)
        for c in range(C):
            freq = 10000 ** (2 * (c // 2) / self.d_model)
            pe[:, c, :, :] = torch.sin((i / freq) + (j / freq))
        return x + pe

class LocalConvAttention(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size=3, stride=1,
    padding=1):
        super().__init__()
        self.depthwise = nn.Conv2d(in_channels, in_channels, kernel_size,
    stride, padding, groups=in_channels, bias=False)
        self.pointwise = nn.Conv2d(in_channels, out_channels, kernel_size=1,
    bias=False)
        self.bn = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.ln = nn.LayerNorm([out_channels])
        self.attention = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Linear(out_channels, 2),
            nn.Sigmoid(),
        )

    def forward(self, x):
        x = self.depthwise(x)
        x = self.pointwise(x)
        x = self.bn(x)
        x = self.relu(x)
        x = x.permute(0, 2, 3, 1)
        x = self.ln(x)
        x = x.permute(0, 3, 1, 2)
        attn = self.attention(x)
        attn = attn.view(attn.size(0), 2, 1, 1)
        mean, _ = attn.chunk(2, dim=1)
        x = x + mean
        return x

class BaseBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        self.local_conv_attention = LocalConvAttention(in_channels,
    out_channels, stride=stride)
        self.op4 = nn.Conv2d(out_channels, out_channels, kernel_size=1,
    stride=1, padding=0, bias=False)
        self.op5 = nn.BatchNorm2d(out_channels)
        self.downsample = (in_channels != out_channels) or (stride != 1)
        if self.downsample:
```

```
            self.proj = nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, padding=0, bias=False)
            self.bn = nn.BatchNorm2d(out_channels)
        self.op6 = nn.ReLU(inplace=True)

    def forward(self, x):
        x1 = self.local_conv_attention(x)
        x_concat = x1
        x_concat = self.op4(x_concat)
        x_concat = self.op5(x_concat)
        if self.downsample:
            x = self.proj(x)
            x = self.bn(x)
        x_concat = x_concat + x
        x_concat = self.op6(x_concat)
        return x_concat

class Network(nn.Module):
    def __init__(self, num_classes=120):
        super().__init__()
        self.stem = nn.Sequential(
            nn.Conv2d(3, 51, kernel_size=3, padding=1, bias=False),
            PositionalEncoding(51),
            nn.BatchNorm2d(51),
            nn.ReLU(inplace=True)
        )
        layers = []
        for _ in range(8):
            layers.append(BaseBlock(51, 51, stride=1))
        self.layers1 = nn.Sequential(*layers)
        layers = []
        layers.append(BaseBlock(51, 102, stride=2))
        for _ in range(15):
            layers.append(BaseBlock(102, 102))
        self.layers2 = nn.Sequential(*layers)
        layers = []
        layers.append(BaseBlock(102, 204, stride=2))
        layers.append(BaseBlock(204, 204, stride=2))  # Added second downsampling block
        self.layers3 = nn.Sequential(*layers)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(204, num_classes)

    def forward(self, x):
        h = self.stem(x)
        h = self.layers1(h)
        h = self.layers2(h)
        h = self.layers3(h)
        h = self.avg_pool(h)
        h = h.view(h.size(0), -1)
        h = self.fc(h)
        return h
```

Code 3: Discovered Architecture for ImageNet16-120 (validation accuracy: 53.93%, test accuracy: 53.67%, parameter size: 0.65M, FLOPs: 0.04G)

In Code 1, Code 2, and Code 3, the codes for the models with the best validation accuracy, searched on CIFAR-10, CIFAR-100, and ImageNet16-120 datasets, respectively, are presented.

The discovered architecture for CIFAR-10 (Code 1) has the following unique characteristics:

- Customized channel attention, `DPCA`: While the Squeeze-and-Excitation (SE) block, a commonly used attention mechanism, identifies important channels using only the channel-wise average values of the feature maps (`AdaptiveAvgPool2d`), the proposed DPCA leverages both average and maximum values (`AdaptiveMaxPool2d`). This combination enables the model to capture both globally and locally important features.

- Dynamic feature sharing mechanism, `FeaturePool`: At an intermediate point in the network (after `layers1`), the feature maps from all images in a batch are averaged, and the resulting mean feature (`stored_features`) is temporarily stored. This mean feature is then added back to the feature map of each image in the batch. This mechanism dynamically shares global features.
- Deeper and wider network: The depth of `layer2` is increased (e.g., from 5 to 7) and the width is increased (e.g., from 16 to 64, 32 to 192, and 64 to 384)).

The discovered architecture for CIFAR-100 (Code 2) has the following unique characteristics:

- Adopting depthwise separable convolution: This is technique, widely used in lightweight models like MobileNet, significantly reduces both parameters and FLOPs while maintaining or even improving the model's representational capacity.
- Innovative attention module, `DualBranchAttention`: By multiplying the outputs of the `isca_branch` (e.g., channel attention) and the `pgca_branch` (e.g., spatial attention), the model can focus on key regions within important channels. Furthermore, the `BinaryPrototypeModule` normalizes the attention-enhanced features using a tanh function, scaling their values to the range $[-1, 1]$. This is expected to stabilize the inputs to subsequent layers and facilitate a smoother training process.
- Customized activation function `BandLimitedReLU`: This function behaves similarly to ReLU, but is smoother around the origin and utilizes a trainable parameter c.
- Stabilizing training by layer-scaling and dropout: Notably, as the network deepens, the dropout rate for `Dropout2d` is gradually increased (e.g., from 0.1 to 0.15 and then to 0.2.)
- Shallow but wide network: Although the depth is decreased (e.g., from 5 to 2, from 5 to 3, from 5 to 1), the width is increased (e.g., from 16 to 128, from 32 to 128, from 64 to 512).

The discovered architecture for ImageNet16-120 (Code 3) has the following unique characteristics:

- Using `PositionalEncoding` with a CNN-based architecture: Positional encoding, which is commonly used in Transformer models but not typically in CNNs, is applied within the stem module.
- Combining multiple techniques in `LocalConvAttention` module: First, Depthwise separable convolution is used to improve computational efficiency. The mean of the attention values is then added to the entire feature map, enabling dynamic feature shifting conditioned on the input. In addition, both `BatchNorm2d` and `LayerNorm` are employed to leverage their complementary properties, thereby improving training stability.
- Non-standard depth and width: The architecture employs non-standard channel sizes (e.g., 51, 102, 204), differing from the common power-of-two design, along with uneven layer depths (e.g., 8, 15, 2), where the `layers2` block is notably deep. FairNAD allows for the flexible customization of channels and layers to balance performance and the computational budget.

In summary, the architectural design of FairNAD provides the following advantages:

- It incorporates innovative modules and functions, allowing for the creation of highly flexible and customizable network topologies.
- It introduces modifications at multiple levels of granularity, including individual operations, connectivity, and the overall network structure.
- The changes are not confined to a single type of module but span diverse components, such as convolution, attention mechanisms, and pooling.
- Beyond individual module design, it offers the flexibility to adjust the overall width and depth of the network.

D.3 Analysis of Failure Cases

A primary limitation of our method lies in the LLM's occasional failure to correctly translate a high-level idea into a functional implementation. We identified five principal failure cases:

1. Incomplete generation: The LLM often truncates the output, failing to generate the complete code for complex architectures.
2. Component hallucination: The model substitutes unknown modules or functions with plausible but non-existent or incorrect alternatives.
3. Shape mismatch: Tensor shape mismatches frequently occur, particularly when integrating heterogeneous modules such as CNNs and Transformers.
4. Model downscaling failure: The initially generated model becomes excessively large, causing the subsequent model downscaling step to fail.
5. Structural verification failure: The LLM incorrectly identifies a valid model as invalid, or an invalid model as valid. Specifically, although the LLM performs well in determining whether the code has been modified, it often fails to determine whether the architecture is multi-layered.

We attribute failures (1) and (2) primarily to the resource constraints of our experimental setup (e.g., GPU memory) and the inherent limitations of the LLM employed. These issues could potentially be mitigated by using hardware with larger memory or more capable LLMs. While our feedback loop can partially address (3) shape mismatches, (4) oversized models, and (5) structurally invalid models, we found that prompt engineering alone has limited effectiveness for these more structural problems.

As a promising direction for future work, we hypothesize that all five failure cases could be more robustly addressed by incorporating a graph-based representation, such as those proposed in NADER [56] and Genesys [10]. The graph-based representation defines the module classes or network structures. For example, Genesys [10] predefines the GPTblock, a meta module implemented in PyTorch. This module can be factorized into a tree structure of sub-modules to be explored for language models. Genesys builds a module library from external sources, and the modules in this library are then searched to serve as sub-modules. By reducing the number of code lines required for implementation, this graph-based approach lowers the chance of (1) incomplete generation. Moreover, using the module library helps prevent (2) component hallucination. Furthermore, the predefined module class potentially reduces the chance of (3) shape mismatch errors, (4) model downscaling failure, and (5) structural verification failure. The limitations of our work are further discussed in the following section.

E Discussion, Limitations, and Future Works

The proposed methodology exhibits broad applicability, allowing for its integration with other NAD approaches. Firstly, our semi-automated knowledge structuring method is designed to be compatible with the idea extraction phase of NAD frameworks, making it particularly useful for sourcing design knowledge from the literature. Secondly, the distinct search stages and the feedback loop within FairNAD can be incorporated into existing NAD methods, offering a promising avenue to enhance their overall performance. Consequently, the proposed method yields positive social impacts by reducing the time and effort required for manual model design by experts and improving model performance by leveraging information from external sources such as papers to design more powerful models.

A potential negative social impact of our method is its high GPU costs for search. Like NAS methods in general, FairNAD generates, trains, and evaluates a vast number of candidate models to find the optimal architecture. This process is a major issue due to its high GPU costs and significant environmental footprint. This challenge is particularly pronounced in FairNAD, as it incrementally modifies and evaluates models, thus requiring a large number of architecture searches. Moreover, FairNAD requires a substantial number of LLM tokens for iterative improvements in the feedback loop. However, we position this cost as an investment to achieve performance levels that are unattainable with conventional methods. FairNAD demonstrates the significant potential of code-based NAS by achieving a substantial accuracy improvement; specifically, it improves test accuracy on CIFAR-100 by a remarkable 6.55% over LLMatic, the state-of-the-art method for model exploration solely through code modification (excluding NADER, which also uses graph representations). We believe that this computational cost can be significantly reduced (e.g., it was observed that using a larger LLM, Qwen3-32B, accelerates the search) in the future through advancements in LLMs and prompt optimization, and we consider this research an important first step toward this new paradigm.

In addition to the failure cases discussed in Section D.3, FairNAD has several limitations. First, there is a limitation on the variety of modules that can be tested. This is because FairNAD is a code-based method, and the implementable modules are constrained by the knowledge of the LLM. For a more detailed analysis, please refer to the failure case analysis in the previous section. Second, the scope of our search was limited to the backbone. In principle, FairNAD can also be applied to search for the neck and head components. However, designing the neck and head is more challenging than that of the backbone, as they are often tailored to specific tasks or domains. The current model design attribute tree is vision-specific, as it is constructed from vision models. Therefore, direct application to domains such as NLP or speech would require reconstructing the attribute tree based on representative models and literature in the target domain. Importantly, however, our method is not tied to the visual domain. The core idea is to build a domain-specific attribute tree from papers and use it to reduce bias in LLM-based inspiration sampling. Thus, the same pipeline can be applied to NLP or speech by rebuilding the attribute tree with appropriate domain-specific corpora.

To address the aforementioned limitations, the directions for future work are as follows:

- Combining FairNAD with graph-based representations: We provide a detailed explanation in Section D.3.
- Reducing GPU costs for search: A coarse-to-fine approach for architectural modification could accelerate the search. For instance, larger parts of the model code could be edited in the earlier stages of the search based on the LLM's knowledge. Subsequently, smaller modifications could be applied to more promising models.
- Exploring open-set extension: The current pipeline relies on predefined categories; therefore, it may miss ideas that do not fit into these categories. Unmatched ideas could be retained in an "unknown" pool and subsequently grouped based on their semantic similarity to identify potential new sub-categories. This would allow the attribute tree to evolve as new architectural paradigms emerge.
- Searching neck and head architectures as well as backbones: To enable the search for more task- or domain-specific architectures, design knowledge can also be extracted from code in addition to papers.
- Fine-tuning the base LLM: Fine-tuning the base LLM with previous search histories and experimental results is a feasible way to transfer knowledge across search spaces and reduce the need to start from scratch. This helps to explore larger search spaces by using the search experiences of smaller search spaces.

F Implementation Details

F.1 Pseudo Code

The mutation process within FairNAD is detailed in Algorithm 1. The pseudo code of FairNAD is shown in Algorithm 2.

Algorithm 1 Mutation with a Feedback Loop

Require: Parent model a_{par}, Idea op_{cur}
Ensure: Generated child model a_{cur} or **null** on failure

$p_{\text{cur}} \leftarrow \text{Format}(a_{\text{par}}, \text{op}_{\text{cur}})$
$a'_{\text{cur}} \leftarrow \text{LLM}(p_{\text{cur}})$
is_compilable $\leftarrow$ **false**
▷ — Execution Verification Phase —
for $i \leftarrow 1 \rightarrow$ max_debug **do**
    is_compilable, error_msg $\leftarrow$ CheckCompilation(a'_{cur})
    if is_compilable **then**
        break
    else
        $p_{\text{debug}} \leftarrow \text{Format}(a_{\text{par}}, \text{error_msg})$
        $a'_{\text{cur}} \leftarrow \text{LLM}(p_{\text{debug}})$
if not is_compilable **then**
    return null ▷ Compilation failed
▷ — Budget Verification Phase —
satisfies_budget $\leftarrow$ **false**
for $i \leftarrow 1 \rightarrow$ max_downscale **do**
    if $\forall i \in \{1, \ldots, n\}, \text{budget}_i(a'_{\text{cur}}) < \tau_i$ **then**
        satisfies_budget $\leftarrow$ **true**
        break
    else
        $p_{\text{downscale}} \leftarrow \text{FormatForDownscale}(a'_{\text{cur}})$
        $a'_{\text{cur}} \leftarrow \text{LLM}(p_{\text{downscale}})$
if not satisfies_budget **then**
    return null ▷ Budget constraints not met
▷ — Structural Verification Phase —
if IsValidStructure(a'_{cur}) **then**
    return a'_{cur} ▷ Success
else
    return null ▷ Invalid structure, retry with a new idea

F.2 Prompts

F.2.1 Prompt for Model Design Attribute Structuring

To structure model design attributes, the attribute template and reference models are fed to an LLM. The attribute template is used as a system prompt, while a reference model is fed to the LLM with the user prompt.

Prompt: Attribute Template

Role: You are a dedicated assistant for generating neural network model architectures' design attributes based on a given visual model. "Model design attribute" means specific types/categories of the key model designs.

Inputs: Each time, the users will provide you with the PyTorch code of the model.

Goals: Your task is to generate a list of initial model design attributes that are relevant to the given model. The generated attributes should be categorized into two lists:

1. Attributes which improves performance:
   {attribute_examples_for_performance_improvements}
2. Attributes which improves efficiency:

Algorithm 2 FairNAD

Require: Base model architecture a^{base};
Model design knowledge database $\mathcal{D}_{\text{kn}}$; Model design attribute tree $\mathcal{D}_{\text{attr}}$;
Number of initial models k_0; Number of parents for each strategy k_1, k_2, k_3;
Number of iterative steps for LLM mutation d;
Constraints for each budget $\tau_1, \ldots, \tau_n$;
Total number of generations G.
Ensure: Optimized architecture a^*.

Initialization:
Initialize history database $\mathcal{D}_{\text{hist}} \leftarrow \emptyset$.
Let $a_0 \leftarrow a^{\text{base}}$.
Sample k_0 ideas $\{\text{op}_1, \ldots, \text{op}_{k_0}\}$ from $\mathcal{D}_{\text{kn}}$ uniformly based on the main category attribute.
for $j = 1$ to k_0 **do**
    $a_j \leftarrow \text{Mutate}(a_0, \text{op}_j)$.
    Train and evaluate a_j.
    $\mathcal{D}_{\text{hist}} \leftarrow \mathcal{D}_{\text{hist}} \cup \{(a_j, \text{acc}(a_j), \text{op}_j, (\text{budget}_1(a_j), \ldots, \text{budget}_n(a_j)))\}$.

Evolutionary Loop:
for generation $g = 1$ to G **do**
    ▷ — I. Mutation with fair idea sampling —
    Select k_1 parent models P_1 from $\mathcal{D}_{\text{hist}}$ by using top-k strategy.
    for each parent $a_{\text{par}} \in P_1$ **do**
        Sample an idea op_{cur} from $\mathcal{D}_{\text{kn}}$ uniformly based on the main category attribute.
        $a_{\text{cur}} \leftarrow \text{Mutate}(a_{\text{par}}, \text{op}_{\text{cur}})$.
        Train and evaluate a_{cur}.
        $\mathcal{D}_{\text{hist}} \leftarrow \mathcal{D}_{\text{hist}} \cup \{(a_{\text{cur}}, \text{acc}(a_{\text{cur}}), \text{op}_{\text{cur}}, (\text{budget}_1(a_{\text{cur}}), \ldots, \text{budget}_n(a_{\text{cur}})))\}$.

    ▷ — II. Pareto-aware mutation —
    Select k_2 parent models P_2 from $\mathcal{D}_{\text{hist}}$ from Pareto frontier by using the sampling method in NSGA-II.
    for each parent $a_{\text{par}} \in P_2$ **do**
        if $\exists i \in \{1, \ldots, n\}$ s.t. $\text{budget}_i(a_{\text{par}}) < 0.9 \cdot \tau_i$ **then**
            $\text{op}_{\text{cur}} \leftarrow \text{ModelUpScalingIdea}(a_{\text{par}})$
        else
            $\text{op}_{\text{cur}} \leftarrow \text{HyperparameterAdjustmentIdea}(a_{\text{par}})$
        $a_{\text{cur}} \leftarrow \text{Mutate}(a_{\text{par}}, \text{op}_{\text{cur}})$.
        Train and evaluate a_{cur}.
        $\mathcal{D}_{\text{hist}} \leftarrow \mathcal{D}_{\text{hist}} \cup \{(a_{\text{cur}}, \text{acc}(a_{\text{cur}}), \text{op}_{\text{cur}}, (\text{budget}_1(a_{\text{cur}}), \ldots, \text{budget}_n(a_{\text{cur}})))\}$.
    ▷ — III. LLM-driven iterative mutation —
    Select k_3 parent models P_3 from $\mathcal{D}_{\text{hist}}$ by using top-k strategy.
    for each parent $a_{\text{par}} \in P_3$ **do**
        for $i = 1$ to d **do** ▷ Iteratively refine the model for d steps
            $x \leftarrow \text{FindEntry}(\mathcal{D}_{\text{hist}}, a_{\text{par}})$.
            $\text{op}_{\text{cur}} \leftarrow \text{IdeaMutation}(a_{\text{par}}, x, \mathcal{D}_{\text{attr}})$.
            $a_{\text{cur}} \leftarrow \text{Mutate}(a_{\text{par}}, \text{op}_{\text{cur}})$.
            Train and evaluate a_{cur}.
            $\mathcal{D}_{\text{hist}} \leftarrow \mathcal{D}_{\text{hist}} \cup \{(a_{\text{cur}}, \text{acc}(a_{\text{cur}}), \text{op}_{\text{cur}}, (\text{budget}_1(a_{\text{cur}}), \ldots, \text{budget}_n(a_{\text{cur}})))\}$
            $a_{\text{par}} \leftarrow a_{\text{cur}}$. ▷ Update the model for the next iteration of refinement

Output:
$a^* \leftarrow$ Select the architecture with the best validation accuracy from $\mathcal{D}_{\text{hist}}$
return a^*.

{attribute_examples_for_efficiency_improvements}

Try to find attributes not in the above list as well.

Constraints:

- Be comprehensive
- Ensure that each attribute is concise, specific, and clearly describes the model's key innovations. For example, "convolution" is valid, but "a visual module" is too vague.
- Avoid duplication in performance and efficiency lists.
- Provide at least one attribute for each category unless it is irrelevant for the given model.
- The attribute must be related to the architecture design of the visual models.
- The attribute does NOT refer to training, optimization, initialization, dataset, pruning/quantization, pre-training or hardwares.
- The attribute is about visual domain. It does NOT refer to other modals such as depth, LiDAR/Radar, or point clouds.

Outputs: Your output is two JSON objects with keys of each attribute category.
Seperate two JSON objects with **Performance** and **Efficiency**.
If there are no items, return an empty list.

Example output
Performance
operation level: {
    feature extraction operators: [convolution, self-attention, mlp],
    normalization:[layer normalization, batch normalization]
}
block and connectivity level: {
    input encoding: [cnn stem],
    residual connections and multi-branch architectures: [multi-branch structure]
}
network level: {
    network structural patterns: [hierarchical structures]
}

Efficiency
operation level: {
    efficient convolution operation: [depthwise separable convolution]
}
block and connectivity level: {
    efficient block structures: [inverted residual block]
}
network level: {
    dynamic computation: [conditional computation]
}

Prompt: Attribute Structuring

We provide a model code of {reference_model_name}.
Considering its innovation, list-up all model attributes.
Here is the code: {reference_model_code}

Attribute Examples for Performance Improvements

Operation level

1. Feature extraction operators:
   Core operations used to extract features from data. For example:
   - Convolution: Improvements such as kernel size design, dilated convolution (expanded receptive field), deformable convolution (spatially adaptive kernels), etc
   - Self-attention: The core mechanism of Transformers. Includes multi-head attention for multi-perspective information extraction and windowed self-attention for localized computation
   - MLP: Methods that perform feature extraction using only MLPs, without convolution or attention (e.g., MLP-Mixer)
2. Normalization:
   Normalization is essential for stabilizing and accelerating training. For example: Batch Normalization, Layer Normalization, Group Normalization, Instance Normalization.
3. Activation:
   Nonlinearity into the network. For example: ReLU, Leaky ReLU, GeLU, Swish (SiLU)

Block and connectivity level

1. Input encoding:
   Methods to encode input data. For example: CNN stem, Patch embedding, Positional encoding
2. Residual connections and multi-branch architectures:
   Structures to enhance the diversity of feature extraction. For example: residual connections (ResNet), multi-branch structures (inception)
3. Feature fusion and aggregation:
   Methods to combine features from different network locations (layers or branches). For example: element-wise addition, concatenation along channels (DenseNet and Inception), multi-scale feature fusion (U-Net, FPN)
4. Adaptive feature recalibration:
   Attention mechanisms that dynamically learn which information is important. For example: channel attention (SE block), spatial attention

Network level

1. Network structural patterns:
   High-level structure of the entire network and how the resolution of feature maps changes throughout the network. For example: hierarchical structure (Swin Transformer), feature pyramid networks (FPN)
2. Layer arrangement and order:
   Order of operations from a more fine-grained perspective, such as within a module. For example: pre-activation (ResNet)

Attribute Examples for Efficiency Improvements

Operation level

1. Efficient convolution operation:

Convolution operation decomposed or replaced with computationally cheaper alternatives. For example: depthwise separable convolution (MobileNet), grouped convolution (ResNeXt), channel shuffle (ShuffleNet)

2. Efficient transformers:
   Proposes efficient self-attention mechanism, which is the main source of computational complexity in Transformers. For example, local attention (Swin Transformer), linear attention

Block and connectivity level

1. Efficient block structures:
   Blocks that balances computational efficiency and expressive ability. For example, Inverted Residual Block (MobileNetV2), Squeeze-and-Excitation (SE) Block
2. Dense connectivity for feature reuse:
   Features from earlier layers are aggressively reused to improve parameter efficiency. For example, Dense Connectivity (DenseNet)

Network level

1. Network-wise scaling:
   Depth and width are scaled in a balnnced manner to improve performance efficiently. For example, Compound Scaling (EfficientNet)
2. Dynamic computation:
   The computational path or amount of computation is dynamically adjusted, reducing average computational cost. For example, Early Exiting, Conditional Computation (Mixture-of-Experts)

F.2.2 Prompt for Knowledge Extraction

First, papers related to visual backbone architectures are filtered according to their title and abstract. Then, design ideas are extracted from each paper.

Prompt: Paper Filtering by Abstract

Background: You need to evaluate a given paper to determine if it can inspire to design the better basic blocks architecture of vision models.

Inputs: paper title, abstract, and model attribute lists structured with level, main category and subcategory. We give two lists for performance and efficiency improvements.

Goals: According to the title and abstract of the paper, analyzing whether it can get inspiration from the paper to design the better basic block architecture of visual models and refers to any item of the given model attribute lists.

Constraints:

- The inspiration must be related to the basic block architecture design of the visual models.
- The idea is about model backbone, not about specific head design.
- The inspiration is NOT about training, optimization, initialization, dataset, hardware, or resolutions.
- The inspiration is useful for 2D image domain. It is NOT specialized to other modals such as depth, LiDAR/Radar, or point clouds.
- The inspiration falls in any category of the model attribute list.

- Even if the paper does not meet the above constraints, keep it if it can provide hints for model design. In this case, provide the most relevant level, main category and subcategory.
- Model attribute lists are structured as:

```
operation level:
    main category 1: [sub-category 1, 2, ...]
    main category 2: [sub-category 1, 2, ...]
    ...
block and connectivity level:
    main category 1: [sub-category 1, 2, ...]
    main category 2: [sub-category 1, 2, ...]
    ...
network level:
    main category 1: [sub-category 1, 2, ...]
    main category 2: [sub-category 1, 2, ...]
    ...
```

- Answer with ##response## in the end.
  - If the paper refers to the model attributes, answer with the appropriate category. Split target, granularity, main category and subcategory by "*" . (e.g., ##response##target*granularity*main category*sub_category)
    For example, ##response##performance*operation level*feature extraction operators*attention
  - If the paper does not refer to the model attributes, answer with no (e.g., ##response##no)

Title: {title}

Abstract: {abstract}

Design attributes for performance improvements:
{attributes_for_performance_improvement} (Table 15)

Design attributes for efficiency improvements:
{attributes_for_efficiency_improvement} (Table 16)

Prompt: Model Design Knowledge Extraction

You are a computer vision research expert. Get inspirations from this paper to list the ideas for designing the basic block architecture of the visual model backbone.
A paper usually contains several sections: abstract, introduction, related work, methods, experiments and conclusion. Please focus on the methods of the paper to respond the inspirations.

Constraints:

- Provide ideas only related to model architecture design for visual model backbone.
- The idea is about model backbone, not about specific head design.
- Do NOT refer to training, optimization, initialization, dataset, or hardware.
- The idea is useful for 2D image domain. It is NOT specialized to other modals such as depth, LiDAR/Radar, or point clouds.
- The idea does NOT mention to use other pre-trained models.

- The inspiration should fall in any category of the model attribute lists. We provide two lists for attributes related to performance and efficiency improvement. The model attribute lists are structured as:

```
operation level:
    main category 1: [sub-category 1, 2, ...]
    main category 2: [sub-category 1, 2, ...]
    ...
block and connectivity level:
    main category 1: [sub-category 1, 2, ...]
    main category 2: [sub-category 1, 2, ...]
    ...
network level:
    main category 1: [sub-category 1, 2, ...]
    main category 2: [sub-category 1, 2, ...]
    ...
```

Output rules:

- Extract multiple ideas from the paper.
- Each idea must be less than 50 words. Keep it simple and exact.
- Do not include special characters (such as '*' or ':') in the output text.
- Answer the ideas with list. For example,
  - idea 1
  - idea 2
  - ...

Design attributes for performance improvements:
{attributes_for_performance_improvement} (Table 15)

Design attributes for efficiency improvements:
{attributes_for_efficiency_improvement} (Table 16)

Content of the paper: {paper}

F.2.3 Prompt for Architectural Mutation

Common Mutation Prompt:

A common mutation prompt template was used for all three types of architectural mutations in FairNAD. Each idea was formatted according to its granularity tag to specify the mutation target.

Prompt: Architectural Mutation

[System Prompt]

Overall Instructions
You are an expert PyTorch assistant. Your goal is to modify the given model to improve its accuracy based on the user's request.

- **Plan First**: Use the Chain-of-Thought technique. Plan the modification first and then start coding.
- Do not change the whole block or model. The changes need to be applied to some parts of blocks (For example, 'stem', 'base block', 'downsample block', 'Network').

- **Respect the Request**: Faithfully implement the user's core idea.
- If the idea already exists, apply it to a different module or propose an improved version.

Technical & output rules

- **PyTorch Only**: Do not use 'tensorflow', 'keras', or 'huggingface transformers'.
- **Preserve Code**: Do not remove existing helper classes/functions.
- **Tensor Shapes**: Ensure shape compatibility. Input is `(B, 3, H, W)`, final output is `(B, num_classes)`. Also, check the shape compatibility for intermediate features when modifications are made.
- **Backbone Only**: Do not add a classifier head.
- **mmengine**: If used, follow the registry pattern (`@MODELS.register_module(force=True)`). Do not assume new `cfg` keys.
- **Avoid import errors**: Do not forget required imports when adding new modules. (For example, `import torch`)
- **Final Output**:
  - Provide the complete, final model code in a single block.
  - Do not include usage examples, training code, or `if __name__ == '__main__':`.

[User Prompt]

{idea} Please modify the following model: {parent_model_code}

Template: Specifying Mutation Target

When the **idea's granularity is tagged as "operation" or "block and connectivity"**, the idea is formatted with the template uniformly sampled from:

- Create and use a new, innovative module based on the idea '{idea}'
- Modify any module (other than `Network`) according to the idea '{insp}'

When the **idea's granularity is tagged as "network"**, the idea is formatted with the template:

- Change the network structure by the idea '{idea}'

Ideas for Pareto-Aware Mutation:

The ideas for Pareto-aware mutation are predefined for model upscaling and hyperparameter adjustment. The ideas are uniformly sampled when applied.

Model Upscaling Ideas

- In Network, change the number of repeats of layers in self.layers1, 2, ...to make the total number of layers larger. Set larger number at the middle of the network
- In Network, change the number of repeats of layers in self.layers1, 2, ...to make the total number of layers larger. Set larger number at the end of the network
- In Network, rewrite channel numbers to larger values while keeping the current ratios
- In Network, rewrite channel numbers to larger values by changing the current ratios

Hyperparameter Adjustment Ideas

- In any block, change kernel size
- In any block, change stride
- In any block, change use_se
- In any block, change activation

F.2.4 Prompt for Idea Mutation

In LLM-driven iterative mutation, the idea was mutated using an LLM with the template, $\mathcal{T}_{\text{idea}}$ as shown in the following. When the accuracy improved in the parent model a_{t-1} from the grand-parent model a_{t-2}, "`Prompt: For Improved Parents`" was used. When the accuracy dropped, "`Prompt: For Degraded Parents`" was used. "Model refinement ideas" refers to the ideas used in the Pareto-aware mutation.

Prompt: Common Template for Idea Mutation

A parent model has been mutated using the idea related to the mutation attribute categories or simple scale-up ideas.

Mutation attribute categories are organized as follows:

```
operation level:
    main category 1: [sub-category 1, 2, ...]
    main category 2: [sub-category 1, 2, ...]
    ...
block and connectivity level:
    main category 1: [sub-category 1, 2, ...]
    main category 2: [sub-category 1, 2, ...]
    ...
network level:
    main category 1: [sub-category 1, 2, ...]
    main category 2: [sub-category 1, 2, ...]
    ...
```

Model refinement ideas are given by a list.

Design attributes:
{attributes_for_performance_improvement}, {attributes_for_efficiency_improvement}

Model refinement ideas: {refinement_ideas}

Prompt: For Improved Parents

{common_template}

Changing from the parent model to the new model with has improved the performance, which shows that the mutation is crucial. Analyze the difference between the parent and mutated models for:

- Is the mutation about any of mutation attribute categories or simple scale-up ideas?
- Why this mutation improves performance?

Based on the analysis, please make the mutated model even better by:

- If the mutation is about any of design attribute categories, come up with ideas that falls in the similar category of mutation attributes.
- If the mutation is about model refinement, try other ideas about model scale-up.
- Provide the complete, final model code in a single block.

Here is the parent model and the mutated model.

Parent model: {parent_model_code}

Mutated model: {current_model_code}

Prompt: For Degraded Parents

{common_template}

Changing from the parent model to the new model with degraded performance. Analyze the difference between the parent and mutated models for:

- Is the mutation about any of design attribute categories or model refinement?
- Why this mutation degraded the performance?

Based on the analysis, please generate a new model based on the **parent model** that address the issue identified by:

- If the mutation is about model refinement, think ideas other than model refinement.
- If the mutation is about any of design attribute categories, come up with ideas with the same category but opposite effects or ideas that falls in the different category.
- The change must NOT be a simple revert.
- Provide the complete, final model code in a single block.

Here is the parent model and the mutated model.

Parent model: {parent_model_code}

Mutated model: {current_model_code}

F.2.5 Prompt for Feedback Loop

For each mutation process, the generated code is verified and modified using a feedback loop. The prompts for debug (in the execution verification stage), model downscaling (in the budget verification stage), and structural verification are shown as follows.

Prompt: Debug

[System Template]

You are an expert AI assistant specializing in debugging PyTorch code. Your task is to fix bugs in a given PyTorch model.

The user will provide you with:

- The PyTorch code of the failed model and its parent model.

- The full error message and trace-back.
- The original prompt that was used to generate the failed model.

Your core task:
Your primary goal is to fix the bug in the provided model code while preserving all intended functionalities described in the original prompt. To achieve this, think step-by-step through the following process:

1. Analyze the Error: Carefully examine the error message and trace-back to pinpoint the exact line and the root cause of the error.
2. Understand the Intent: Read the original prompt to fully understand the intended functionality of the entire model and the specific role of the problematic module.
3. Formulate a Fix: Based on your analysis, devise a correction strategy that addresses the root cause of the error without sacrificing any features.
4. Implement the Correction: Rewrite the code to implement the fix, ensuring the entire model is complete and correct.

IMPORTANT: You must NOT remove, disable, or bypass any modules or code blocks in the target model. For instance, do not remove any lines in the forward method of {base_module_classes} classes.

Error cases:

`Imports` error
- Do not forget importing 'torch'

`CUDA out of memory` error
- The modules that were added to or modified from the parent model may be used repeatedly within the model. Please reduce the number of repeats of these modules**.

** `NoneType` error**
- A `NoneType` error often indicates that a function or module expected to return a tensor is returning None. This is frequently caused by an incomplete forward method or a missing class name for the final model. Ensure the model's implementation is complete and explicitly returns a tensor from the forward method.

`embed_dim must be divisible by num_heads` error
- Most likely to occur in the attention module such that the `embed_dim` must be divisible by num_heads:

  ```
  multihead_attn = nn.MultiheadAttention(embed_dim,
      num_heads)
  ```

  Change `num_heads` value so that `embed_dim // num_heads`

`Size mismatch` error
- Do not change the output dimension size

`Expected more than 1 value per channel when training, got input size torch.Size([B, C, 1, 1])` error
- The resolution of features is too small. Consider making smaller strides, or other methods to increase the resolution of features.

Rules for when LayerNorm is implemented to CNN-based modules

- LayerNorm module **MUST** accept a 4D tensor '(B, C, H, W)' and output a tensor of the **exact same shape**. For example:

```
self.layer_norm = nn.LayerNorm(embed_dim)
x = x.permute(0, 2, 3, 1)  # (N,C,H,W) -> (N,H,W,C)
x = self.layer_norm(x)
x = x.permute(0, 3, 1, 2)  # (N,H,W,C) -> (N,C,H,W)
```

Rules for when Conv1d are implemented with 4D inputs

- Conv1d must accept 4D tensor 4D tensor '(B, C, H, W)' and output a tensor of the **exact same shape**. For example:

```
self.conv1d = nn.Conv1d(in_channels, out_channels,
    kernel_size)
B, C, H, W = x.shape
x = x.reshape(B, C, H * W)
x = self.conv1d(x)
x = x.reshape(B, C, H, W)
```

Rules for when Transformer or attention blocks are implemented to CNN-based modules

- CNN-Transformer Hybrid module **MUST** accept a 4D tensor '(B, C, H, W)' and output a tensor of the **exact same shape**. For example:

```
**Example 1: Using nn.MultiheadAttention**
qkv = nn.Linear(embed_dim, embed_dim * 3)
multihead_attn = nn.MultiheadAttention(embed_dim,
    num_heads)  # embed_dim must be divisible by
    num_heads
# forward
B, C, H, W = x.shape
x = x.reshape(B, C, H*W)
x = x.permute(2, 0, 1)  # (H*W, B, C)
qkv = self.qkv(x)
q, k, v = qkv.chunk(3, dim=2)
x, _ = multihead_attn(q, k, v)
x = x.permute(1, 2, 0)  # (B, C, H*W)
x = x.reshape(B, -1, H, W)  # (B, C, H, W)

**Example 2: With LayerNorm**
# With LayerNorm, use LayerNorm before reshaping back
    to 4D tensor.
layer_norm = nn.LayerNorm(embed_dim)
# forward
x = x.reshape(B, C, H*W)
x = x.permute(2, 0, 1)  # (H*W, B, C)
qkv = self.qkv(x)
q, k, v = qkv.chunk(3, dim=2)
x, _ = multihead_attn(q, k, v)
x = layer_norm(x)  # (H*W, B, C)
x = x.permute(1, 2, 0)  # (B, C, H*W)
x = x.reshape(B, -1, H, W)  # (B, C, H, W)

**Example 3: Defining qkv**
qkv = nn.Linear(embed_dim, embed_dim * 3)
# forward
x = x.reshape(B, C, H*W)
```

```
        x = x.permute(2, 0, 1)  # (H*W, B, C)
        qkv = self.qkv(x)
        q, k, v = qkv.chunk(3, dim=2)
        q = q * self.scale
        attn = (q @ k.transpose(-2, -1)) * self.scale
        attn = attn.softmax(dim=-1)
        x = attn @ v
        x = x.permute(1, 2, 0)  # (B, C, H*W)
        x = x.reshape(B, C, H, W)
```

Output rules:

- Provide the complete, corrected, and runnable model code as the final response.
- Do not add any explanations, comments, or apologies before the code block.
- Ensure all necessary imports are included at the beginning of the script.
- Wrap the final, complete code inside python . Do not wrap any other text or code snippets.

[User Template]

The target model was mutated from the parent model with the prompt {prompt}.
This caused the error with the error message: {error}.
Please fix the error of the target model.

Target model: {sample_model_code}

Parent model: {parent_model_code}

Prompt: Model Downscaling

[System Template] You are an expert coding assistant. Your task is to fix a given PyTorch model to reduce FLOPs and parameter size.

The user will provide you with:

- The PyTorch code of the target model and the parent model.
- The prompt that was used to mutate the target model from the parent model.

{model_downscaling_operation_template}

Output rules:

- Do not return the parent or target model back.
- Provide the complete, corrected, and runnable model code as the final response.
- Do not add any explanations, comments, or apologies before the code block.
- Ensure all necessary imports are included at the beginning of the script.
- Wrap the final, complete code inside python. Do not wrap any other text or code snippets.

[User template]
The target model was mutated from the parent model with the prompt '{prompt}'. Please reduce the FLOPs and parameter size of the model by '{model_dowmscaling_idea}'.

Target model: {sample_model_code}

Parent model: {parent_model_code}

Model downscaling operations were selected based on the HW budgets ($\tau_i, \quad \forall i \in \{1, \ldots, N\}$) of the mutated model. If any of the model's resulting HW costs is larger than $1.5 \times \tau_i$, the newly added module by the idea was restricted to a single use once within the network. Otherwise, depth or width shrinking is performed.

Model Downscaling Operation Template

[Restricting the Newly Added Module to a Single Use] Please reduce the FLOPs and parameter size by using the added module only once.
Here is the example:

```
self.stem = stem(in_channels, out_channels)
layers = []
for _ in range(n):
    layers.append(block(in_channels, out_channels))
...
self.new_layer = AddedBlock(in_channels, out_channels)  #
   move the added module in the end of backbone
self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
self.fc = nn.Linear(out_channels, num_classes)
```

[Depth or Width Shrinking]
Please reduce the FLOPs and parameter size without removing the added/modified module.

Model Downscaling Ideas

[Restricting the Newly Added Module to a Single Use]

- Ensure that the added module is used only once within the model

[Depth or Width Shrinking]

- In Network, change the number of repeats of layers in self.layers1, 2, ... to make the total number of layers slightly smaller. Set larger number at the middle of the network
- In Network, change the number of repeats of layers in self.layers1, 2, ... to make the total number of layers slightly smaller. Set larger number at the end of the network
- In Network, rewrite channel numbers to slightly smaller values while keeping the current ratios
- In Network, rewrite channel numbers to slightly smaller values by changing the current ratios

Prompt: Structural Verification

The target model was mutated from the parent model.
Judge following and answer if the target model satisfies all of them:

- The model architecture has been changed from the parent model. It's not enough for the code to change (including comments, outputs shapes); the built model architecture itself must be different.
- In Network class, the base block (or something equivalent) is repeated more than or equal to two times.

Answer with ##response## in the end. (e.g., **response**yes)

Target model: {sample_model_code}

Parent model: {parent_model_code}

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: In the abstract and introduction, the inefficiency in exploring the open-ended search space is explained. This paper addresses this issue by introducing the semi-automated design knowledge structuring and the efficient search framework, FairNAD.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: This paper discusses the elaborated limitations of the work in the appendix section. Further, the failure case analysis is provided in the appendix as well.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: This paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: This paper fully discloses the information needed to reproduce the main experimental results. The setups for searched model evaluations and evolution search are explained in the experimental setup section and the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The code will not be made publicly available due to the potential for commercial application. However, the elaborated experimental setup is provided in the paper as explained above.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: All the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) are provided in the experimental setup section and the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: This paper reports the standard deviation for the benchmark result following the prior works.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The information on the computer resources including type of compute workers, memory, and time of execution is reported in detail in the experimental setup section and appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The authors have reviewed the NeurIPS Code of Ethics and confirmed that the paper follows its content.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: This paper discusses both potential positive social impacts and negative social impacts in the discussion section in the appendix.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper does not open-source the models nor datasets that have a high risk for misuse or dual-use.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The original papers that produced evaluation dataset are all cited. The library used for the experiments is also cited. For scraped data from a website which is used for model design knowledge collection, the copyright and terms of services are provided in the appendix.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: This paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are used for the architecture search and design knowledge structuring. The usage of LLMs is elaborated in the method section. Also, the prompts are provided in the appendix.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.