RZ-NAS: Enhancing LLM-guided Neural Architecture Search via Reflective Zero-Cost Strategy

Zipeng Ji [1] **Guanghui Zhu** [1] **Chunfeng Yuan** [1] **Yihua Huang** [1]

Abstract

LLM-to-NAS is a promising field at the intersection of Large Language Models (LLMs) and Neural Architecture Search (NAS), as recent research has explored the potential of architecture generation leveraging LLMs on multiple search spaces. However, the existing LLM-to-NAS methods face the challenges of limited search spaces, time-cost search efficiency, and uncompetitive performance across standard NAS benchmarks and multiple downstream tasks. In this work, we propose **R**eflective **Z**ero-cost **NAS** (RZ-NAS) that can search NAS architectures with humanoid reflections and training-free metrics to elicit the power of LLMs. We rethink LLMs' roles in NAS in current work and design a structured, prompt-based to comprehensively understand the search task and architectures from both text and code levels. By integrating LLM reflection modules, we use LLM-generated feedback to provide linguistic guidance within architecture optimization. RZ-NAS enables effective search within both micro and macro search spaces without extensive time cost, achieving SOTA performance across multiple downstream tasks.

1. Introduction

Neural Architecture Search (NAS) has rapidly advanced, transforming neural network design by automating the search for optimal architectures (Zoph & Le, 2017; Liu et al., 2019; Termritthikun et al., 2021; Jiang et al., 2023; Zhu et al., 2022). With increasing computational complexity and demand for high-performance neural networks in tasks such as image recognition (Real et al., 2019) and object detection (Sun et al., 2022), NAS has proven to be a vital and useful tool. Recently, the combination of Large Language Models (LLMs) with NAS represents a cutting-edge development in automated machine learning, seeking to alleviate the difficulties of manual designs and explore novel architectures on diverse NAS tasks. However, most of the existing work remains in the exploratory phase, where LLMs generate neural architectures directly through textual prompts (Zhao et al., 2023; Yu et al., 2023; Wei et al., 2023). This approach suffers from two key drawbacks: (1) Reproducibility: Stochastic LLM responses hinder consistent results. (2) Interpretability: Text-based prompts lack clarity on the design rationale, making optimization and trust difficult. Moreover, LLM-to-NAS methods that focus on generating architecture code (Lehman et al., 2024) can only support tiny search spaces and simplified networks, thus exhibiting poorer performance compared to established NAS algorithms on standard NAS benchmarks (Chen et al., 2023). Furthermore, current LLM-to-NAS algorithms rely on iterative or evolutionary methods, facing high computational costs, since each architecture requires full training for evaluation. To address this, we aim to design a novel LLM-to-NAS algorithm that (1) enhances LLMs' understanding of NAS architectures from both text-level and code-level, (2) addresses the time-cost issue in existing LLM-to-NAS methods, and (3) achieves better performance and scalability for broader search spaces and standard benchmarks.

In this paper, we introduce a novel framework that combines the text- and code-level comprehension capabilities of LLMs with a **R**eflective **Z**ero-Cost evaluation strategy for **NAS** (RZ-NAS)[1]. To integrate the text- and code-level understanding abilities of LLMs, we develop structured prompts to precisely define NAS tasks. These prompts include: a high-level *role*, detailed *instructions*, an *in-context example*, and the key *reflective module*. Moreover, we utilize Zero-Cost proxies instead of training architectures to reduce computational resources and time cost while maintaining competitive performance (Abdelfattah et al., 2021; Sun et al., 2023). The reflective module guides the LLM to reflect on mutation performance and generates targeted suggestions for further iteration improvements.

[1]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China. Correspondence to: Guanghui Zhu <zgh@nju.edu.cn>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

[1]RZ-NAS is available at https://github.com/PasaLab/RZ-NAS.

Algorithm	Search Space		Search Efficiency on CIFAR10 (GPU days)	Support Zero-Cost NAS	LLM's Role in NAS	
	Micro	Macro			Text-Level Understanding	Code-Level Understanding
GPT-NAS (Yu et al., 2023)	✗	✓	1.5	✗	✓	✗
EvoPrompting (Chen et al., 2023)	✓	✓	–	✗	✗	✓
GENIUS (Zhao et al., 2023)	✗	✓	1.0	✗	✓	✗
LLMatic (Nasir et al., 2024)	✓	✗	41	✗	✓	✗
FL-NAS (Qin et al., 2024)	✗	✓	–	✗	✓	✗
RZ-NAS (Ours)	✓	✓	**0.03**	✓	✓	✓

Table 1. Comparison of key capabilities across existing LLM-to-NAS work. *Text-Level Understanding* indicates providing the textual description of NAS to LLMs, whereas *Code-Level Understanding* refers to supplying NAS code to LLMs. ✓ means that the property is present in the algorithm, while ✗ indicates its absence. – denotes that the source code isn't available, or the search cost is not mentioned.

Building on this, we adopt the evolutionary NAS strategy, optimizing the random mutation process guided by the text- and code-level understanding of LLM. In Table 1, we summarize the improvements of our method compared to five existing LLM-to-NAS studies. Through the reflective Zero-Cost evaluation strategy, RZ-NAS can achieve better performance than the original proxies. In addition, it can even outperform traditional NAS methods while maintaining a low search cost. The contributions of this work are summarized as follows:

- We design a reflective Zero-Cost NAS strategy, coupling architecture generation with humanoid reflections and training-free metrics to elicit the power of LLM-to-NAS.
- We rethink the role of LLMs in NAS and design a structured, prompt-based approach by combining the text- and code-level understanding of LLMs for architecture mutation in evolutionary NAS.
- We show that RZ-NAS outperforms the SOTA LLM-to-NAS and Zero-Cost NAS methods across various datasets and search spaces.

2. Related Work

2.1. Zero-Cost Neural Architecture Search

Zero-Cost NAS means "NAS without training", which has emerged as a promising approach that addresses the expensive computation cost in traditional NAS methods (Xiang et al., 2023; Javaheripi et al., 2022). Using proxy metrics with minimal computational overhead, Zero-Cost NAS allows evaluating many architectures without the full training cost. Some existing Zero-Cost proxies are based on the models' gradient, such as Gradnorm (Abdelfattah et al., 2021), Synflow (Tanaka et al., 2020), and SNIP (Lee et al., 2019), others are built on different perspectives to evaluate models' capacity and expressivity. Gradnorm (Abdelfattah et al., 2021) measures the L2 norm of the gradients of network parameters for a random loss, estimating how well the gradients propagate. Synflow (Tanaka et al., 2020) computes the sum of the element-wise product of weights and gradients to reflect the sensitivity of a network. Jacob (Lopes et al., 2021) leverages the Jacobian matrix between the loss and multiple input samples to quantify the capacity. MAE-DET (Sun et al., 2022) leverages the differentiable entropy to evaluate the amount of information the model extracts. The most recent Zero-Cost NAS strategies (Abdelfattah et al., 2021; Gao et al., 2022) use the evolutionary algorithm as the search framework, mimicking the evolutionary process by iteratively selecting, mutation and recombining candidate architectures based on proxy scores. Recently, benchmarks such as NAS-Bench-Suite-Zero (Krishnakumar et al., 2022), have raised concerns about using them as full replacements for accuracy. These insights motivate the careful selection and combination of proxies in practice.

2.2. LLM for Evolutionary Algorithm

The LLM-based evolutionary algorithm is still at a very early age. Pioneering work (Lehman et al., 2024) finds that LLMs designed for code generation can significantly enhance the effectiveness of the mutation process in Genetic Programming (GP). Building on this, recent studies explore LLMs' role in selection within evolutionary algorithms, reducing reliance on black-box functions. (Hao et al., 2024) encodes architectures as natural language via prompt engineering and uses LLMs to predict proxy scores, reducing computational costs. LLaMEA (van Stein & Bäck, 2024) leverages LLM to automatically generate meta-heuristic algorithms based on a given set of criteria and task definitions. (Romera-Paredes et al., 2023) indicates that LLMs can achieve state-of-the-art performance in the cap set problem in finite-dimensional spaces, revealing the potential of LLMs to contribute to mathematical optimization problems.

2.3. LLM for Neural Architecture Search

Recent advancements in large language models have sparked remarkable results in evolutionary algorithms for neural architecture search, presenting a novel avenue for

automating the design of neural networks. GPT-NAS (Yu et al., 2023) utilizes a GPT model to predict components of neural architecture based on an encoding scheme traditionally employed for evolving architectures within evolutionary algorithms. Similarly, GENIUS (Zhao et al., 2023) replaces the architecture suggestion step with GPT-4, prompting it iteratively to propose architectures, evaluate their performance, and provide feedback for suggesting improvements. EvoPrompting (Chen et al., 2023) leverages LLMs to understand the code of crossover and mutation operators within evolutionary algorithms, focusing on small search spaces and simplified architectural constructions. LLMatic (Nasir et al., 2024) designs candidate operation pools and utilizes LLM to update operations in the architecture.

2.4. Self-Reflections of LLMs

The self-reflection mechanism in LLMs refers to the model's ability to review the outputs or reasoning process to identify and correct errors. While the theoretical support for the effectiveness of reflection mechanism remains unclear, in the context of LLMs, reinforcement learning with human feedback (RLHF) enables output refinement through feedback, simulating the process of reflection (Lee et al., 2024). Reflexion (Shinn et al., 2023) proposes a framework to strengthen LLM agents through language feedback. It introduces a "Generate-Evaluate-Reflect" loop, where Actor generates a trajectory, Evaluator computes rewards, and Self-Reflection provides natural language feedback to guide subsequent decisions. ReEvo (Ye et al., 2024) combines evolutionary computation with human-like reflection to generate heuristic algorithms. Moreover, the reflection mechanism greatly improves model performance on tasks such as multiple-choice reasoning (Renze & Guven, 2024).

3. THE PROPOSED METHODOLOGY

In this section, we first discuss the role of LLM in Zero-Cost NAS and formulate a systematic framework that can be applied to multiple Zero-Cost NAS proxies. Then, we integrate LLM into Zero-Cost NAS methods for better and more intelligent search.

3.1. Problem Formulation

Let S and D denote the search space S and the dataset, respectively. Our goal is to find the optimal architecture a^* from the candidate architecture set A in S via computing Zero-Cost score during LLM-guided architecture mutation. The mutated architecture is evaluated via a Zero-Cost proxy object function O. Given task T, search space S, dataset D, and model M, the search process returns the architecture with the maximum Zero-Cost proxy score. The optimization problem can be expressed as follows.

$$a^* = argmax_{\alpha \in A} O(\alpha, T, S, D, M) \tag{1}$$

3.2. Overall Search Framework

As shown in Figure 1, RZ-NAS is an iterative NAS optimization framework that combines LLMs, Zero-Cost proxy evaluation, and the LLM reflection mechanism. It leverages the understanding of LLM on the specified task, search space, model architecture, and metrics to mutate architectures and evaluate them using Zero-Cost proxies, refining the search for optimized designs through reflective feedback.

In Figure 1, we precisely extract the roles of LLM, which includes two functions: *a mutation generator* for selected architecture and *a reflection module* for yielding better mutations. To help the LLM understand the mutation process, we provide the genotype of the selected architecture in string format and three system-related descriptions with corresponding textual and code representations. The search space description introduces each operation in the defined space. The network construction description and the Zero-Cost proxy description explain how the network is built and how the proxy is computed, respectively, each using code and a brief textual explanation. The above inputs are integrated into a prompt template and fed into the LLM to guide the mutation process, as detailed in Figure 2. Specifically, the architecture textual genotype is incorporated into the user prompt to request new mutations, while the three system-related descriptions are embedded in the system prompt, enabling the LLM to comprehensively understand the task context. During each iteration, the three system-related descriptions remain fixed, providing consistent context for the specific NAS task, while the architecture genotype is updated based on the mutations generated by the LLM.

After the LLM completes the mutation process, the muated architecture is first validated and then evaluated using predefined Zero-Cost proxies. Then, the LLM reflects on the mutated operation and its performance through the reflection module. The reflection module compares the architecture before and after the mutation and prompts the LLM to generate better solutions based on the current mutation.

Population Construction. RZ-NAS maintains a dynamic architecture population throughout the optimization process. The initial population is randomly generated and the Zero-Cost score is calculated for each architecture. RZ-NAS supports both macro and micro search space. For the micro search space, the genotype format follows the approach in DARTS (Liu et al., 2019), where operations are organized within a cell as a directed acyclic graph (DAG) and stacked to form a supernet. In the macro search space, the operations are connected sequentially and the architecture consists of vanilla convolutional layers, where each layer includes a

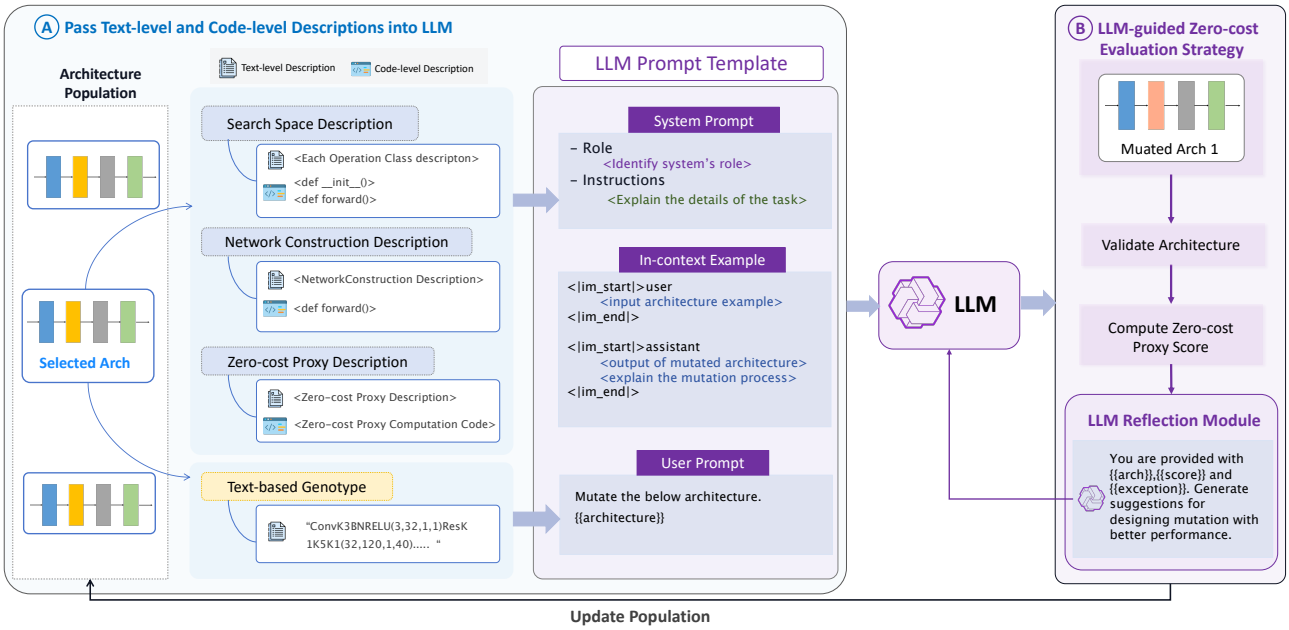


Figure 1. The pipeline of RZ-NAS. The workflow starts by initializing a population of architectures. One architecture is randomly selected, and its text-based description along with system information including search space, network construction, and selected Zero-Cost proxy, is sent to the LLM prompt template. The LLM generates a mutated architecture, which is then validated and scored using the proxy. The mutation, along with its score and possible errors, is fed back into the LLM to guide better mutations in subsequent iterations.

convolutional operator followed by ReLU activation (Serra et al., 2018; Hanin & Rolnick, 2019). Once the population is initialized, we transform each architecture into string format as shown in the text-based genotype in Figure 1.

Architecture Mutation. RZ-NAS understands both text- and code-level descriptions to guide the LLM in mutating the architecture genotype during each iteration. Rather than directly mutating architecture code, the LLM performs text-based mutation, which is more feasible since code-level modifications may span multiple files and classes. The LLM's first role is to guide the mutation of a selected architecture by modifying operators in formatted textual genotype. These mutated operators are defined in the search space and replace the original ones in the architecture genotype. As a result, the output of the mutation process is a textual genotype, which is then transformed into executable architecture code for performance evaluation. The code and semantic descriptions used in the four input components will be detailed in Section 3.3.

Mutation in NAS typically relies on random perturbations to explore the search space. However, this often requires extensive customization and domain-specific knowledge to ensure meaningful and effective changes (Lehman et al., 2024). LLMs with implicit knowledge of architecture design can guide architecture mutation more intelligently. Recent research in evolutionary computation demonstrates that LLMs can outperform traditional mutation in genetic programming tasks (Liu et al., 2023; van Stein & Bäck, 2024), showcasing their ability to embed domain knowledge and intelligently explore the search space. Unlike prior studies (Zhao et al., 2023) that treat LLM as a black box by replacing the entire NAS strategy with simplistic prompts, our approach leverages LLM more strategically within the mutation process.

Architecture Validation. After LLM generates the mutation, RZ-NAS validates its construction for correctness (e.g., checking if the layer length is within the allowed limits). It also adheres to the FLOPs budgets of Zero-Cost NAS to search for optimal networks within various budgets. The invalid architecture will not be added to the population.

Compute Zero-Cost Proxies. RZ-NAS provides a versatile Zero-Cost evaluation strategy that supports different Zero-Cost proxies. As discussed in Section 2.1, Zero-Cost proxies initially originate from gradient-based methods such as Grasp (Wang et al., 2020) and later evolve into more diverse proxies grounded in different theoretical frameworks, such as Zen_Score (Lin et al., 2021). As a general search framework, RZ-NAS can incorporate different Zero-Cost proxies. The Zero-Cost proxy computation module in Figure 1 provides a code block and proxy description for model understanding. The user prompt includes a JSON list with the current architecture, proxy type, and pre-mutation score. The LLM is guided by structured prompts, I/O formats, and in-context examples. Reflections from prior outputs further

Algorithm 1 LLM-guided Mutation and Zero-Cost Evaluation Strategy

INPUT: Search space S, inference budget B, maximal depth L, total number of iterations T, evolutionary population size N, initial structure F_0, the selected Zero-Cost proxy zc
OUTPUT: The architecture A with highest proxy score.

Initialize population $P = \{F_0\}$.
for $t = 1, 2, \ldots, T$ **do**
  Randomly select $F_t \in P$.
  $\hat{F}_t \leftarrow$ GENERATINGMUTATION(F_t, S).
  if VALIDATE$(\hat{F}_t)$ is True **then**
    if $\hat{F}_t$ meets the inference budget B **then**
      Compute Zero-Cost score $z = zc(\hat{F}_t)$.
      Append $\hat{F}_t$ to P.
      if $|P| > E$ **then**
        Remove the architecture with the smallest Zero-Cost score from P.
      end if
    end if
  end if
  Get exception e if F_t is invalid.
  REFLECTION$(\hat{F}_t, z, e)$.
end for
Return F^*, the architecture with the highest score in P.

refine the process. To avoid hallucinations, Zero-Cost scores are computed by code, not LLM.

LLM Reflection Module. We introduce an innovative LLM reflection module that evaluates each mutated architecture and its computed score, providing suggestions and insights for improved designs in the next iteration. The reflection mechanism includes two modules: *internal reflection module in the system prompt* (e.g., Reflect to generate better mutation) and *external LLM reflection module* (e.g., Generate suggestion for better mutation). The external module takes as input the architecture before and after mutation in the current iteration, the Zero-Cost proxy score, and any encountered exceptions as input, and outputs dynamic, structured reflection suggestions to guide future mutations. These two modules form a closed-loop feedback system: the external module provides explicit optimization directions, while the internal module enables implicit semantic reasoning. Previous research on self-reflection in LLMs (Renze & Guven, 2024; Ye et al., 2024) has demonstrated significant improvements in performance, mitigated hallucination, and smoother fitness landscapes by employing self-reflection on problem-solving and heuristic tasks. A detailed example of reflection is shown in Appendix A.2.

Population Update. All architectures including both the original and the mutated candidates are first sorted by their Zero-Cost scores. The architecture with the lowest score is then removed to maintain a fixed population size. After T iterations, the architecture with the highest score is returned. More details on the search workflow can be found in Algorithm 1.

3.3. Prompt Template Design

To achieve LLM-guided architecture mutation, we design prompts with structured context and a specified output format. Figure 2 shows the system-level and task-specific prompt template inspired by Copilot (Denny et al., 2023) and multiple templates from OpenAI Library [2]. The left side outlines the full template structure, while the right displays four input text-level and code-level descriptions. The prompt template includes three components: system prompts defining LLM's role and task descriptions, illustrative in-context examples, and specific user instructions.

- **System Prompt.** The instruction emphasizes the expert role of LLM in Zero-Cost NAS and Python programming. The role definition should be clear and concise to generalize across domains without relying on specific scenarios (Zhang et al., 2024), guiding the LLM to produce scientifically relevant and meaningful architectures. Next, we provide a concise description of the search space. Language models help us understand the structure of each operation by detailing textual properties and construction code. For each operation in the search space, the definition, parameters, and *forward()* function are explained in detail in Appendix A.3. Furthermore, to enhance expertise in optimizing mutation, RZ-NAS introduces the system reflection module, which reflects and gives hints for the improved mutation design in the next step.
  During the network construction process, we define the input/output formats and mutation constraints, such as kernel size and input channels, through textual descriptions. In addition, in the code block, we illustrate the architecture construction process. In Figure 2, we provide the input code where the architecture is formatted as an operation string and instantiated within the function *create_arch_list_from_str()* to build the architecture within the *Network* class.
  Figure 2 also shows the architecture mutation logic. Multiple Zero-Cost proxy computation codes are integrated to ensure system compatibility with various evaluation methods. Based on the type parameter in the user prompt, the system selects the appropriate Zero-Cost proxy.
- **In-context example.** The in-context example provides task-specific instructions, including a mutation example and an explanation of the reasoning steps that lead to the output. Moreover, we integrate the step-to-step reflection

[2] https://platform.openai.com/docs/examples

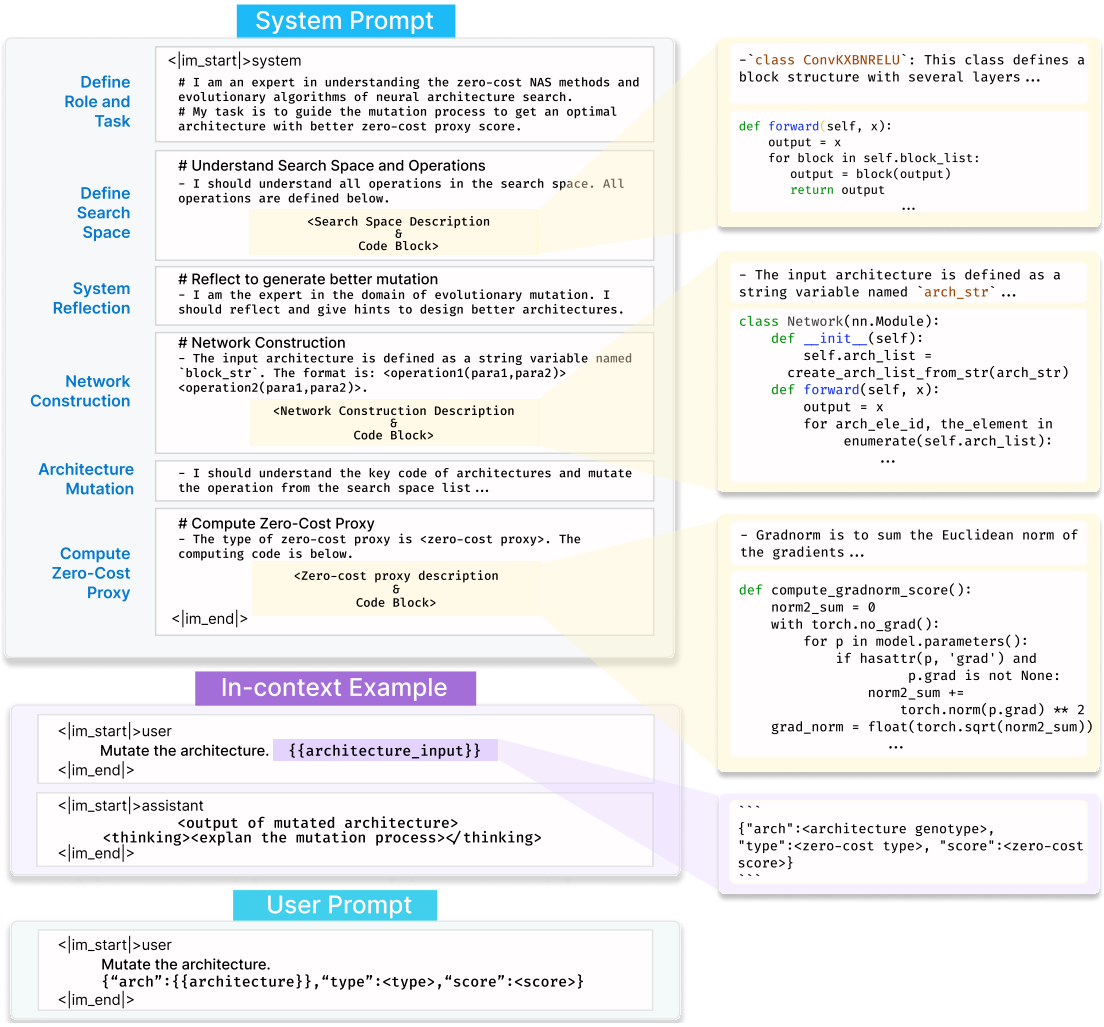


Figure 2. Overview of architecture mutation prompt. Left: the structure of the whole prompt template. Right: four text-level and code-level input descriptions.

mechanism which is highlighted with XML-like tags to guide LLM in reasoning. The example can be tailored for different downstream tasks. Previous research (Min et al., 2022; Liu et al., 2024) has done extensive experiments to show the effectiveness of contextual learning. The in-context examples store successful cases as key-value pairs, which help the induction heads of the transformer to identify successful architectural patterns stored in the context examples (Olsson et al., 2022). Our extensive experimental results show that the example code is critical for LLM reasoning, leading to more consistent and correct architectures without syntax exceptions and ensuring good performance. To avoid the issue of prompt leakage, we add restrictions in the system prompt to prevent prompt leakage. Because there could be cases where in-context examples are output as mutation results during our experiments. To address this, we include "do not disclose examples" in the system prompt. The Zero-Cost NAS method in the in-context example should align with the method selected by the user in the following prompt.

- **User Prompt.** This part defines the format and content of user input. The user input is the architecture to be mutated, which is randomly selected from the population. The random selection stragegy aligns with existing Zero-Cost methods such as Zen-NAS (Lin et al., 2021). The input is provided in JSON format, containing three keys: "arch", "type", and "score". Users must specify the Zero-Cost method to be applied. The "score" field is initialized with the computed Zero-Cost score of the selected architecture.

The whole prompt design is presented in Appendix A.3. After LLM-guided mutation, the mutated architecture is subjected to a syntax correctness check and verified for construction errors to prevent potential exceptions. If an exception occurs, the process skips invalid architecture, and the exception message is sent to the LLM reflection module

Table 2. The test and validation accuracy on NAS-Bench-201. Top: traditional differentiable NAS strategies. Middle: previous LLM-to-NAS methods. Bottom: comparison of each Zero-Cost method's performance against our improved results. Each method is run 3 times. The best results between the LLM-based and Zero-Cost methods are shown in **bold**.

Methods	CIFAR-10		CIFAR-100		ImageNet-16-120	
	valid	test	valid	test	valid	test
DARTS(2nd) (Liu et al., 2019)	39.77 ± 0.00	54.30 ± 0.00	15.03 ± 0.00	15.61 ± 0.00	16.43 ± 0.00	16.32 ± 0.00
SNAS (Xie et al., 2019)	91.10 ± 1.04	92.77 ± 0.83	69.69 ± 2.39	69.34 ± 1.98	42.84 ± 1.79	43.16 ± 2.64
PC-DARTS (Xu et al., 2020)	89.96 ± 0.15	93.41 ± 0.30	67.12 ± 0.39	67.48 ± 0.89	40.83 ± 0.08	41.31 ± 0.22
RLNAS (Zhang et al., 2021)	89.94 ± 0.00	93.35 ± 0.00	70.98 ± 0.00	70.71 ± 0.00	46.86 ± 0.00	43.70 ± 0.00
DrNAS (Chen et al., 2021)	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-NAS (Ye et al., 2022)	91.55 ± 0.00	94.36 ± 0.00	73.49 ± 0.00	73.51 ± 0.00	46.37 ± 0.00	46.34 ± 0.00
GENIUS	91.07 ± 0.20	93.79 ± 0.09	70.96 ± 0.33	70.91 ± 0.72	45.29 ± 0.81	44.96 ± 1.02
LLMatic	91.42 ± 0.13	$\mathbf{94.26 \pm 0.10}$	71.41 ± 1.44	71.62 ± 1.73	44.98 ± 0.87	45.87 ± 0.96
GraSP	89.69 ± 1.39	89.34 ± 2.16	61.11 ± 4.17	60.89 ± 3.88	24.17 ± 10.38	22.99 ± 11.01
Ours(GraSP)	90.11 ± 1.00	92.79 ± 0.80	69.69 ± 2.39	69.34 ± 1.98	42.84 ± 1.79	43.16 ± 2.64
Gradnorm	89.69 ± 1.39	89.27 ± 2.10	61.11 ± 4.17	61.06 ± 4.02	24.17 ± 10.38	24.10 ± 11.36
Ours(Gradnorm)	90.76 ± 1.77	93.99 ± 3.44	71.03 ± 2.34	71.61 ± 2.20	42.43 ± 1.40	42.61 ± 1.38
Synflow	91.03 ± 0.44	93.48 ± 0.00	71.36 ± 1.50	71.35 ± 1.51	44.87 ± 0.00	45.12 ± 0.00
Ours(Synflow)	91.45 ± 0.42	93.47 ± 0.74	73.23 ± 0.14	73.21 ± 0.12	46.37 ± 0.11	46.16 ± 0.30
Zen-NAS	90.20 ± 0.00	93.76 ± 0.00	70.21 ± 0.71	70.67 ± 0.62	40.78 ± 0.00	41.44 ± 0.00
Ours(Zen-NAS)	91.03 ± 0.44	93.48 ± 0.00	71.36 ± 1.51	71.35 ± 1.51	44.87 ± 0.00	45.12 ± 0.00
ZiCo	89.94 ± 0.00	93.35 ± 0.00	70.98 ± 0.00	70.71 ± 0.00	46.39 ± 0.03	46.18 ± 0.04
Ours(ZiCo)	$\mathbf{91.45 \pm 0.10}$	94.24 ± 0.12	$\mathbf{73.35 \pm 0.14}$	$\mathbf{73.30 \pm 0.21}$	$\mathbf{46.53 \pm 0.24}$	$\mathbf{46.24 \pm 0.23}$

for the error analysis, enabling the LLM to generate valid architectures in the next iterations.

4. Experiment

We evaluate RZ-NAS on multiple widely adopted Zero-Cost NAS proxies for different downstream tasks. The details of Zero-Cost proxies are provided in Appendix A.1. We define the mutation space based on the set of architecture operators specified in the system prompt. The search procedure runs for 1500 evolutionary iterations. The population size is set to 100 for NAS-Bench-201 and CIFAR-10, and 256 for CIFAR-100, ImageNet, and COCO. All populations are initialized from scratch using random sampling. The search spaces differ by task: for NAS-Bench-201, CIFAR-10, and CIFAR-100, we use the micro cell-based search space; for ImageNet, we adopt the MobileNet macro search space, consistent with prior works like Zico and Zen-NAS. For COCO object detection, we stack operators to build the backbone following the same configuration used in MAE-DET (Sun et al., 2022). In all experiments, we use the GPT4o model to generate mutations. We also perform an ablation study with different LLMs in Appendix A.4.2. We sample the temperature of the model from [0.2, 0.4, 0.6, 0.8, 1.0] to encourage output diversity. The other settings are identical to different Zero-Cost proxies. In RZ-NAS, the number of input tokens and output tokens is in the range of 2300-2600 and 150-200, respectively. We perform 1500 iterations for one proxy in one search space per proxy. Therefore, the total cost per proxy is around $75.

Table 3. The correlation coefficients between various Zero-Cost proxies and test accuracy on NAS-Bench-201 (KT and SPR represent Kendall's τ and Spearman's ϕ, respectively). The best results are shown in **bold**.

Method	NAS-Bench-201					
	CIFAR-10		CIFAR-100		ImageNet	
	KT	SPR	KT	SPR	KT	SPR
GraSP	0.37	0.54	0.36	0.51	0.40	0.56
Ours(GraSP)	0.43	0.57	0.41	0.55	0.44	0.59
Synflow	0.54	0.73	0.57	0.76	0.56	0.75
Ours(Synflow)	0.56	0.73	0.60	0.79	0.58	0.78
Zen-Score	0.29	0.38	0.28	0.36	0.29	0.40
Ours(Zen-Score)	0.37	0.54	0.36	0.51	0.40	0.56
ZiCo	0.61	0.80	0.61	0.81	0.60	0.79
Ours(ZiCo)	**0.63**	**0.82**	**0.63**	**0.84**	**0.64**	**0.81**

4.1. Performance Comparison

4.1.1. NAS-Bench-201 Search Space

We compare the accuracy performance on the NAS-Bench-201 Benchmark as shown in Table 2. We train the network with the best score for each proxy for three runs under its training setting. RZ-NAS can significantly improve the performance of all Zero-Cost proxies. We further compare the performance with existing LLM-to-NAS methods available in published results or open-source implementations. Table 2 demonstrates the superior performance of RZ-NAS. We also compute Kendall's τ and Spearman's ϕ correlation coefficients between the Zero-Cost proxy and test accuracy to evaluate whether these proxies accurately rank architectures in Table 3. Our method improves the correlation scores of all Zero-Cost proxies across the three datasets, indicating that our method can help Zero-Cost proxies have a greater

Table 4. Performance comparison on CIFAR-10 and CIFAR-100 on the DARTS search space. For the "Method" column, "MS" represents multi-shot NAS; "OS" is short for one-shot NAS; "ZC" is short for Zero-Cost NAS. The best results are shown in **bold**.

Method	Test Error [%]		Method	Cost(GPU days)
	CIFAR-10	CIFAR-100		
PNAS (Liu et al., 2018)	3.41 ± 0.09	19.53	MS	225
ENAS (Pham et al., 2018)	2.89	19.43	RL	0.5
DARTS(2nd) (Liu et al., 2019)	2.76 ± 0.09	20.58 ± 0.44	OS	1
SNAS (Xie et al., 2019)	2.39	-	OS	1.5
P-DARTS (Liu et al., 2018)	2.5	17.46	OS	0.3
R-DARTS (Chen et al., 2020)	2.95 ± 0.21	18.24	OS	1.6
DARTS+PT (Wang et al., 2021)	2.61 ± 0.08	19.05	OS	0.8
β-DARTS (Ye et al., 2022)	2.53 ± 0.08	17.43	OS	0.4
OLES (Jiang et al., 2023)	$\mathbf{2.41 \pm 0.11}$	**17.33**	OS	0.4
GraSP	6.0 ± 0.13	27.8 ± 0.17	ZC	0.01
Ours(GraSP)	5.4 ± 0.07	26.8 ± 0.12	ZC	0.02
Gradnorm	5.7 ± 0.05	25.7 ± 0.08	ZC	0.01
Ours(Gradnorm)	5.2 ± 0.12	24.6 ± 0.16	ZC	0.01
Synflow	5.9 ± 0.11	24.1 ± 0.12	ZC	0.02
Ours(Synflow)	4.3 ± 0.07	21.3 ± 0.20	ZC	0.02
Zen-NAS	2.55 ± 0.04	19.9 ± 0.11	ZC	0.01
Ours(Zen-NAS)	2.50 ± 0.17	17.77 ± 0.09	ZC	0.01
ZiCo	2.45 ± 0.11	17.78 ± 0.13	ZC	0.03
Ours(ZiCo)	$\mathbf{2.41 \pm 0.13}$	17.49 ± 0.08	ZC	0.03

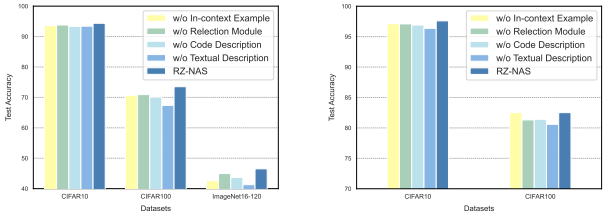
ability to select better candidates.

4.1.2. DARTS Search Space

We also evaluate RZ-NAS on CIFAR-10 and CIFAR-100 within the DARTS search space, two of the most widely adopted datasets in the NAS research, which were not previously applied in existing LLM-to-NAS studies. As shown in Table 4, we compare RZ-NAS with traditional image classification models, NAS methods, and previous Zero-Cost methods. We adhere to the search space of each Zero-Cost method without introducing additional operations. Following the computation of each Zero-Cost proxy, our method significantly outperforms previous Zero-Cost methods and demonstrates competitive results compared to SOTA NAS methods. Among all Zero-Cost proxies, ZiCo and Zen-NAS emerge as the top-performing runners-up, followed by Synflow.

4.1.3. MobileNet Search Space

We further adopt a widely used MobileNetV2-based search space, where architectures are constructed by stacking multiple Inverted Bottleneck Blocks with SE modules (Howard et al., 2017; Sandler et al., 2018). RZ-NAS is applied to search for architectures under varying FLOPs budgets (450M, 600M, and 1000M). As shown in Table 5, RZ-NAS using ZiCo demonstrates significant improvements over all prior NAS methods, solidifying its effectiveness in the search for low-cost, high-accuracy neural architectures. For example, with a FLOPs budget of approximately 450M, our method achieves a remarkable 79.0% Top-1 test accuracy on ImageNet, which is higher than the leading NAS approach DONNA (Moons et al., 2021) and over 62 times faster in search cost. This dramatic improvement in efficiency highlights the strength of RZ-NAS in balancing performance and computational cost, making it particularly advantageous in scenarios where fast architecture search is essential, especially in resource-constrained environments. Moreover, for a given Zero-Cost method, our proposed search strategy built on this Zero-Cost method can always achieve an improvement in accuracy across different FLOP budgets.



(a) NAS-Bench-201 (b) DARTS

Figure 3. Ablation results on different search spaces.

Table 5. Top-1 test error on ImageNet under various FLOP budgets. For the "Method" column, "MS" means multi-shot NAS; "OS" is short for one-shot NAS; Scaling represents network scaling methods; "ZC" is short for Zero-Cost NAS. ♢: The test error is reported from (Li et al., 2023). The best results are shown in **bold**.

Budget(maximal #FLOPs)	Approach	Top-1(%)	Method	Cost(GPU days)
450M	EfficientNet♢	23.9	MS	3800
	NasNet-B♢	27.2	MS	1800
	DONNA	22.0	OS	25
	Zen-NAS	27.9	ZC	0.3
	Ours(Zen-NAS)	27.0	ZC	0.3
	ZiCo	22.0	ZC	0.4
	Ours(ZiCo)	**21.0**	ZC	0.4
600M	EfficientNet	21.9	Scaling	3800
	DARTS♢	26.7	OS	4
	PC-DARTS♢	24.2	OS	3.8
	DONNA♢	21.6	OS	25
	Synflow	20.9	ZC	0.25
	Ours(Synflow)	20.0	ZC	0.25
	Zen-NAS	20.9	ZC	0.4
	Ours(Zen-NAS)	20.0	ZC	0.4
	ZiCo	20.6	ZC	0.4
	Ours(ZiCo)	**19.9**	ZC	0.4
1000M	EfficientNet♢	19.9	OS	3800
	sharpDARTS ♢	24.0	Scaling	-
	Zen-NAS♢	19.9	ZC	0.5
	Ours(Zen-NAS)	18.9	ZC	0.5
	ZiCo	19.5	ZC	0.5
	Ours(ZiCo)	**18.7**	ZC	0.5

4.2. Ablation Study

4.2.1. Explanation of Removing Different Components

In Figure 3, we evaluate RZ-NAS without in-context example, reflection module, code description, and textual description on NAS-Bench-201 and DARTS search spaces. This investigation aims to uncover the contribution of each prompt element to the overall effectiveness of RZ-NAS in accurately navigating the search space and identifying high-performing architectures. More detailed explanations and more ablation studies can be found in Appendix A.4.1.

Table 6. Performance comparison on CIFAR-10 using NAS-Bench-201 search space. RZ-NAS with the random selection strategy shows competitive results compared to the highest Zero-Cost score-based selection stragety.

Method		GraSP	Zen-NAS	ZiCo
Test Error	RZ-NAS	5.4 ± 0.07	$\mathbf{2.50 \pm 0.17}$	$\mathbf{2.41 \pm 0.13}$
%	RZ-NAS (Highest Proxy Score)	$\mathbf{5.38 \pm 0.12}$	2.52 ± 0.15	2.42 ± 0.20

Table 7. Perfromance comparison between the original prompt template and two other different prompts phrasing on CIFAR-10. By keeping the in-context example and code block fixed in the prompt template, we use GPT-4o to rephrase the semantic prompt. Different prompt phrasings can influence the outcomes, yet they still demonstrate superiority over the original Zero-Cost methods.

Method		GraSP	Gradnorm	Synflow	Zen-NAS	ZiCo
Test Error	Original Zero-Cost Proxy	6.0 ± 0.13	5.7 ± 0.05	5.9 ± 0.11	2.55 ± 0.04	2.45 ± 0.11
	RZ-NAS	$\mathbf{5.4 \pm 0.07}$	$\mathbf{5.2 \pm 0.12}$	$\mathbf{4.3 \pm 0.07}$	$\mathbf{2.50 \pm 0.17}$	2.41 ± 0.13
%	RZ-NAS (Prompt Rephrase 1)	5.5 ± 0.18	$\mathbf{5.2 \pm 0.08}$	$\mathbf{4.3 \pm 0.17}$	2.51 ± 0.08	$\mathbf{2.40 \pm 0.20}$
	RZ-NAS (Prompt Rephrase 2)	5.7 ± 0.10	5.3 ± 0.07	4.7 ± 0.09	2.54 ± 0.14	2.44 ± 0.12

4.2.2. Performance Comparison With Different Architecture Selection Strategies

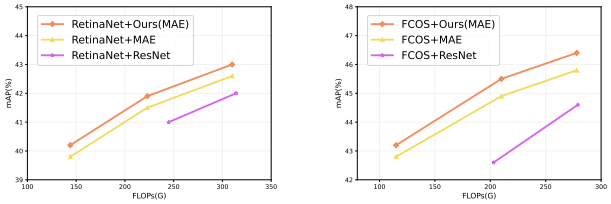
In each iteration of RZ-NAS, the architecture selected for mutation is randomly chosen from the population, rather than selecting the architecture with the highest Zero-Cost score. Table 6 shows that the random selection strategy achieves competitive performance. The random selection strategy maintains population diversity and avoids premature convergence. LLM-guided mutations steer low-scoring candidates toward better solutions. Unlike traditional algorithms, which may lead to inefficiency, LLM-guided mutations intelligently explore architectures, maintaining diversity, and accelerating convergence. This aligns with previous LLM-to-NAS work such as ELM (Lehman et al., 2024) and EvoPrompting (Chen et al., 2023).

4.2.3. Performance Comparison with Rephrasing Prompt

We further evaluate the effectiveness of the structured prompt template. GPT4o is used to rephrase prompts. Table 7 shows consistent performance with different rephrasing changes on CIFAR-10, demonstrating the robustness of the structured prompt template in RZ-NAS. Previous research confirms that LLMs are insensitive to phrasing changes in structured tasks when examples, I/O formats, and specific context remain consistent (Raffel et al., 2020).

4.3. Object Detection

RZ-NAS can also be applied to search efficient backbones for object detection tasks, leveraging the Zero-Cost detection strategy MAE-DET (Sun et al., 2022). Following the comparison setting, our method is designed to align with traditional ResNet and the MAE-DET strategy with similar FLOPs and parameter counts. For training, we use ResNet-like backbones on the COCO dataset (Lin et al., 2014), incorporating multi-scale training and Synchronized Batch Normalization. Figure 4(a) and Figure 4(b) present the performance comparison across different heads (RetinaNet and FCOS). Notably, RZ-NAS improves the COCO mAP while maintaining similar FLOPs.



(a) RetinaNet (b) FCOS

Figure 4. Object detection with RetinaNet and FCOS.

5. Conclusion

In this work, we proposed a LLM-to-NAS framework via a reflective Zero-Cost strategy. We designed a structured, prompt-based approach that combines text- and code-level understanding, enabling the LLM to generate robust architectures tailored to diverse search spaces and tasks. By coupling architecture generation with reflections and training-free metrics, RZ-NAS effectively leverages the reasoning capabilities of LLMs to enhance NAS performance. Experimental results demonstrate that RZ-NAS outperforms state-of-the-art LLM-to-NAS and Zero-Cost NAS methods across multiple datasets and search spaces, achieving competitive or superior performance with significantly reduced search costs. In future work, we will integrate more cost-efficient open-source models into our framework to improve cost-efficiency. These findings highlight the versatility and scalability of LLMs as powerful tools for efficient and high-performance architecture design.

Acknowledgements

This work was supported by the Frontier Technology R&D Program of Jiangsu Province (#BF2024005), Nanjing Science and Technology Program (#202304016), and the Collaborative Innovation Center of Novel Software Technology and Industrialization, Jiangsu, China.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Abdelfattah, M. S., Mehrotra, A., Łukasz Dudziak, and Lane, N. D. Zero-cost proxies for lightweight NAS. In *Proceedings of the International Conference on Learning Representations*, 2021.

Chen, A., Dohan, D., and So, D. EvoPrompting: Language models for code-level neural architecture search. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, volume 36, pp. 7787–7817, 2023.

Chen, X., Xie, L., Wu, J., and Tian, Q. Progressive DARTS: Bridging the optimization gap for nas in the wild. *International Journal of Computer Vision*, pp. 1–18, 2020.

Chen, X., Wang, R., Cheng, M., Tang, X., and Hsieh, C.-J. DrNAS: Dirichlet neural architecture search. In *Proceedings of the International Conference on Learning Representations*, 2021.

Denny, P., Kumar, V., and Giacaman, N. Conversing with copilot: Exploring prompt engineering for solving cs1 problems using natural language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pp. 1136–1142, 2023.

Gao, J., Xu, H., Shi, H., Ren, X., Yu, P. L., Liang, X., Jiang, X., and Li, Z. Autobert-zero: Evolving bert backbone from scratch. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 10663–10671, 2022.

Hanin, B. and Rolnick, D. Complexity of Linear Regions in Deep Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 2596–2604. PMLR, 2019.

Hao, H., Zhang, X., and Zhou, A. Large language models as surrogate models in evolutionary algorithms: A preliminary study. *Swarm and Evolutionary Computation*, 91:101741, 2024.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Javaheripi, M., de Rosa, G. H., Mukherjee, S., Shah, S., Religa, T. L., Mendes, C. C., Bubeck, S., Koushanfar, F., and Dey, D. LiteTransformerSearch: training-free neural architecture search for efficient language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022.

Jiang, S., Ji, Z., Zhu, G., Yuan, C., and Huang, Y. Operation-level early stopping for robustifying differentiable NAS. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, 2023.

Krishnakumar, A., White, C., Zela, A., Tu, R., Safari, M., and Hutter, F. NAS-bench-suite-zero: accelerating research on zero cost proxies. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022.

Lee, K., Hwang, D., Park, S., Jang, Y., and Lee, M. Reinforcement learning from reflective feedback (RLRF): Aligning and improving llms via fine-grained self-reflection. *arXiv preprint arXiv:2403.14238*, 2024.

Lee, N., Ajanthan, T., and Torr, P. H. S. SNIP: single-shot network pruning based on connection sensitivity. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Lehman, J., Gordon, J., Jain, S., Ndousse, K., Yeh, C., and Stanley, K. O. *Evolution Through Large Models*, pp. 331–366. Springer Nature Singapore, 2024.

Li, G., Yang, Y., Bhardwaj, K., and Marculescu, R. Zico: Zero-shot nas via inverse coefficient of variation on gradients. In *Proceedings of the Eleventh International Conference on Learning Representations*, 2023.

Lin, M., Wang, P., Sun, Z., Chen, H., Sun, X., Qian, Q., Li, H., and Jin, R. Zen-NAS: A zero-shot nas for high-performance deep image recognition. In *Proceedings of the International Conference on Computer Vision*, 2021.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *Proceedings of the European Conference on Computer Vision*, pp. 740–755, 2014.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision*, pp. 19–35, 2018.

Liu, F., Tong, X., Yuan, M., and Zhang, Q. Algorithm evolution using large language model. *arXiv preprint arXiv:2311.15249*, 2023.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Liu, S., Chen, C., Qu, X., Tang, K., and Ong, Y.-S. Large language models as evolutionary optimizers. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1–8, 2024.

Lopes, V., Alirezazadeh, S., and Alexandre, L. A. EPE-NAS: Efficient performance estimation without training for neural architecture search. In *Proceedings of the 30th International Conference on Artificial Neural Networks*, pp. 552–563, 2021.

Min, S., Lyu, X., Holtzman, A., Artetxe, M., Lewis, M., Hajishirzi, H., and Zettlemoyer, L. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11048–11064, 2022.

Moons, B., Noorzad, P., Skliar, A., Mariani, G., Mehta, D., Lott, C., and Blankevoort, T. Distilling optimal neural networks: Rapid search in diverse spaces. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 12229–12238, 2021.

Nasir, M. U., Earle, S., Togelius, J., James, S., and Cleghorn, C. LLMatic: Neural architecture search via large language models and quality diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 89, pp. 1110–1118, 2024.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameters sharing. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 4095–4104, 2018.

Qin, R., Hu, Y., Yan, Z., Xiong, J., Abbasi, A., and Shi, Y. FL-NAS: Towards fairness of nas for resource constrained devices via large language models. In *Proceedings of the 29th Asia and South Pacific Design Automation Conference*, pp. 429–434, 2024.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21 (1):5485–5551, 2020.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, pp. 4780–4789, 2019.

Renze, M. and Guven, E. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*, 2024.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J. R., Ellenberg, J. S., Wang, P., Fawzi, O., Kohli, P., Fawzi, A., Grochow, J., Lodi, A., Mouret, J.-B., Ringer, T., and Yu, T. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2023.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks . In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Serra, T., Tjandraatmadja, C., and Ramalingam, S. Bounding and counting linear regions of deep neural networks. In *Proceedings of the International Conference on Machine Learning*, pp. 4558–4566, 2018.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pp. 8634–8652, 2023.

Sun, Z., Lin, M., Sun, X., Tan, Z., Li, H., and Jin, R. Maedet: Revisiting maximum entropy principle in zero-shot nas for efficient object detection. In *Proceedings of the International Conference on Machine Learning*, pp. 20810–20826, 2022.

Sun, Z., Sun, Y., Yang, L., Lu, S., Mei, J., Zhao, W., and Hu, Y. Unleashing the power of gradient signal-to-noise ratio for zero-shot nas. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5740–5750, 2023.

Tanaka, H., Kunin, D., Yamins, D. L. K., and Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pp. 6377–6389, 2020.

Termritthikun, C., Jamtsho, Y., Ieamsaard, J., Muneesawang, P., and Lee, I. Eeea-net: An early exit evolutionary neural architecture search. *Engineering Applications of Artificial Intelligence*, 104:104397, 2021.

van Stein, N. and Bäck, T. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29(2):331–345, 2024.

Wang, C., Zhang, G., and Grosse, R. Picking winning tickets before training by preserving gradient flow. In *Proceedings of the International Conference on Learning Representations*, 2020.

Wang, R., Cheng, M., Chen, X., Tang, X., and Hsieh, C.-J. Rethinking architecture selection in differentiable nas. In *Proceedings of the International Conference on Learning Representations*, 2021.

Wei, L., He, Z., Zhao, H., and Yao, Q. Unleashing the power of graph learning through llm-based autonomous agents. *arXiv preprint arXiv:2309.04565*, 2023.

Xiang, L., Dudziak, L., Abdelfattah, M. S., Chau, T., Lane, N. D., and Wen, H. Zero-cost operation scoring in differentiable architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 10453–10463, 2023.

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

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

Ye, H., Wang, J., Cao, Z., Berto, F., Hua, C., Kim, H., Park, J., and Song, G. ReEvo: Large language models as hyper-heuristics with reflective evolution. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2024.

Ye, P., Li, B., Li, Y., Chen, T., Fan, J., and Ouyang, W. b-darts: Beta-decay regularization for differentiable architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10874–10883, 2022.

Yu, C., Liu, X., Wang, Y., Liu, Y., Feng, W., Deng, X., Tang, C., and Lv, J. GPT-NAS: Evolutionary neural architecture search with the generative pre-trained model. *arXiv preprint arXiv:2305.05351*, 2023.

Zhang, L., Ergen, T., Logeswaran, L., Lee, M., and Jurgens, D. SPRIG: Improving large language model performance by system prompt optimization. *arXiv preprint arXiv:2410.14826*, 2024.

Zhang, X., Hou, P., Zhang, X., and Sun, J. Neural architecture search with random labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10907–10916, 2021.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

Zhu, G., Wang, W., Xu, Z., Cheng, F., Qiu, M., Yuan, C., and Huang, Y. PSP: Progressive space pruning for efficient graph neural architecture search. In *Proceedings of the IEEE 38th International Conference on Data Engineering*, pp. 2168–2181, 2022.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2017.

A. Appendix

A.1. Illustration of Zero-cost Proxyies

We illustrate the mentioned zero-cost proxies in our experiment.

GraSP (Wang et al., 2020). It attempts to approximate the change in gradient norm (instead of loss), which is denoted as,

$$S(\theta) = -(H\frac{\vartheta L}{\vartheta \theta}) \odot \theta \tag{2}$$

where L is the loss function of a neural network with parameters θ, H is the Hessian.

Gradnorm (Abdelfattah et al., 2021). Gradnorm sums the Euclidean norm of the gradients after a single mini-batch of training data, which can be denoted as,

$$S_{\text{GradNorm}}(\theta) = \sum_{i=1}^{N} \|\nabla_\theta L(x_i, y_i; \theta)\|_2 \tag{3}$$

where $L(x_i, y_i; \theta)$ is the loss function parameterized by θ, and ∇_θ represents the gradient with respect to θ.

Synflow (Tanaka et al., 2020). Synflow simply computes the loss, which is the product of all the parameters in the network.

$$S(\theta) = \frac{\vartheta L}{\vartheta \theta} \odot \theta \tag{4}$$

ZenNAS (Lin et al., 2021). ZenNAS approximates the gradient w.r.t feature maps and measures the complexity of the neural network. It computes

$$\bar{\sigma_i} = \sqrt{\textstyle\sum_j \sigma_{i,j}^2 / m} \tag{5}$$

where $\sigma_{i,j}$ is the mini-batch standard deviation statistic of the j-th channel of the i-th BN layer.

ZiCo (Li et al., 2023). ZiCo is the SOTA Zero-Cost NAS method that can consistently surpass many conventional NAS strategies. Zico has a relatively complicated computation method that considers both absolute mean and standard deviation,

$$\sum_{l=1}^{D} log(\sum_{\omega \in \theta_l} \frac{\mathbb{E}\left[|\nabla_w L(X_i, y_i; \Theta|\right]}{\sqrt{Var}(|\nabla_\omega (X_i, y_i; \Theta)|)}) \tag{6}$$

where Θ denotes the initial parameters of the given network, Θ_l denotes the parameters of the network, and ω represents each element in Θ_l, X_i and y_i are the ith input batch and corresponding labels from the training set.

MAE-DET (Sun et al., 2022). Beyond image classification, MAE-DET extends to the field of object detection and instance segmentation. DeepMAD regards the neural network as an information system and calculates the differential entropy $H(x)$ of the output feature map,

$$H(x) = \frac{1}{2} log(Var(\hat{h}^D)) + \sum_{l=1}^{D} log(\gamma^l) \tag{7}$$

where $\hat{h}^D$ is the variance of the last feature map and γ^l is constant.

A.2. Example of Reflection

We provide an example of the iterative reflection under the NAS-Bench-201 search space and the GradNorm Zero-Cost proxy.

- The input of the randomly selected architecture:


```
{"arch":"|nor_conv_1x1~0|+|skip_connect~0|skip_connect~1|+|skip_connect~0|skip_connect~1|
skip_connect~2|", "type":"Gradnorm", "score":3.5}
```

- Output after LLM mutation:

```
{"arch":"|nor_conv_1x1~0|+|nor_conv_1x1~0|skip_connect~1|+|nor_conv_3x3~0|none~1|skip_connect
~2|", "type":"Gradnorm", "score":4.2, exception:""}
```

- Output of the LLM reflection module:

```
"Try replacing one of the skip_connect operations with 3x3 Conv to increase the expressiveness of
 the cell. Convolution operations like 3x3 Conv can capture more spatial features compared to
skip connections, potentially improving the architecture's performance."
Our architecture Genotype might exceed the limit of MAX_LAYERS. The exception will output as "
Error: out of the limit of MAX_LAYERS."
```

- Output of reflection module with exception:

```
"The "MAX_LAYERS" error indicates that the architecture exceeds the allowed number of layers. You
 can reduce the number of layers in the architecture. For example, consider reducing one of the
ResK1K5K1 layers to a simpler operation like ConvK1BNRELU or reduce the number of channels to
stay within the limit."
```

A.3. Prompts

We present an example below to illustrate how to design the prompt.

A.3.1. System Prompt

- Define Role and Task:

```
# I am an expert in understanding the zero-cost NAS methods and evolutionary algorithms of neural
 architecture search.

# I am good at Python programming evolutionary algorithms.

# My task is to guide the mutation process of the architecture to get an optimal architecture
with a better zero-cost proxy score. I should understand the given task and dataset, search space
 and network descriptions, and mutate the selected architecture textual genotype to generate
better models.

# I should only generate valid Python code.
```

- Define Search Space (Here we take an operation in the search space as an example):

````
- I should generate an optimal architecture by mutating an operation in the search space. All
operations are defined below.
- **class SuperResK1K3K1**: This class defines a residual block with convolutional layers having
kernel sizes of 1x3x1. It is designed for applications that require smaller kernel sizes to
extract local features while maintaining a residual connection for improved gradient flow.
   - The `__init__(self, in_channels, out_channels, stride, bottleneck_channels, sub_layers)`
   parameters are:
      - `in_channels`: Number of input channels.
      - `out_channels`: Number of output channels.
      - `stride`: Stride length for the convolutional layers.
      - `bottleneck_channels`: Number of channels in the bottleneck layer for reduced
      dimensionality.
      - `sub_layers`: Number of sub-layers in the block.
   - Function `forward` code:
   ```python
   def forward(self, x):
      output = x
      for block in self.block_list:
         output = block(output)
      return output
   ```
````

- System Reflection

```
- I am the expert in the domain of evolutionary mutation. I should reflect and give hints to
design better architectures.
```

- Network Construction

```
- The input architecture is defined as a string variable named `block_str`. The format is: <
operation1(para1,para2)><operation2(para1,para2)>...<operationk(para1,para2)>. For example, the
block_str is "SuperConvK3BNRELU(3,64,1,1)SuperResK1K5K1(64,168,1,16,3)SuperResK1K3K1
(168,80,2,32,4)".
- The parameter `bottleneck_channels` and `out_channels` **must not exceed** 2048 and **must** be
 a multiple of 8. The `sub_layers` should be at least 1.
- The output channels of the previous block should match the input channels of the next block.
```python
def create_block_list_from_str(s):
   block_list = []
   while len(s) > 0:
      is_found_block_class = False
      for the_block_class_name in blocks_dict.keys():
         tmp_idx = s.find('(')
         if tmp_idx > 0 and s[0:tmp_idx] == the_block_class_name:
            is_found_block_class = True
            the_block_class = _all_netblocks_dict_[the_block_class_name]
            the_block, remaining_s = the_block_class.create_from_str(s, no_create=no_create, **
            kwargs)
            if the_block is not None:
               block_list.append(the_block)
            s = remaining_s
            if len(s) > 0 and s[0] == ';':
               return block_list, s[1:]
            break
         pass # end if
      pass # end for
      assert is_found_block_class
   pass # end while
   return block_list
class MasterNet(PlainNet.PlainNet):
   def __init__(self,):
      self.block_list = create_block_list_from_str(block_str)
    def forward(self, x):
      output = x
       for block_id, the_block in enumerate(self.block_list):
          output = the_block(output)
       output = F.adaptive_avg_pool2d(output, output_size=1)
       output = torch.flatten(output, 1)
       output = self.fc_linear(output)
       return output
```
```

- Architecture Mutation:

```
- I should understand the key code of architectures and mutate one operation of the architecture
by replacing it with another operation defined in the search space list. I can mutate the
operation type and parameters of operations.
```

- Compute Zero-Cost NAS Proxy:

```
- The type of zero-cost proxy is gradnorm. The computing code is below.

```python
import os, sys, time
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
import torch
from torch import nn
import numpy as np

def network_weight_gaussian_init(net: nn.Module):
   with torch.no_grad():
      for m in net.modules():
         if isinstance(m, nn.Conv2d):
            nn.init.normal_(m.weight)
            if hasattr(m, 'bias') and m.bias is not None:
               nn.init.zeros_(m.bias)
         elif isinstance(m, (nn.BatchNorm2d, nn.GroupNorm)):
            nn.init.ones_(m.weight)
            nn.init.zeros_(m.bias)
         elif isinstance(m, nn.Linear):
            nn.init.normal_(m.weight)
            if hasattr(m, 'bias') and m.bias is not None:
               nn.init.zeros_(m.bias)
         else:
            continue
   return net
```
```

```
import torch.nn.functional as F
def cross_entropy(logit, target):
   # target must be one-hot format!!
   prob_logit = F.log_softmax(logit, dim=1)
   loss = -(target * prob_logit).sum(dim=1).mean()
   return loss

def compute_nas_score(gpu, model, resolution, batch_size):

   model.train()
   model.requires_grad_(True)

   model.zero_grad()

   if gpu is not None:
      torch.cuda.set_device(gpu)
      model = model.cuda(gpu)

   network_weight_gaussian_init(model)
   input = torch.randn(size=[batch_size, 3, resolution, resolution])
   if gpu is not None:
      input = input.cuda(gpu)
   output = model(input)
   # y_true = torch.rand(size=[batch_size, output.shape[1]], device=torch.device('cuda:{}'.format
   (gpu))) + 1e-10
   # y_true = y_true / torch.sum(y_true, dim=1, keepdim=True)

   num_classes = output.shape[1]
   y = torch.randint(low=0, high=num_classes, size=[batch_size])

   one_hot_y = F.one_hot(y, num_classes).float()
   if gpu is not None:
      one_hot_y = one_hot_y.cuda(gpu)

   loss = cross_entropy(output, one_hot_y)
   loss.backward()
   norm2_sum = 0
   with torch.no_grad():
      for p in model.parameters():
         if hasattr(p, 'grad') and p.grad is not None:
            norm2_sum += torch.norm(p.grad) ** 2

   grad_norm = float(torch.sqrt(norm2_sum))

   return grad_norm
```

A.3.2. IN-CONTEXT EXAMPLE

- Define Role and Task:

```
Here is the example:
<|im_start|>user
Mutate the below architecture.
```json
{"arch":"ConvK3BNRELU(3,32,1,1)ResK1K5K1(32,120,1,40)ResK1K5K1(120,176,2,32)", "type":"synflow",
"score":0.9}
```
<|im_end|>
<|im_start|>assistant
```json
{"arch":"ConvK1BNRELU(3,32,1,1)ResK1K5K1(32,120,1,40)ResK1K5K1(120,176,2,32)", "type":"synflow",
"score":1.9}
```
<Thinking>
- I can mutate `ConvK3BNRELU` into `ConvK1BNRELU` and get the optimal architecture by improving
the zero-cost NAS score from 0.9 to 1.9. Other mutation methods cannot get such high score.
</thinking>
<|im_end|>
```

A.3.3. USER PROMPT

- Define Role and Task:

```
Now given the new architecture below with a new user:
<|im_start|>user
```

```
Mutate the below architecture.
```json
{``arch'':{{architecture}},``type'':<type>,``score'':<score>}
```
<|im_end|>
<|im_start|>assistant
```json
```

A.4. Ablation Study

A.4.1. Explanation of Removing Different Components

The Role of In-context Example We create a "*w/o In-context Example*" variant by removing the in-contextual example module from the LLM input prompt and conducting the same search process on NAS-Bench-201. We found that this setting performs worse with increased exceptions from 2% to 7% and worse accuracy. This demonstrates the critical role that in-context examples play in guiding the LLM to make informed predictions and decisions by providing it with prior examples of successful architectures. Without these examples, the model struggles to establish meaningful patterns in the search space, leading to suboptimal architecture selections and a higher likelihood of generating invalid output.

Reflection Module The "*w/o Reflection Module*" variant is created to reflect on previous architecture performance and error exceptions. The absence of reflection leads to a performance drop and increased exceptions. Without the reflection module, we observe a significant decline in performance and a noticeable increase in exceptions from 2% to 5% during the search process. This suggests that the absence of reflection leads to a lack of iterative improvement, as the system fails to learn from past successes and failures.

The reflection module not only improves accuracy, but also reduces the exception rate by allowing the model to adjust its search strategy dynamically. For instance, when encountering frequent invalid architectures, the reflection module helps the system identify problematic patterns in the search and generate solutions that better adhere to the constraints of the search space. Furthermore, reflection enables the system to adaptively refine its exploration of the search space, prioritizing areas with higher potential while discarding unpromising regions. Without this module, the RZ-NAS framework essentially operates in a static, trial-and-error manner, resulting in lower search efficiency and less competitive performance compared to the full RZ-NAS setup.

Code Description We create a "*w/o Code Description*" variant by removing the three code descriptions in the system prompt that detail the NAS process and architecture construction. This results in a significant decrease in RZ-NAS performance, highlighting the importance of these descriptions in providing contextual guidance to the LLM. These descriptions help the LLM understand the intricacies of the NAS process and ensure that its outputs align with the search objectives. Without these descriptions, the model struggles to interpret the search space effectively, leading to less accurate and less efficient architecture searches.

Textual Description We also create a "*w/o Textual Description*" variant by removing textual explanations from the system prompt and relying solely on instructions in the user prompt and code blocks. The results show that textual descriptions play a crucial role in aligning the LLM's understanding of the problem and its objectives. Without them, the code blocks alone are insufficient, especially given the complexity of the search space and network construction in RZ-NAS. Unlike simpler and smaller search spaces in prior studies, RZ-NAS involves a more sophisticated setup, requiring detailed textual descriptions to provide the necessary context for effective search guidance. This finding highlights the need for a balance between textual and code-based inputs to maximize the effectiveness of the LLM in NAS tasks.

A.4.2. Performance Comparison on different LLMs

We evaluate RZ-NAS with different LLMs under the NAS-Bench-201 search space. This comparison allows us to assess how the choice of LLM affects the overall effectiveness of our approach. To ensure a comprehensive comparison, we select three widely used LLMs that have garnered significant attention in the research community. In addition to GPT4-o, which has established itself as a leading model in various applications, we include LLaMA 3.1 [3], which displays impressive code generation capabilities. Furthermore, we incorporate Claude 3.5 [4]. Known for its strong performance in understanding and

[3] https://ai.meta.com/blog/meta-llama-3-1/

[4] https://claude.ai/
```

Table 9. Accuray Comparison under the NAS-Bench-201 search space with different LLMs. The best results are shown in **bold**.

Method	CIFAR-10			CIFAR-100			ImageNet-16-120		
	GPT4-o	Llama 3.1	Claude3.5	GPT4-o	Llama 3.1	Claude3.5	GPT4-o	Llama 3.1	Claude3.5
Ours(GraSP)	**92.79** ± 0.80	91.67 ± 1.02	89.97 ± 0.24	**69.34** ± 1.98	67.45 ± 0.79	66.31 ± 0.33	**43.16** ± 2.64	42.45 ± 0.39	41.23 ± 0.18
Ours(Gradnorm)	**93.99** ± 3.44	93.41±0.30	93.21 ± 0.13	**71.61** ± 2.20	67.48 ± 0.89	64.12 ± 1.95	**42.61** ± 1.38	41.48 ± 0.59	41.83 ± 0.28
Ours(Synflow)	93.47 ± 0.74	**93.51** ± 0.44	93.08 ± 2.42	73.21 ± 0.12	**73.33** ± 0.40	71.59 ± 0.32	46.34 ± 0.32	**46.44** ± 0.12	45.22 ± 1.10
Ours(Zen-NAS)	**93.48** ± 0.00	93.34 ± 0.27	92.75 ± 0.81	**71.35** ± 1.51	70.68 ± 0.60	60.91 ± 0.88	**45.12** ± 0.00	41.44 ± 0.10	41.43 ± 0.40
Ours(ZiCo)	**94.24** ± 0.12	93.31 ± 1.40	91.55 ± 0.72	**74.30** ± 0.21	71.37 ± 2.13	71.23 ± 1.44	**46.18** ± 0.33	45.57 ± 0.12	44.37 ± 0.30

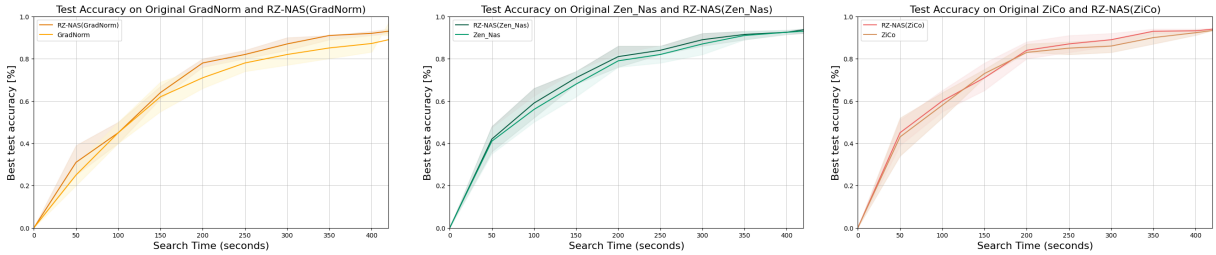
Table 10. Performance comparison between the temperature sampling method and the fixed temperature setting on CIFAR10 under the NAS-Bench-201 search space.

Temperature	GraSP		Gradnorm		Synflow		Zen-NAS		ZiCo	
	valid	test	valid	test	valid	test	valid	test	valid	test
0	89.91±0.11	92.11±1.24	90.65±0.41	93.12±0.63	90.90±0.33	92.89±0.56	90.37±0.31	92.76±0.24	90.65±0.22	93.68±0.32
0.2	90.08±0.45	92.55±0.53	90.74±0.65	93.44±0.94	**91.47±0.58**	**93.47±0.86**	90.52±0.45	92.81±0.25	90.94±0.20	93.89±0.23
0.4	90.10±0.34	**92.79±0.46**	**90.76±1.10**	93.91±1.07	90.89±0.32	92.85±0.44	90.83±0.30	92.82±0.47	91.22±0.14	94.11±0.22
0.6	89.92±0.44	92.34±0.27	90.44±1.21	93.02±0.89	91.01±0.29	93.18±0.61	**91.03±0.87**	93.46±0.61	91.15±0.12	94.04±0.14
0.8	89.98±0.74	92.20±1.08	90.48±1.32	93.20±1.72	91.10±0.27	93.02±0.53	90.77±0.55	92.41±0.44	90.93±0.63	93.62±0.41
1.0	89.45±1.10	89.33±1.20	90.10±1.89	93.10±2.01	90.91±0.85	92.89±0.76	90.63±0.66	92.97±0.83	89.96±0.13	93.48±0.40
Uniform sampling	**90.11±1.00**	**92.79±0.80**	**90.76±1.77**	**93.99±3.44**	91.45±0.42	**93.47±0.74**	**91.02±0.44**	**93.48±0.00**	**91.45±0.10**	**94.24±0.12**

generating natural language, Claude 3.5 has also shown potential in code-related tasks. The results are shown in Table 9, our findings reveal that RZ-NAS delivers similar results across different LLMs, while the GPT4-o model consistently outperforms the other LLMs in terms of accuracy.

A.4.3. Performance Comparison under different temperature strategies

In RZ-NAS, the temperature of the large language model is sampled uniformly from the domain, not a fixed value. The comparison with different fixed temperatures is shown in Table 10. The results demonstrate the superiority of the temperature sampling approach in RZ-NAS.



(a) GraSP (b) Zen-NAS (c) ZiCo

Figure 5. Visualize the test accuracy of three original Zero-Cost strategies and our LLM-guided methods on CIFAR10 under the NAS-Bench-201 search space over time. We record best test accuracy every 50 seconds in the population.

A.5. Visualization of Test Accuracy Comparison Between Original Zero-Cost Strategies and RZ-NAS

We visualize the test accuracy comparison between original Zero-Cost proxies and RZ-NAS over time in Figure 5. The CIFAR10 dataset and the NAS-Bench-201 search space are used. The results show that our approach achieves higher test accuracy during the training process compared to the original Zero-Cost strategies.