CoLLM-NAS: Collaborative Large Language Models for Efficient Knowledge-Guided Neural Architecture Search

Anonymous CVPR submission

Paper ID

Abstract

*The integration of Large Language Models (LLMs) with Neural Architecture Search (NAS) has introduced new possibilities for automating the design of neural architectures. However, most existing methods face critical limitations, including architectural invalidity, computational inefficiency, and inferior performance compared to traditional NAS. In this work, we present **Co**llaborative **LLM**-based **NAS** (CoLLM-NAS), a two-stage NAS framework with knowledge-guided search driven by two complementary LLMs. Specifically, we propose a stateful Navigator LLM to guide search direction, a stateless Generator LLM to synthesize high-quality candidates, and a Coordinator module to orchestrate inter-LLM communication and manage evaluation processes. CoLLM-NAS efficiently guides the search process by combining LLMs' inherent knowledge of structured neural architectures with progressive knowledge from iterative feedback and historical trajectory. Experimental results on ImageNet and NAS-Bench-201 show that CoLLM-NAS surpasses existing NAS methods and conventional search algorithms, achieving new state-of-the-art results while significantly reducing search costs by 4–10×. Furthermore, CoLLM-NAS consistently enhances the performance and efficiency of various two-stage NAS methods (e.g., OFA, SPOS, and AutoFormer) across diverse search spaces (e.g., MobileNet, ShuffleNet, and AutoFormer), demonstrating its excellent generalization.*

1. Introduction

Designing neural architectures remains a critical task in the era of deep learning. Neural Architecture Search (NAS), a cornerstone of AutoML, is dedicated to automating the discovery of optimal neural architectures. Early NAS methods typically employ reinforcement learning [32, 42, 43] or evolutionary algorithms [28, 29] to search for high-performing architectures. While effective, these methods require training numerous independent architectures from scratch, resulting in prohibitive computational costs. To address this issue, one-shot NAS [13, 23] is developed, utilizing a weight-sharing supernet to amortize training expenses. Two-stage NAS [4, 7, 13] further decouples this process into two phases, *i.e.,* supernet training and architecture search. Although this approach avoids costly retraining by inheriting supernet weights, conventional search algorithms (*e.g.,* evolutionary algorithms) still require sampling and evaluating thousands of candidates in vast search spaces during the second phase to find optimal architectures, incurring notable test costs and risking convergence to local optima.

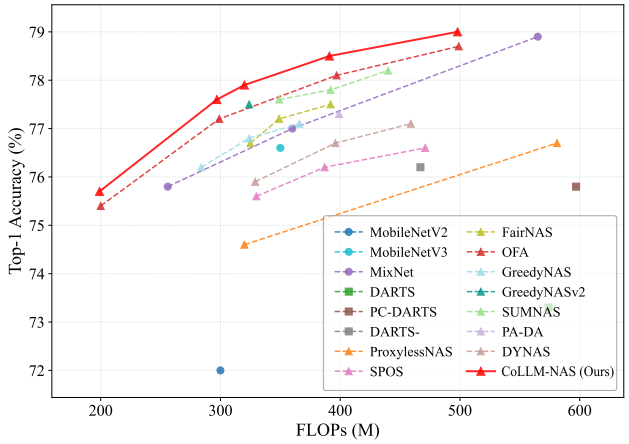


Figure 1. FLOPs-accuracy trade-off of architectures discovered by different NAS methods on ImageNet.

Recently, large language models (LLMs) have emerged as novel participants in the NAS landscape, leveraging their powerful representation capabilities, code generation proficiency, and reasoning abilities to enhance the search process from diverse perspectives [36]. However, most existing LLM-based NAS methods [6, 26] directly modify architectures at the code level within unconstrained programming-language token spaces, often resulting in architectural invalidity, limited robustness, and independent training of each candidate. Consequently, these methods fail to surpass established NAS baselines on standard benchmarks [10, 11]

while generating substantial computational overhead [26]. To address these limitations, we propose integrating the advantages of LLMs with mature two-stage NAS methods to provide more effective and efficient neural architecture design. Specifically, we aim to enhance the search phase of two-stage NAS by replacing conventional search algorithms with knowledge-guided LLM reasoning, enabling intelligent navigation of the search space to accelerate convergence to high-performing regions.

To this end, this paper introduces a novel **Co**llaborative **LLM**-based **NAS** framework (CoLLM-NAS) that leverages synergistic interactions between two complementary LLMs to efficiently discover high-performing architectures for two-stage NAS. Concretely, we design three key components: (1) A stateful *Navigator LLM* that provides search strategies through dynamic refinement, leveraging iterative evaluation feedback and historical trajectory analysis. (2) A stateless *Generator LLM* that synthesizes high-quality architectures according to these strategies. (3) A *Coordinator* module that orchestrates inter-LLM communication, validates architectural legality, evaluates candidate performance, and maintains an architecture archive. At its core, CoLLM-NAS employs a collaborative framework where the *Navigator* and *Generator LLMs* work in concert, guiding the search with dual knowledge sources, *i.e.,* LLMs' inherent architectural prior and progressive insights learned from iterative feedback and historical trajectory. The stateful-stateless design further enhances the exploration-exploitation balance. Extensive experimental results demonstrate that our method can be applied to various two-stage NAS methods across diverse search spaces, consistently outperforming baselines under different resource constraints while significantly reducing search costs by 4–10×. As shown in Fig. 1, architectures discovered by CoLLM-NAS achieve superior FLOPs-accuracy trade-off compared to those from state-of-the-art NAS methods on ImageNet.

The main contributions of this work are as follows:

- We present a novel collaborative LLM-based NAS framework, CoLLM-NAS, to enhance two-stage NAS with knowledge-guided search. To our knowledge, this is the first work integrating LLMs with two-stage NAS.
- We propose three key components: a stateful *Navigator LLM* to provide adaptive search strategies, a stateless *Generator LLM* to synthesize high-quality architectures, and a *Coordinator* module to orchestrate LLMs' interactions and manage evaluation processes.
- Experiments demonstrate that CoLLM-NAS surpasses existing NAS methods across diverse search spaces both in performance and efficiency, achieving new SOTA results while reducing search costs significantly.

2. Related Work

2.1. Two-stage NAS

Two-stage NAS, a prominent branch of one-shot NAS, addresses the computational inefficiency of traditional NAS by employing weight-sharing mechanisms. This approach decomposes the NAS problem into two sequential phases:

$$\begin{aligned} w_{\mathcal{A}}^{*} &= \arg\min_{w_{\mathcal{A}}} \mathbb{E}_{\alpha\sim\Omega(\mathcal{A})}\left[\mathcal{L}(w_{\mathcal{A}}(\alpha), \mathcal{D}^{\text{train}})\right], \\ \alpha^{*} &= \arg\max_{\alpha\in\mathcal{A}} \mathcal{P}(w_{\mathcal{A}}^{*}(\alpha), \mathcal{D}^{\text{val}}) \quad \text{s.t.Cost}(\alpha) \le \Lambda, \end{aligned} \tag{1}$$

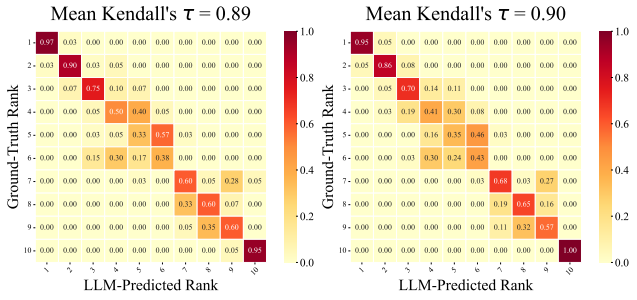
where $\mathcal{A}$ denotes the search space, $w_{\mathcal{A}}$ represents shared supernet weights, $w_{\mathcal{A}}(\alpha)$ indicates subnet weights inherited from the supernet, $\Omega(\mathcal{A})$ represents the sampling strategy, $\mathcal{P}$ and Λ are the performance evaluation function and the resource constraint, respectively. By decoupling architecture parameter optimization from network weight training, it eliminates conflicts in simultaneous updates and reduces mutual interference.

In the first stage, a weight-sharing supernet is trained via diverse sampling strategies. SPOS [13] pioneers uniform single-path sampling. OFA [4] enables multi-scale subnet extraction via progressive shrinking. In [24], Pa&da jointly samples paths and data for consistent training. DY-NAS [19] employs subnet-aware dynamic supernet training strategy. While AutoFormer [7] adapts this weight-sharing mechanism for vision transformers. In the second stage, optimal architectures are searched under resource constraints. During evaluation, candidates can directly inherit weights from the supernet, enabling efficient performance estimation without retraining. Since exhaustive evaluation remains infeasible, most methods employ random search [22], reinforcement learning [42], or evolutionary algorithms [4, 13] to efficiently explore the search space.

This paper introduces an LLM-based knowledge-guided search paradigm to replace conventional search algorithms in the second stage, aiming to significantly reduce evaluations and achieve better performance with lower costs.

2.2. LLM for NAS

The rise of LLMs offers a fundamentally new perspective for NAS by reformulating the search process through advanced reasoning capabilities and semantic understanding. GENIUS [41] pioneers GPT-4 as a black-box optimizer to generate and refine architectures via natural language. Evo-Prompting [6] repositions LLMs as mutation and crossover operators, employing evolutionary prompt engineering and soft prompt-tuning to explore a vast code-based search space. Similarly, LLMatic [26] leverages LLMs as operators but uniquely integrates Quality-Diversity Optimization to create diverse networks. A reflective zero-cost strategy is developed by RZ-NAS [20], combining LLM reflection



(a) CIFAR-10 (b) CIFAR-100

Figure 2. Consistency heatmap between LLM-predicted and ground-truth rankings on CIFAR-10 and CIFAR-100 within NAS-Bench-201 search space.

modules with training-free metrics. LM-Searcher [16] reformulates NAS as a ranking task using NCode, enabling cross-domain architecture search. While NADER [39] employs a multi-agent approach for neural architecture design, its open design space and full training requirements limit its scalability to ImageNet-scale datasets.

In contrast, this paper proposes a collaborative LLM-based NAS framework with knowledge-guided search. Moreover, our approach leverages a pre-trained supernet for efficient evaluation and inherits its pre-optimized search space to avoid the invalidity issue of generated architectures, enabling scalable search on large-scale datasets.

3. Method

In this section, we first validate LLMs' capability to comprehend structured neural architectures within hand-crafted search spaces. Building on this, we introduce CoLLM-NAS, our collaborative LLM-based NAS framework.

3.1. Architecture Comprehension in LLMs

Recent advances in LLMs demonstrate their capability to understand complex technical domains, acquired through pre-training on extensive technical literature. We hypothesize that *modern LLMs have internalized knowledge of neural architecture design principles that could be valuable for NAS*. To investigate this capability, we design an experiment on NAS-Bench-201.

Proof-of-Concept Experiment. We partition architectures in NAS-Bench-201 into 10 equal-sized subsets by their precomputed performance, and sample one architecture from each subset. The Qwen3-30B-A3B LLM [33] is prompted to rank these architectures based on its comprehension of neural network design principles, while being blinded to ground-truth accuracy. Architectures are represented in their standard graph-based encoding format. To ensure statistical robustness, we perform 40 independent trials on CIFAR-10 and CIFAR-100. As illustrated in Fig. 2, results reveal strong alignment between LLM predictions and empirical performance, achieving mean Kendall's τ values of 0.89 on CIFAR-10 and 0.90 on CIFAR-100. Moreover, the best architecture is correctly identified in the vast majority of trials. This demonstrates that *LLMs exhibit non-trivial comprehension of neural architecture performance patterns*, even when operating on structured representations within hand-crafted search spaces. Notably, modern LLMs may have encountered some architectures of NAS-Bench-201 in their training corpus. To avoid prior knowledge contamination, our prompts prevent transmission of any explicit information about the benchmark. Furthermore, as evident from the reasoning contents in Appendix C, the LLM's ranking decisions stem from its understanding of architectural principles (*e.g.,* operator effectiveness and information flow) rather than memorized architecture-performance mappings.

Algorithm 1 CoLLM-NAS: Exploring High-Performing Architectures via Collaborative LLMs

INPUT: Target accuracy P_{target}, Resource constraint Λ, Iteration limit T
OUTPUT: Best architecture α^*

Initialization: $\alpha^* \leftarrow \emptyset, p^* \leftarrow 0, \mathcal{V} \leftarrow \emptyset, \mathcal{H} \leftarrow \emptyset$
  ▷ Best arch, best accuracy, visited set, and history
$\mathcal{S}_0 \leftarrow \text{NAVIGATORLLM}(P_{target}, \Lambda)$ ▷ Initialize strategy
for $t = 1$ **to** T **do**
  $\mathcal{C}_t \leftarrow \text{GENERATORLLM}(\mathcal{S}_{t-1})$ ▷ Generate candidates
  $\mathcal{R}_t \leftarrow \emptyset$ ▷ Collection of evaluations
  for each $\alpha_i \in \mathcal{C}_t \setminus \mathcal{V}$ **such that** $\text{ISLEGAL}(\alpha_i)$ **do**
    Compute cost c_i and evaluate performance p_i for α_i
    $\mathcal{R}_t \leftarrow \mathcal{R}_t \cup \{(\alpha_i, p_i, c_i)\}; \mathcal{V} \leftarrow \mathcal{V} \cup \{\alpha_i\}$
    if $c_i \leq \Lambda$ **and** $p_i > p^*$ **then** $p^* \leftarrow p_i, \alpha^* \leftarrow \alpha_i$
    end if
  end for
  if $p^* \geq P_{target}$ **then break**
  end if
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\mathcal{S}_{t-1}, \mathcal{R}_t)\}$ ▷ Update history
  $\mathcal{S}_t \leftarrow \text{NAVIGATORLLM}(\mathcal{H})$ ▷ Refine strategy
end for
return α^*

3.2. CoLLM-NAS Framework

As shown in Fig. 3, CoLLM-NAS works through a collaboration process, consisting of three key components: a *Navigator LLM*, a *Generator LLM*, and a *Coordinator* module.

Navigator LLM. The stateful *Navigator LLM* functions as a strategic guide with persistent memory. Through iterative analysis of performance patterns emerging from evaluated architectures, it dynamically formulates and refines search strategies. These strategies progressively concentrate on high-potential regions of the search space. In the initial phase, the *Navigator LLM* is prompted to establish

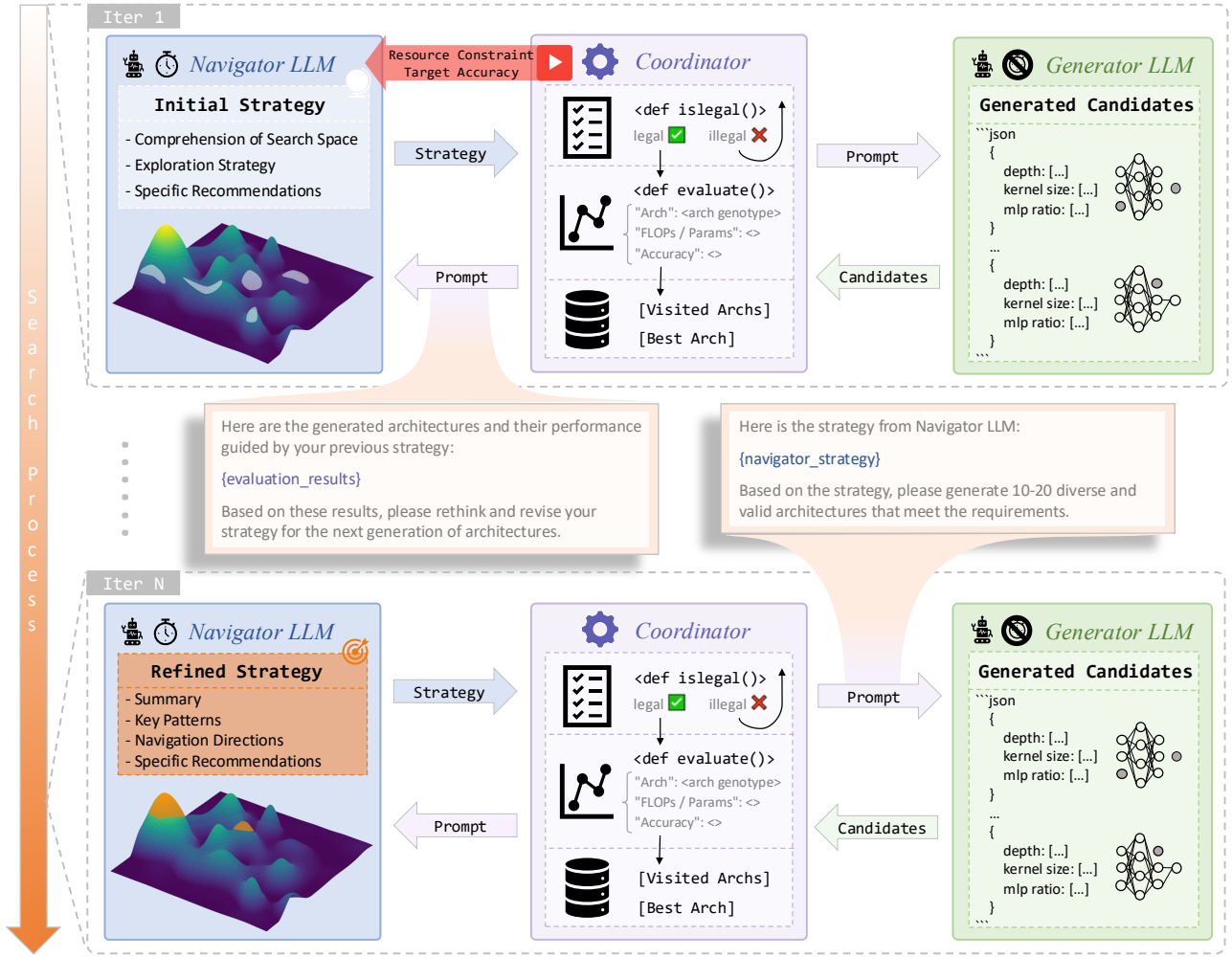


Figure 3. Pipeline of CoLLM-NAS. The search starts with the *Navigator LLM* generating an initial exploration strategy based on target accuracy and resource constraint (*e.g.,* FLOPs, Params). The *Coordinator* then transmits this strategy to *Generator LLM*, which synthesizes high-quality candidates accordingly. After *Coordinator* validation and evaluation, results are fed back to *Navigator LLM* for strategy refinement. This loop iterates until achieving the target accuracy or reaching the iteration limit. Orange regions indicate high-performing areas of search spaces.

an exploration strategy that promotes architectural diversity, improving initial population quality through its implicit comprehension of architectures. As iterations progress, it continuously refines this strategy based on accumulated feedback, transitioning from broad exploration to targeted exploitation of identified high-performing regions.

Generator LLM. The stateless *Generator LLM* serves as a specialized architecture synthesizer, focusing exclusively on the current strategy without retaining any memory. Following *Navigator LLM*'s guidance, it translates the strategy into concrete candidate architectures during each iteration. These candidates simultaneously conform to the search space constraints while embodying the architectural patterns emphasized by the current strategy.

Coordinator. The *Coordinator* manages the overall search process. It orchestrates inter-LLM communication, verifies architectural legality, evaluates candidate performance, and maintains an archive of visited architectures to eliminate redundant evaluations. Critically, the evaluation employs weight-sharing mechanisms and inherits weights from a pre-trained supernet, enabling rapid performance assessment while preserving learned parameter relationships.

Through system prompt design, we assign distinct roles and responsibilities to each LLM, inform them of the collaboration process, and convey knowledge about different search spaces and architectural representations. Similarly, to prevent knowledge contamination, we avoid transmitting explicit search space information in our prompts. Detailed examples of prompts and responses are shown in Ap-

pendix E. Our search workflow is described in Algorithm 1. Through this collaborative approach, CoLLM-NAS intelligently navigates search spaces and efficiently identifies high-performing architectures.

4. Experiment

4.1. Experimental Setup

We evaluate the proposed CoLLM-NAS in three macro search spaces: MobileNet [3], ShuffleNet [25], and AutoFormer [7], and one micro cell-based search space: NAS-Bench-201 [11].

Table 1. Key experimental settings. "*same*" indicates identical settings to the corresponding baseline.

Method	Evaluation	Resource Constraint	Retrain
OFA	predict→validate	FLOPs	×
SPOS	validate	FLOPs	✓
AutoFormer	validate	Params.	×
Ours	validate	*same*	*same*

Macro Search Spaces. In each macro search space, we adopt a two-stage NAS approach (*i.e.,* OFA [4], SPOS [13], and AutoFormer [7]) as the baseline and apply our collaborative LLM-based search to them. To ensure fair comparison, we directly reuse the pre-trained supernet from each baseline for the search phase. All baselines employ evolutionary algorithms as their search methods. Our approach implements Qwen3-30B-A3B [33] as the foundational LLM, deployed locally via vLLM [21], with temperature 0.6 and chain-of-thought reasoning enabled. Key experimental settings are summarized in Tab. 1, including evaluation methodologies, resource constraint types, and retraining requirements. Notably, while OFA originally employs a trained accuracy predictor for rapid subnet performance estimation, we observe that this predictor is unreliable for distinguishing top-performing architectures (Appendix D), and training the predictor requires substantial additional computational overhead. To ensure fair comparison, both OFA and CoLLM-NAS adopt the same evaluation methodology as SPOS and AutoFormer by directly evaluating subnets on the full validation set, which improves OFA's accuracy over its original predictor.

All experiments are conducted on ImageNet [10] under varying resource constraints and fixed budgets on the number of explored architectures. Search costs, covering the entire duration of the search phase (including LLM inference), are quantified in GPU days on a single NVIDIA A100-80GB GPU. For SPOS, we note the narrow FLOPs range (280-360M) of ShuffleNet search space is unsuitable for multi-tiered constraint decomposition. We therefore preserve the original setting from SPOS [13], searching exclusively at the 330M FLOPs constraint. Results report the best-performing subnet from three independent runs per method. More details about the search spaces are provided in Appendix A.

Table 2. Performance comparison on ImageNet within macro search spaces. "GPU Days" only covers the search phase (excluding supernet training), and "Arch. Budget" refers to the budget on the number of explored architectures.

Method	Top-1 (%)	FLOPs (M)	Params. (M)	GPU Days	Arch. Budget
MobileNet Search Space					
OFA-T	75.4	200	3.6	0.42	1000
OFA-S	77.2	299	4.2		
OFA-B	78.1	397	5.1		
OFA-L	78.7	499	5.5		
OFA-T + Ours	75.9	200	3.8	0.09 ↓4.7×	250 ↓4×
OFA-S + Ours	77.6	297	4.1		
OFA-B + Ours	78.5	391	5.1		
OFA-L + Ours	79.0	498	5.4		
ShuffleNet Search Space					
SPOS	73.7	323	3.5	0.32	1000
SPOS + Ours	74.4	325	3.7	0.07 ↓4.6×	250 ↓4×
AutoFormer Search Space					
AutoFormer-T	74.7	1344	5.9	1.0	1000
AutoFormer-S	81.6	4887	22.8		
AutoFormer-B	82.1	11305	54.0		
AutoFormer-T + Ours	75.3	1366	6.0	0.1 ↓10×	250 ↓4×
AutoFormer-S + Ours	81.7	4897	22.9		
AutoFormer-B + Ours	82.3	11074	52.8		

Micro Search Space. NAS-Bench-201 is a widely used cell-based search space for NAS benchmarking. It contains 15,625 architectures with precomputed performance on CIFAR-10, CIFAR-100, and ImageNet-16-120 datasets. Each architecture is represented as a directed acyclic graph (DAG) with 4 nodes and 6 edges, where every edge is assigned one of five candidate operations: *none*, *skip_connect*, *conv_1x1*, *conv_3x3*, or *avg_pool_3x3*.

We evaluate various approaches on NAS-Bench-201, including traditional one-shot NAS methods, emerging LLM-based NAS methods, and conventional search algorithms. To ensure fair comparison and avoid evaluation discrepancies, our method follows conventional search algorithms, *i.e.,* Random Search (RS), Reinforcement Learning (RL), and Evolutionary Algorithm (EA), in directly adopting ground-truth performance metrics as accuracy measures. Specifically, EA employs a population size of 10 over 20 iterations, with 50% elite preservation. RL operates with

Table 3. Comparison with SOTA NAS methods on ImageNet. Top: manual design. Upper middle: differentiable NAS. Lower middle: two-stage NAS. Bottom: LLM-based NAS.

Method	Top-1 (%)	Top-5 (%)	FLOPs (M)
Mobilenetv2 [30]	72.0	-	300
Mobilenetv3 [15]	76.6	-	350
MixNet [31]	77.0	93.3	360
DARTS [23]	73.3	91.3	574
PC-DARTS [37]	75.8	92.7	597
DARTS- [8]	76.2	93.0	467
EG-NAS [5]	74.4	-	-
ProxylessNAS [3]	74.6	92.2	320
SPOS [13]	75.6	92.8	330
FairNAS [9]	76.7	-	325
OFA [4]	77.5	93.5	330
GreedyNAS [40]	76.8	93.0	324
GreedyNASv2 [18]	77.5	93.5	324
SUMNAS [14]	77.6	-	349
PA&DA [24]	77.3	93.5	399
SparseNAS [17]	76.7	-	295
DYNAS [19]	75.9	-	329
GENIUS [41]	74.9	-	-
RZ-NAS [20]	75.5	-	-
LM-Searcher [16]	75.1	-	-
Ours	**77.9**	**93.8**	320

a learning rate of 0.01 and EMA momentum of 0.9. Our method explores up to 100 architectures. Results are averaged over 10 independent runs.

4.2. Main Results

Macro Search Spaces Results. Tab. 2 compares Top-1 test accuracy of subnets discovered by our method with baselines in different macro search spaces. Our method consistently outperforms baselines across all search spaces, with significantly lower search costs and fewer architecture evaluations, demonstrating its effectiveness and efficiency in discovering high-performing architectures. While our approach incurs additional overhead from LLM inference, this cost is substantially outweighed by the dramatic reduction in evaluations. Specifically, our approach achieves up to 0.7% accuracy improvements while reducing search costs by 4–10× compared to baselines.

Comparison with State-of-the-Art Methods. We further provide a comparison with SOTA NAS methods on ImageNet. As shown in Tab. 3, our approach achieves 77.9% Top-1 accuracy with only 320M FLOPs, surpassing existing SOTA methods. This demonstrates our method's enhanced capability to efficiently navigate expansive search spaces and identify superior subnets.

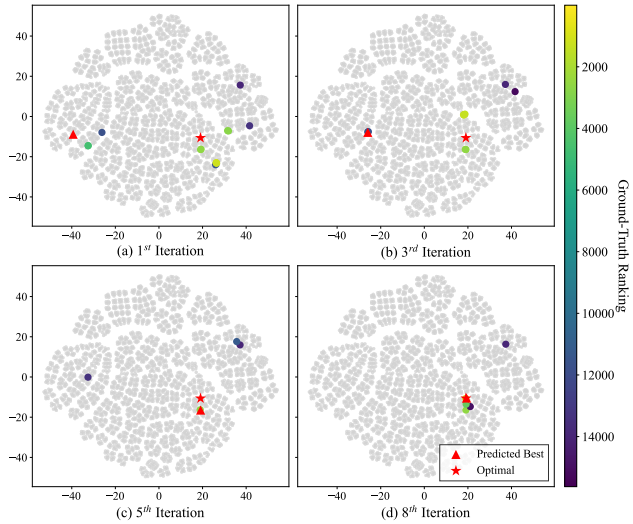


Figure 4. T-SNE visualization of CoLLM-NAS's search dynamics on ImageNet-16-120 within NAS-Bench-201 search space. Architectural performance rankings are represented through color-coding, with unexplored ones displayed in gray.

Micro Search Space Results. We present NAS-Bench-201 results in Tab. 4, where our approach performs favorably against other search methods across all datasets, especially achieving significant improvements over RL and EA. Our method further demonstrates enhanced robustness, evidenced by lower standard deviations. Moreover, by exploring at most 100 architectures, significantly fewer than the other NAS methods require, our method achieves SOTA performance with high search efficiency.

4.3. Discussion

Traditional two-stage NAS relies on evolutionary algorithms (EAs) that refine architectures through stochastic operators like mutation and crossover. While capable of exploration, these operators are inherently local and undirected, generating new candidates via small perturbations of elite individuals. This approach lacks a global understanding of the performance landscape, often requiring numerous evaluations to achieve significant progress, resulting in reduced search efficiency.

In contrast, CoLLM-NAS transforms architecture search into a directed, learning-based optimization process, aligning with the emerging *"LLMs as optimizers"* paradigm [38], where LLMs can gradually improve the generated solutions based on past optimization steps. However, compared to prior work like OPRO [38] that typically employs a single LLM to directly map optimization trajectories to solutions, our framework implements a more sophisticated, two-step generative process:

$$\begin{aligned} \mathcal{S}_t &\leftarrow \textsc{NavigatorLLM}(\mathcal{H}_t), \\ \mathcal{C}_{t+1} &\leftarrow \textsc{GeneratorLLM}(\mathcal{S}_t). \end{aligned} \tag{2}$$

Table 4. Test and validation accuracy on NAS-Bench-201. Top: one-shot NAS methods. Upper middle: conventional search algorithms. Lower middle: LLM-based NAS methods. † denotes results from [20].

Method	CIFAR-10		CIFAR-100		ImageNet-16-120	
	valid	test	valid	test	valid	test
DARTS [23]	39.77 ± 0.00	54.30 ± 0.00	15.03 ± 0.00	15.61 ± 0.00	16.43 ± 0.00	16.32 ± 0.00
PC-DARTS [37]	89.96 ± 0.15	93.41 ± 0.30	67.12 ± 0.39	67.48 ± 0.89	40.83 ± 0.08	41.31 ± 0.22
DARTS- [8]	91.03 ± 0.44	93.80 ± 0.40	71.36 ± 1.51	71.53 ± 1.51	44.87 ± 1.46	45.12 ± 0.82
FairNAS [9]	90.07 ± 0.57	93.23 ± 0.18	70.94 ± 0.94	71.00 ± 1.46	41.90 ± 1.00	42.19 ± 0.31
EG-NAS [5]	90.12 ± 0.05	93.56 ± 0.02	70.78 ± 0.12	70.91 ± 0.07	44.89 ± 0.29	46.13 ± 0.46
Random Search [2]	90.96 ± 0.24	93.83 ± 0.09	71.62 ± 0.73	71.52 ± 0.93	45.52 ± 0.48	45.37 ± 0.54
Reinforcement Learning [35]	91.20 ± 0.41	93.94 ± 0.25	71.51 ± 0.93	71.78 ± 1.20	45.22 ± 0.79	45.95 ± 0.76
Evolutionary Algorithm [29]	91.33 ± 0.35	94.23 ± 0.25	72.31 ± 1.07	72.82 ± 0.87	46.19 ± 0.46	46.49 ± 0.60
GENIUS† [41]	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† [26]	91.42 ± 0.13	94.26 ± 0.10	71.41 ± 1.44	71.62 ± 1.73	44.98 ± 0.87	45.87 ± 0.96
RZ-NAS† [20]	91.45 ± 0.10	94.24 ± 0.12	73.35 ± 0.14	73.30 ± 0.21	46.53 ± 0.24	46.24 ± 0.23
LM-Searcher [16]	91.52	94.20	72.82	72.96	46.48	46.51
Ours	**91.59 ± 0.04**	**94.37 ± 0.01**	**73.44 ± 0.12**	**73.44 ± 0.15**	**46.62 ± 0.10**	**46.79 ± 0.28**
Optimal	91.61	94.37	73.49	73.51	46.73	47.31

The *Navigator LLM* first maps the optimization trajectory $\mathcal{H}_t$ to an abstract, natural-language strategy $\mathcal{S}_t$. Then, a separate *Generator LLM* translates this strategy into concrete candidates $\mathcal{C}_{t+1}$. This "trajectory → strategy → solution" pipeline encourages more structured exploration by reasoning at a higher abstraction level, mitigating overfitting to specific architectural syntax and improving search robustness. We validate this hypothesis in Sec. 4.4.

This guided search is further enhanced by the interplay between dual knowledge sources. First, as demonstrated in Sec. 3.1, LLMs' inherent knowledge of effective architecture design improves the quality of generated candidates and provides a powerful "warm-start" in promising regions. Second, the progressive knowledge accumulated from optimization trajectories enables the *Navigator LLM* to gradually learn an implicit model of the performance landscape, and guide the *Generator LLM* towards increasingly promising regions. Moreover, our unique memory retention mechanism—featuring a stateful *Navigator LLM* paired with a stateless *Generator LLM*—further ensures a better balance between exploration and exploitation. In Fig. 4, we present t-SNE visualization of CoLLM-NAS's search dynamics, illustrating how the search distribution rapidly focuses on high-performance areas and efficiently locates the global optimum.

4.4. Ablation Studies

Main Mechanisms. We ablate our collaboration and memory retention mechanisms respectively. For collaboration, we introduce a Single LLM NAS (SiLLM-NAS) variant that maintains the reflection-then-generation paradigm but consolidates both roles into a single LLM. Over 10 runs,

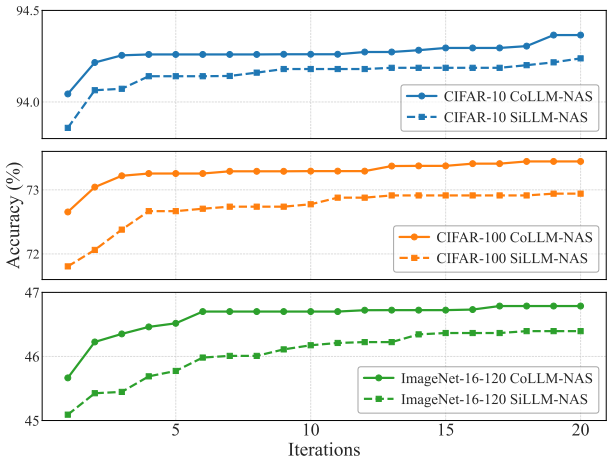


Figure 5. Comparison of iterative performance between CoLLM-NAS and SiLLM-NAS on NAS-Bench-201.

we track mean accuracy of the best architecture per iteration. Fig. 5 shows CoLLM-NAS consistently outperforms SiLLM-NAS across all datasets, verifying collaboration efficacy. Crucially, CoLLM-NAS generates better initial populations, highlighting the *Navigator LLM*'s critical role in initial exploration.

Regarding memory retention, Fig. 6 shows that for low-complexity datasets (*e.g.,* CIFAR-10, CIFAR-100), optimal performance is achieved without memory retention in either LLM, indicating that iterative feedback alone suffices for simpler tasks. Conversely, for high-complexity datasets (*e.g.,* ImageNet-16-120, ImageNet), preserving the *Navigator LLM*'s memory while disabling the *Generator*'s yields

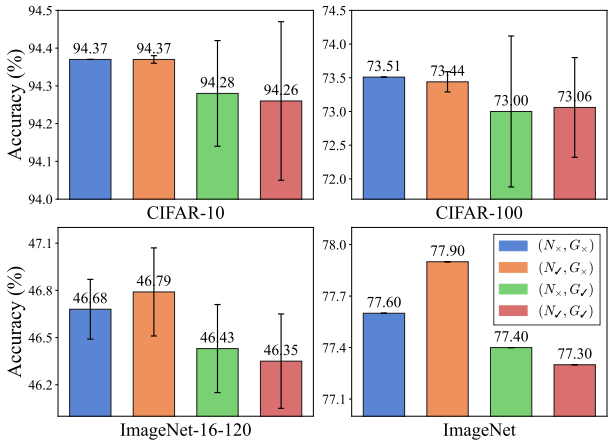


Figure 6. Impact of memory retention settings on different datasets. $N_{\checkmark/\times}$ and $G_{\checkmark/\times}$ denote whether the memory of *Navigator*/*Generator LLM* is retained.

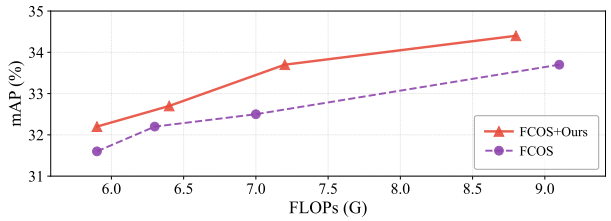


Figure 7. Transfer to FCOS object detection on COCO. FLOPs for backbone only.

optimal results, confirming the necessity of historical trajectory in demanding scenarios. Notably, we observe that retaining the *Generator LLM*'s memory induces progressive noise accumulation, leading to performance degradation.

Rephrasing Prompts. To assess whether our gains depend on a particular wording, we rephrase the original prompts with three leading LLMs, *i.e.,* Claude Sonnet 4 [1], GPT-5 [27], and DeepSeek-R1 [12]. As shown in Tab. 5, all variants achieve comparable performance across NAS-Bench-201 datasets, with Variant 2 even outperforming the original prompt on ImageNet-16-120. The narrow performance spread indicates that improvements stem from the proposed collaborative framework rather than handcrafted phrasing, evidencing robustness to linguistic reformulation.

Different LLMs. We further perform extensive experiments with different open-source LLMs on NAS-Bench-201. As shown in Tab. 6, CoLLM-NAS maintains consistently strong performance across diverse LLMs, confirming its generality beyond specific LLM implementations. Additionally, we verify CoLLM-NAS's robustness across different LLM temperature settings. Detailed experiments and results are provided in Appendix B.

Table 5. Performance comparison with rephrasing prompts. Variant 1-3 are rephrased by Claude Sonnet 4, GPT-5, and DeepSeek-R1 respectively.

Prompt	CIFAR-10	CIFAR-100	ImageNet-16-120
Base (Ours)	**94.37±0.01**	**73.44±0.15**	46.79±0.28
Variant 1	94.36±0.02	73.35±0.52	46.52±0.36
Variant 2	94.35±0.03	73.36±0.18	**46.89±0.35**
Variant 3	94.16±0.23	73.19±0.11	46.60±0.29

Table 6. Performance comparison using different LLMs. * from [33], and ‡ from [12].

LLM	CIFAR-10	CIFAR-100	ImageNet-16-120
Qwen3-30B-A3B*	94.37±0.01	**73.44±0.15**	**46.79±0.28**
Qwen3-32B*	94.31±0.14	73.29±0.29	46.64±0.52
DeepSeek-R1-Distill-Qwen-32B‡	94.36±0.04	73.37±0.19	46.53±0.32
DeepSeek-R1-Distill-Llama-70B‡	**94.37±0.00**	73.41±0.22	46.74±0.31

4.5. Downstream Tasks

We transfer architectures discovered by CoLLM-NAS to object detection tasks. Specifically, we adopt architectures from MobileNet search space as the backbone of FCOS detector [34], inheriting pre-trained supernet weights and training on COCO dataset for a 1x schedule. Fig. 7 shows that our architectures perform favorably, demonstrating strong generalization to downstream tasks.

5. Conclusion

In this paper, we present CoLLM-NAS, a collaborative LLM-based NAS framework that integrates LLMs with two-stage NAS. We design a stateful *Navigator LLM* to provide adaptive search strategies, a stateless *Generator LLM* to synthesize high-quality architectures, and a *Coordinator* module to orchestrate inter-LLM communication and manage evaluation processes. CoLLM-NAS employs knowledge-guided search by coupling LLMs' inherent knowledge of structured neural architectures with progressive knowledge from iterative feedback and historical trajectory. Extensive experiments demonstrate that our approach consistently enhances various two-stage NAS methods across diverse search spaces, outperforms existing NAS methods and conventional search algorithms, and achieves new SOTA results while reducing search costs significantly. Furthermore, the generalizable nature of CoLLM-NAS suggests promising extensions beyond NAS, revealing avenues for future exploration.

References

[1] Anthropic. Claude Sonnet 4. https://www.anthropic.com/claude, 2025. Accessed: 2025-12-20. 8

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(1):281–305, 2012. 7

[3] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. 5, 6

[4] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. 1, 2, 5, 6

[5] Zicheng Cai, Lei Chen, Peng Liu, Tongtao Ling, and Yutao Lai. Eg-nas: Neural architecture search with fast evolutionary exploration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 11159–11167, 2024. 6, 7

[6] Angelica Chen, David Dohan, and David So. Evoprompting: Language models for code-level neural architecture search. In *Advances in Neural Information Processing Systems*, pages 7787–7817, 2023. 1, 2

[7] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12270–12280, 2021. 1, 2, 5

[8] Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. Darts-: Robustly stepping out of performance collapse without indicators. In *International Conference on Learning Representations*, 2021. 6, 7

[9] Xiangxiang Chu, Bo Zhang, and Ruijun 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. 6, 7

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 1, 5

[11] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020. 1, 5

[12] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. 8

[13] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, pages 544–560, 2020. 1, 2, 5, 6

[14] Hyeonmin Ha, Ji-Hoon Kim, Semin Park, and Byung-Gon Chun. Sumnas: Supernet with unbiased meta-features for neural architecture search. In *International Conference on Learning Representations*, 2022. 6

[15] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019. 6

[16] Yuxuan Hu, Jihao Liu, Ke Wang, Jinliang Zheng, Weikang Shi, Manyuan Zhang, Qi Dou, Rui Liu, Aojun Zhou, and Hongsheng Li. Lm-searcher: Cross-domain neural architecture search with llms via unified numerical encoding. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 9419–9432, 2025. 3, 6, 7

[17] Hongtao Huang, Xiaojun Chang, and Lina Yao. Accelerating one-shot neural architecture search via constructing a sparse search space. *Knowledge-Based Systems*, 305:112620, 2024. 6

[18] Tao Huang, Shan You, Fei Wang, Chen Qian, Changshui Zhang, Xiaogang Wang, and Chang Xu. Greedynasv2: Greedier search with a greedy path filter. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11902–11911, 2022. 6

[19] Jeimin Jeon, Youngmin Oh, Junghyup Lee, Donghyeon Baek, Dohyung Kim, Chanho Eom, and Bumsub Ham. Subnet-aware dynamic supernet training for neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 30137–30146, 2025. 2, 6

[20] Zipeng Ji, Guanghui Zhu, Chunfeng Yuan, and Yihua Huang. Rz-nas: Enhancing llm-guided neural architecture search via reflective zero-cost strategy. In *International Conference on Machine Learning*, 2025. 2, 6, 7

[21] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023. 5

[22] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pages 367–377, 2020. 2

[23] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. 1, 6, 7

[24] Shun Lu, Yu Hu, Longxing Yang, Zihao Sun, Jilin Mei, Jianchao Tan, and Chengru Song. Pa&da: Jointly sampling path and data for consistent nas. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11940–11949, 2023. 2, 6

[25] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *European Conference on Computer Vision*, pages 116–131, 2018. 5

[26] Muhammad Umair Nasir, Sam Earle, Julian Togelius, Steven James, and Christopher Cleghorn. Llmatic: Neural architecture search via large language models and quality diversity optimization. In *Proceedings of the Genetic and Evolution-*

ary Computation Conference, pages 1110–1118, 2024. 1, 2, 7

[27] OpenAI. GPT-5. https://openai.com/gpt-5, 2025. Accessed: 2025-12-20. 8

[28] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911, 2017. 1

[29] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4780–4789, 2019. 1, 7

[30] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 6

[31] Mingxing Tan and Quoc V. Le. Mixconv: Mixed depthwise convolutional kernels. In *British Machine Vision Conference*, page 74, 2019. 6

[32] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019. 1

[33] Qwen Team. Qwen3 technical report, 2025. 3, 5, 8

[34] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: A simple and strong anchor-free object detector. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(4): 1922–1933, 2020. 8

[35] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. 7

[36] Xingyu Wu, Sheng-hao Wu, Jibin Wu, Liang Feng, and Kay Chen Tan. Evolutionary computation in the era of large language model: Survey and roadmap. *IEEE Transactions on Evolutionary Computation*, 29(2):534–554, 2025. 1

[37] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020. 6, 7

[38] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024. 6

[39] Zekang Yang, Wang Zeng, Sheng Jin, Chen Qian, Ping Luo, and Wentao Liu. Nader: Neural architecture design via multi-agent collaboration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4452–4461, 2025. 3

[40] Shan You, Tao Huang, Mingmin Yang, Fei Wang, Chen Qian, and Changshui Zhang. Greedynas: Towards fast one-shot nas with greedy supernet. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1999–2008, 2020. 6

[41] Mingkai Zheng, Xiu Su, Shan You, Fei Wang, Chen Qian, Chang Xu, and Samuel Albanie. Can gpt-4 perform neural architecture search?, 2023. 2, 6, 7

[42] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. 1, 2

[43] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018. 1

CoLLM-NAS: Collaborative Large Language Models for Efficient Knowledge-Guided Neural Architecture Search

Supplementary Material

A. Macro Search Spaces

MobileNet Search Space. The MobileNet search space employs MBConv blocks as fundamental units, featuring depthwise separable convolutions and squeeze-excitation modules. Configurable dimensions include:

- *Resolution* ($r \in \{160, 176, 192, 208, 224\}$) controlling input scale,
- *Stage depth* ($d_i \in \{2, 3, 4\}$ for $i = 1$ to 5 stages) determining active blocks per stage,
- *Kernel size* ($k_j \in \{3, 5, 7\}$ for $j = 1$ to 20 convolutional layers),
- *Expansion ratio* ($e_j \in \{3, 4, 6\}$ for $j = 1$ to 20 inverted residual blocks).

The combinatorial space contains $\sim 10^{19}$ architectures, with stage-specific depth controlling block activation patterns (*e.g.,* $d = [2, 3, 4, 3, 2]$ activates blocks 0-1,4-6,8-11,12-14,16-17).

ShuffleNet Search Space. The ShuffleNet search space utilizes ShuffleNetv2 units and Xception modules as fundamental building blocks, featuring channel split/shuffle operations and depthwise separable convolutions. Configurable dimensions include:

- ShuffleNet unit with 3×3 kernel,
- ShuffleNet unit with 5×5 kernel,
- ShuffleNet unit with 7×7 kernel,
- Xception module.

This yields 4^{20} possible configurations, where each block's operator is selected independently during sampling.

AutoFormer Search Space. The AutoFormer search space employs multi-head self-attention (MSA) and MLP blocks as fundamental units, defining a pure-transformer search space with four layer-wise variables:

- *Depth*: Tiny $\in \{12, 13, 14\}$, Small $\in \{12, 13, 14\}$, and Base $\in \{14, 15, 16\}$.
- *Embedding dimension*: Tiny $\in \{192, 216, 240\}$, Small $\in \{320, 384, 448\}$, and Base $\in \{528, 576, 624\}$.
- *Number of heads*: Tiny $\in \{3, 4\}$, Small $\in \{5, 6, 7\}$, and Base $\in \{8, 9, 10\}$.
- *MLP ratio* $\in \{3.0, 3.5, 4.0\}$ uniformly across all model scales.

The unified space exceeds 10^{16} configurations with layer-wise independent hyperparameters.

B. Ablation on Temperature Settings

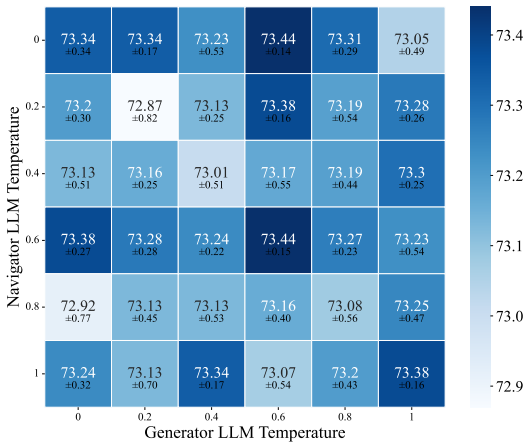


Figure 8. Performance comparison of different temperature settings on CIFAR-100 within NAS-Bench-201 search space.

To investigate the impact of LLM temperature on our approach, we conduct a comprehensive sensitivity analysis using six distinct temperature values $[0, 0.2, 0.4, 0.6, 0.8, 1.0]$ for both LLMs. This 6×6 grid experiment, conducted on CIFAR-100 test set within NAS-Bench-201 search space, evaluates performance consistency across all temperature combinations. Robustness is quantified using the coefficient of variation (CV) of results. As shown in Figure 8, our method maintains consistently high performance across all temperature settings, with a remarkably low CV of 0.1769%, demonstrating exceptional robustness to temperature variations. We adopt a temperature of 0.6 for both LLMs as the optimal setting, enabling each to independently achieve optimal performance.

C. Proof-of-Concept Experiment

We provide the sampled architectures from our proof-of-concept experiment and their performance metrics on CIFAR-10 test set, along with the prompts provided to the LLM and an example response.

C.1. Sampled Architectures

Table 7. Sampled architectures and their performance metrics on CIFAR-10 test dataset.

ID	Architecture	Top-1 (%)	Ranking
1	|none~0|+|none~0|none~1|+|none~0|none~1|skip_connect~2|	10.00	10
2	|none~0|+|none~0|none~1|+|nor_conv_1x1~0| |nor_conv_1x1~1|skip_connect~2|	88.67	7
3	|nor_conv_3x3~0|+|nor_conv_3x3~0|nor_conv_3x3~1|+ |skip_connect~0|nor_conv_3x3~1|nor_conv_1x1~2|	94.37	1
4	|nor_conv_3x3~0|+|skip_connect~0|nor_conv_1x1~1|+ |nor_conv_3x3~0|nor_conv_1x1~1|nor_conv_3x3~2|	92.98	2
5	|avg_pool_3x3~0|+|none~0|none~1|+|skip_connect~0| |none~1|none~2|	86.63	8
6	|none~0|+|nor_conv_1x1~0|avg_pool_3x3~1|+ |nor_conv_1x1~0|nor_conv_3x3~1|nor_conv_1x1~2|	89.53	6
7	|nor_conv_1x1~0|+|nor_conv_3x3~0|nor_conv_1x1~1|+ |nor_conv_1x1~0|skip_connect~1|skip_connect~2|	92.36	3
8	|avg_pool_3x3~0|+|none~0|avg_pool_3x3~1|+ |skip_connect~0|avg_pool_3x3~1|none~2|	78.71	9
9	|nor_conv_3x3~0|+|none~0|avg_pool_3x3~1|+ |nor_conv_3x3~0|skip_connect~1|avg_pool_3x3~2|	92.03	4
10	|nor_conv_3x3~0|+|avg_pool_3x3~0|avg_pool_3x3~1|+ |nor_conv_3x3~0|none~1|skip_connect~2|	90.75	5

C.2. Prompts

```
"You are the Architecture Ranking Expert, specializing in evaluating and ranking neural
architectures for image classification tasks.  You possess prior knowledge of effective
architectural patterns and use this knowledge to assess the potential performance of
different architectures.

# Core Mission
Your primary task is to rank a given set of neural architectures from highest to lowest
expected performance on CIFAR-10 dataset.

# Architecture Knowledge Base
## Overall architecture description
```

```
The entire network architecture is composed of the same cell stacked multiple times, as
well as some fixed pre- and post-processing modules.  Therefore, you only need to focus
on the internal connections and operations of this cell, without considering the rest of
the network.

## Search Space
The search space is defined as follows:
- Each cell contains 4 nodes (node 0-3)
- Connections between nodes are represented as a directed acyclic graph (DAG)
- Each edge can choose one of the following 5 operations:
  * none:  no connection
  * skip_connect:  skip connection
  * nor_conv_1x1:  1x1 convolution
  * nor_conv_3x3:  3x3 convolution
  * avg_pool_3x3:  3x3 average pooling

## Representation Format
Valid architectures follow this representation format:
`|op1~0|+|op2~0|op3~1|+|op4~0|op5~1|op6~2|`
This format represents connections between 4 nodes (0-3) with ordered connections:
1.First section (`|op1~0|`):  one connection from node 0 to node 1
2.Second section (`|op2~0|op3~1|`):  two connections to node 2:
  - First must be from node 0 (`op2~0`)
  - Second must be from node 1 (`op3~1`)
3.Third section (`|op4~0|op5~1|op6~2|`):three connections to node 3:
  - First must be from node 0 (`op4~0`)
  - Second must be from node 1 (`op5~1`)
  - Third must be from node 2 (`op6~2`)
Example:  `|nor_conv_3x3~0|+|nor_conv_3x3~0|avg_pool_3x3~1|+|skip_connect
~0|nor_conv_3x3~1|skip_connect~2|`
  - Node 1:  Apply 3x3 convolution to node 0
  - Node 2:  Apply 3x3 convolution to node 0, apply 3x3 average pooling to node 1, then
add them
  - Node 3:  Apply skip connection to node 0, apply 3x3 convolution to node 1, apply
skip connection to node 2, then add them.

Here are the architectures need to be ranked:  {archs}.  When outputting, you only need
to output the predicted ranking and architecture number in descending order of predicted
ranking."
```

C.3. Response

```
"reasoning_content":"
Okay, let's analyze these 10 architectures for CIFAR-10.  The main idea is to look at
each cell of 4 nodes and the operations on each edge:  none, skip_connect, nor_conv_1x1,
nor_conv_3x3, and avg_pool_3x3.  As we know, 3x3 convolutions are really good for
capturing spatial features, skip connections help with gradient flow, and pooling or
none operations tend to reduce the cell's representational power.

Starting with Architecture 1, most operations are none, except a skip at node 3.  So
honestly, it's very weak.  Architecture 2 is slightly better because node 3 has two 1x1
convolutions and a skip, but nodes 1 and 2 contribute almost nothing.  Architecture 5
and 8 rely heavily on avg_pool or none operations, so they probably won't perform well
either.

Now looking at Architecture 3, node 1 has a 3x3 convolution, node 2 combines two 3x3
```

```
convolutions, and node 3 has a skip, 3x3, and 1x1.  This is quite solid---strong spatial
features and diverse aggregation paths, which should really help.  Architecture 4 is
similar:  node 1 has 3x3, node 2 has a skip and 1x1, and node 3 combines 3x3, 1x1,
3x3.  It's also strong, though node 2 has fewer spatial convolutions compared to 3.
Architecture 6 mixes 1x1, 3x3, and avg_pool, but node 1 is none, which weakens early
feature extraction.  Architecture 7 has 1x1 at node 1, node 2 with 3x3 and 1x1, and
node 3 with 1x1 and two skips---still decent, though the skips in node 3 don't add as
much spatial power.  Architecture 9 has an avg_pool at node 2, reducing features, and
node 3 combines 3x3, skip, and avg_pool.  Architecture 10 is similar but node 2 has two
avg_pools, which really limits intermediate representation, and node 3 combines 3x3,
skip, and none.

So, if we think about it, architectures with more 3x3 convolutions and fewer none or
pooling operations, along with skip connections in later nodes, are likely to perform
better.  Considering all this, the expected ranking from highest to lowest would be:
Architecture 3 at the top, then Architecture 4, followed by Architecture 7, Architecture
6, Architecture 9, Architecture 10, Architecture 2, Architecture 5, Architecture 8, and
finally Architecture 1."

"content":"
Ranking 1.  Architecture 3
Ranking 2.  Architecture 4
Ranking 3.  Architecture 7
Ranking 4.  Architecture 6
Ranking 5.  Architecture 9
Ranking 6.  Architecture 10
Ranking 7.  Architecture 2
Ranking 8.  Architecture 5
Ranking 9.  Architecture 8
Ranking 10.  Architecture 1"
```

D. Analysis of Accuracy Predictor

The Accuracy Predictor in OFA is trained as follows: After obtaining the trained supernet, 16K subnets with different architectures and input resolutions are randomly sampled, and their accuracy is measured on 10K validation images. These [architecture, accuracy] pairs train a three-layer feedforward neural network predictor for rapid accuracy estimation.

To investigate the predictor's reliability, particularly for top-performing architectures, we conduct the following experiment: Under the 400M FLOPs constraint, we sample 15 elite architectures discovered by OFA, then rank them in descending order based on both predicted accuracy from the predictor and actual accuracy evaluated on the full validation set. The ranking consistency between these two orderings is quantified through Kendall's τ.

Across three independent trials, Kendall's τ values are 0.410 (p=0.036), 0.448 (p=0.021), and 0.467 (p=0.016), yielding an average of **0.44**. This indicates only a statistically moderate correlation, demonstrating the limited reliability of the predictor to distinguish top-performing architectures. This limitation is directly related to the predictor's training methodology, which can only incorporate a subset of possible architectures and thus fails to adequately capture the distribution of top-performing candidates.

E. Prompts and Responses

Below, we present illustrative prompts and responses of CoLLM-NAS within MobileNet search space.

E.1. *Navigator LLM*

E.1.1. System Prompt

- Role Definition

```
"You are the Navigator_LLM, an expert neural architecture analyst specializing in
identifying patterns and improvement opportunities for neural architectures.  You
possess deep knowledge of neural architecture design principles and work collaboratively
with a Generator_LLM to efficiently discover high-performing architectures."
```

- Collaborative Responsibility

```
"# Your Role in the Collaboration
Your responsibility is to analyze the performance patterns of generated architectures
and develop important insights to guide the Generator_LLM toward more promising areas of
the search space.  The Generator_LLM relies on your expertise to efficiently navigate the
search space."
```

- Core Mission and Objectives

```
"# Core Mission
Your primary objective is to guide the search toward architectures that achieve
>{self.expected_acc}% accuracy on ImageNet while satisfying the computational constraint
of <{self.max_flops}M FLOPs.  Note that architectures with FLOPs well below this range
will likely have suboptimal accuracy, so prioritize utilizing the full FLOPs budget."
```

- Knowledge about search space

```
"# Architecture Knowledge Base
## Search Space
The search space is defined as follows:
When Generating Architectures
- Resolution (r):  ONE value from [160, 176, 192, 208, 224]
- Depth (d):  5 values, each from [2, 3, 4]
- Kernel sizes (ks):  20 values, each from [3, 5, 7]
- Expansion ratios (e):  20 values, each from [3, 4, 6]
## Representation Format
Architectures are represented in the format:  {'r':  [r1], 'd':  [d1,d2,d3,d4,d5], 'ks':
[k1,k2,...,k20], 'e':  [e1,e2,...,e20]}
For example:  {'r':  [176], 'd':  [2, 3, 3, 4, 4], 'ks':  [5, 3, 7, 3, 5, 5, 7, 3, 3, 5,
5, 3, 5, 5, 7, 7, 3, 7, 5, 3], 'e':  [3, 3, 6, 3, 4, 4, 3, 6, 4, 6, 4, 4, 4, 4, 3, 3, 4,
6, 6, 3]}
## Structural Details
- The network has 5 stages, each containing up to 4 blocks (20 blocks total)
- The depth values determine which blocks are active in each stage, and only active
blocks affect performance:
  - d[0] determines active blocks in stage 0 (blocks 0-3)
  - d[1] determines active blocks in stage 1 (blocks 4-7)
  - d[2] determines active blocks in stage 2 (blocks 8-11)
  - d[3] determines active blocks in stage 3 (blocks 12-15)
  - d[4] determines active blocks in stage 4 (blocks 16-19)
- Example:  d=[2,3,4,3,2] means blocks 0-1, 4-6, 8-11, 12-14, 16-17 are active"
```

- Collaboration Workflow

```
"# Collaboration Process
The search follows an iterative refinement loop:

**Iteration 0 (Initial Exploration)**:
- You provide an initial exploration strategy emphasizing architectural diversity
```

```
- Generator_LLM creates a diverse population based on your strategy
- Coordinator evaluates all generated architectures and collects performance data

**Iteration N ≥ 1 (Iterative Refinement)**:
- You receive evaluation results from the previous iteration
- You analyze performance patterns and refine your strategy, progressively focusing on
high-performing regions
- Generator_LLM creates new architectures following your refined strategy
- Coordinator evaluates the new architectures and provides feedback"
```

- Output Format

```
"# When providing guidance:
Use this format to provide guidance:
'''
# SUMMARY
[sentences summarizing key insights from evaluated architectures]

# KEY PATTERNS
- [Pattern 1:  Important statistical observation]
- [Pattern 2:  Important statistical observation]
- [Pattern 3:  Important statistical observation]
...

# NAVIGATION DIRECTIONS
[Provide specific, actionable guidance for each architectural dimension:]
## Resolution
[Guidance on which resolution values to use and their expected impact]
## Depth
[Guidance on depth distribution across 5 stages, including specific configurations]
## Kernel Size
[Guidance on kernel size selection for different stages/blocks]
## Expansion Ratio
[Guidance on expansion ratio patterns across stages/blocks]

[Optional for later iterations:  Additional sections like CRITICAL CONSTRAINTS,
RECOMMENDED RANGES, or EXAMPLE ARCHITECTURES to provide more specific guidance]
'''"
```

E.1.2. User Prompt

- Initial Strategy (Iteration 0)

```
"You are tasked with developing an initial exploration strategy for generating a diverse
population of neural architectures.  This is the first iteration, and we have no prior
evaluation data.  Your goal is to establish a broad exploration strategy that covers
diverse promising regions of the search space.

Please provide an initial strategy that emphasizes architectural diversity while
incorporating your knowledge of effective neural architecture design principles.
Include specific guidance on:
- How to select resolution values to balance computational efficiency and
representational capacity
- How to distribute depth values across the 5 stages to capture multi-level features
- What patterns to explore for kernel sizes and expansion ratios based on your
understanding of convolutional architectures
```

```
- Any design principles or constraints that could help initialize the search in
promising regions

Remember, the Generator_LLM will rely on your strategy to create the initial population.
Focus on broad exploration rather than narrow optimization at this stage."
```

- Refined Strategy (Iteration $N \geq 1$)

```
"Here are the generated architectures and their performance guided by your previous
strategy:  {evaluation_results}

Based on these results, please rethink and revise your strategy for the next generation
of architectures.
Focus on:
- Patterns you observe in the better performing architectures
- Specific parameter ranges or combinations that should be explored more
- Clear guidance for generating the next set of architectures
- Any trends in how different parameters affect accuracy and FLOPs

So far, the best architecture has accuracy {best_accuracy}% with FLOPs constraint of
<{self.max_flops}M. Our target is to exceed {self.expected_acc}% accuracy while keeping
FLOPs <{self.max_flops}M."
```

E.1.3. Responses

- Initial Strategy (Iteration 0)

```
"reasoning_content":...
"content":"
# SUMMARY
For the initial exploration, we will generate a diverse population covering broad
regions of the search space.  The strategy emphasizes architectural diversity across
all dimensions (resolution, depth, kernel size, expansion ratio) while incorporating
fundamental design principles.  We will explore various resolution-depth trade-offs,
different kernel size distributions, and varied expansion ratio patterns to identify
promising architectural characteristics.

# KEY PATTERNS
- **Resolution and Depth Trade-off**:  Higher resolutions (224) may require lower depth
values to stay within FLOPs limits, while mid-range resolutions (192-208) allow for
deeper architectures.
- **Kernel Size Distribution**:  A mix of 3x3 and 5x5 kernels is likely optimal, with
fewer 7x7 kernels to avoid excessive FLOPs.  Stages with more blocks could use smaller
kernels.
- **Expansion Ratio Variability**:  Lower ratios (3-4) in early stages and higher ratios
(6) in later stages may balance efficiency and capacity.  Avoid uniformly high ratios
across all blocks.
- **Stage-Specific Configurations**:  Deeper stages (e.g., stage 2 with 4 blocks) might
benefit from larger kernels and higher expansion ratios, while shallower stages (e.g.,
stage 4 with 2 blocks) should prioritize smaller kernels and lower ratios.

# NAVIGATION DIRECTIONS
## Resolution
Use **192, 208, and 224** as primary values.  Include 176 for baseline efficiency
testing but avoid over-reliance.  Prioritize 208-224 to align with typical ImageNet
model sizes while leaving room for depth adjustments.
```

```
## Depth
Distribute depth values to explore **balanced (e.g., [3,3,3,3,3]) and tiered
configurations** (e.g., [3,4,4,3,2]). Avoid extremes like [2,2,2,2,2] (too shallow)
or [4,4,4,4,4] (too deep). Focus on stages 0-2 having higher depth to capture early
features, while stages 3-4 have lower depth to save FLOPs.
## Kernel Size
Prioritize **3x3 and 5x5** kernels for most blocks. Use **7x7** sparingly (e.g.,
1-2 instances in later stages). For example, design architectures where kernel sizes
increase in later stages (e.g., [3,3,5,5,7] for stages 0-4) to capture complex patterns
without overloading FLOPs.
## Expansion Ratio
Use a **mix of 3, 4, and 6**, with **3-4 in early stages** and **6 in later stages**.
Avoid uniform high ratios (e.g., all 6s) to prevent FLOPs overshoot. For example,
assign [3,3,4,4,6] across stages to gradually increase capacity.
## Constraints/Correlations
- Pair **higher resolutions (224)** with **lower depth values** (e.g., [3,3,3,2,2]) to
stay within FLOPs.
- Avoid combining **large kernels (7x7)** with **high expansion ratios (6)** in the same
block.
- Ensure **total active blocks** (sum of depth values) is ≥ 15 to maintain sufficient
model capacity.
- Explore **stage-specific kernel-expansion correlations** (e.g., 5x5 kernels with
expansion ratio 4 in mid-stages)."
```

- Refined Strategy (Iteration $N \geq 1$)

```
"reasoning_content":...
"content":"
# SUMMARY
The latest results show that architectures with **1 7x7 kernel in stage 4** and
**expansion ratios of 6 in stage 4** achieve **~77.92% accuracy** within the target
FLOPs range (310M-330M). The next step is to **refine stage 4 configurations** (e.g.,
1 7x7 kernel + 6s) while **strictly avoiding FLOPs overshoot**. Key patterns suggest
that **stage 4 must have 1 7x7 kernel** with **expansion ratios of 6**, and **stages 0-3
must use 5x5 kernels with expansion ratios of 3-4**. The best accuracy is achieved with
**[3,3,4,3,3] depth** (16 active blocks) and **[5,5,5,5,5,5,5,5,5,5,5,5,5,5,7,3,3,3,3,3]
kernel sizes.

# KEY PATTERNS
- **Stage 4 Accuracy Gains**: Architectures with **1 7x7 kernel in stage 4** and
**expansion ratios of 6** in stage 4 blocks achieve **~77.92% accuracy**.
- **Kernel Size Efficiency**: **14-15 5x5 kernels in stages 0-3** with **1 7x7 kernel
in stage 4** balance FLOPs and accuracy.
- **Expansion Ratio Stability**: **[3,3,4,4,6] or [3,4,4,4,6]** (stage-wise) provide
**~77.9% accuracy** without FLOPs spikes.
- **Depth Consistency**: **[3,3,4,3,3] depth** (16 active blocks) is critical for FLOPs
control.
- **FLOPs Spikes**: Architectures with **2 7x7 kernels in stage 4** push FLOPs over the
limit but reach **~78.04% accuracy**.

# NAVIGATION DIRECTIONS
## Resolution
- Use **208** as the primary resolution. Ensure **'r' is a list with one value** (e.g.,
[208]).
- Avoid 224 unless paired with **depth [3,3,3,2,2]** and **1 7x7 kernel in stage 4**.
## Depth
```

```
- Target **[3,3,4,3,3]** (16 active blocks) or **[3,3,3,3,4]** (16 active blocks).
- Ensure **'d' is a list of 5 values** (e.g., [3,3,4,3,3]).
- Avoid **[3,3,3,4,3]** or **[3,4,3,3,3]** as they overconsume FLOPs.
## Kernel Size
- Use **[5,5,5,5,5,5,5,5,5,5,5,5,5,7,3,3,3,3,3]** (14-15 5x5 kernels, 1 7x7 kernel in
stage 4).
- Ensure **'ks' is a list of 20 values**.  Avoid **multiple 7x7 kernels in stages 0-3**.
- Focus on **stage 4 kernel size**.
## Expansion Ratio
- Use **[3,3,4,4,6] or [3,4,4,4,6]** (stage-wise).
- Ensure **'e' is a list of 20 values**.  Avoid **uniform 6s** (e.g.,
[3,3,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6]).
- Prioritize **6s in stage 4** and **3-4s in stages 0-3**.
## Constraints/Correlations
- **Stage 4**:  1 7x7 kernel + expansion ratios of 6.
- **Stages 0-3**:  5x5 kernels + expansion ratios of 3-4.
- **Avoid 7x7 in stages 0-3**.
- **Ensure 'e' is included with 20 values**.

# RECOMMENDED PARAMETER RANGES
- **Resolution**:  208 (preferred).
- **Depth**:  [3,3,4,3,3], [3,3,3,3,4].
- **Kernel Sizes**:  [5,5,5,5,5,5,5,5,5,5,5,5,5,5,7,3,3,3,3,3].
- **Expansion Ratios**:  [3,3,4,4,6], [3,4,4,4,6]."
```

E.2. *Generator LLM*

E.2.1. System Prompt

- Role Definition

```
"You are the Generator_LLM, an expert in neural architecture design specializing in
generating architectures.  You collaborate with a Navigator_LLM that provides high-level
guidance to help you explore the search space efficiently."
```

- Collaborative Responsibility

```
"# Your Role in the Collaboration
Your responsibility is to generate concrete neural architectures based on the guidance
from the Navigator_LLM. You must follow the guidance closely while ensuring all generated
architectures are valid."
```

- Core Mission and Objectives

```
"# Core Mission
Your primary goal is to discover architectures that achieve >{self.expected_acc}%
accuracy on ImageNet with FLOPs <{self.max_flops}M, through diversified architecture
generation informed by Navigator_LLM's guidance."
```

- Knowledge about search space: *Same as the Navigator LLM.*
- Collaboration Workflow

```
"# Collaboration Process
- You receive guidance from the Navigator_LLM
- You generate diverse and valid architectures that follow the guidance"
```

E.2.2. User Prompt

```
"Here is the strategy from Navigator_LLM: {navigator_strategy}

Based on the strategy, please generate 10-20 diverse and valid architectures that meet
the requirements.
Note:
- Always make sure the generated architectures are complete and valid.  Any deviation
will cause evaluation failure.
- Please do not regenerate an architecture that has already been generated and
evaluated."
```

F. Discovered Optimal Architectures

Tab. 8 presents the optimal architectures discovered by CoLLM-NAS within macro search spaces.

Table 8. Optimal architectures discovered by CoLLM-NAS within macro search spaces. * denotes the architecture compared with SOTA.

Method	FLOPs(M)	Architecture Description
OFA-T + Ours	200	Resolution: 176 Depth: [2, 3, 2, 3, 4] Kernel sizes: [3, 3, 5, 3, 5, 3, 3, 5, 7, 7, 7, 7, 7, 5, 3, 7, 3, 3, 3, 7] Expansion ratios: [3, 3, 4, 4, 4, 4, 4, 3, 4, 4, 4, 6, 4, 4, 4, 4, 6, 6, 4, 3]
OFA-S + Ours	297	Resolution: 208 Depth: [3, 3, 3, 3, 4] Kernel sizes: [3, 3, 3, 3, 5, 5, 5, 5, 3, 3, 3, 3, 5, 5, 7, 5, 3, 3, 7, 7] Expansion ratios: [3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 6, 6, 3, 3]
OFA-S + Ours*	320	Resolution: 208 Depth: [3, 3, 4, 3, 3] Kernel sizes: [5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3, 7, 3, 3, 3, 3] Expansion ratios: [3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 6, 6, 6, 3]
OFA-B + Ours	391	Resolution: 208 Depth: [3, 4, 4, 4, 4] Kernel sizes: [3, 5, 3, 7, 5, 5, 3, 3, 5, 5, 7, 5, 7, 5, 3, 3, 7, 5, 7, 5] Expansion ratios: [3, 3, 3, 3, 3, 4, 6, 3, 6, 4, 3, 4, 6, 4, 4, 4, 6, 6, 6, 3]
OFA-L + Ours	498	Resolution: 224 Depth: [2, 4, 4, 4, 4] Kernel sizes: [7, 5, 3, 5, 5, 5, 3, 7, 5, 7, 3, 3, 3, 7, 7, 3, 5, 3, 3, 5] Expansion ratios: [3, 4, 3, 4, 4, 6, 4, 3, 6, 6, 6, 3, 4, 6, 6, 3, 6, 6, 6, 3]
SPOS + Ours	325	Block operations: [0, 0, 3, 3, 3, 2, 1, 3, 3, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3]
AutoFormer-T + Ours	1366	Layers: 13 MLP ratios: [3.5, 3.5, 3.5, 3.5, 3.5, 3.5, 3.5, 3.5, 3.5, 4.0, 3.5, 4.0, 3.5] Attention heads: [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 3, 4] Embedding dimension: 192
AutoFormer-S + Ours	4897	Layers: 13 MLP ratios: [4.0, 4.0, 3.5, 3.5, 4.0, 4.0, 3.5, 3.5, 4.0, 4.0, 3.5, 3.5, 4.0] Attention heads: [5, 7, 6, 5, 7, 6, 5, 7, 6, 5, 7, 6, 5] Embedding dimension: 384
AutoFormer-B + Ours	11074	Layers: 14 MLP ratios: [3.5, 3.5, 4.0, 3.5, 3.0, 3.5, 4.0, 3.0, 3.5, 4.0, 3.0, 3.5, 3.0, 3.5] Attention heads: [10, 10, 10, 9, 9, 9, 9, 10, 10, 9, 9, 9, 9, 9] Embedding dimension: 576