CoDA-NAS: Pattern Knowledge-Based Cross-Task Neural Architecture Search with LLM Agents

Anonymous ACL submission

Abstract

Recent studies increasingly employ large language models (LLMs) for neural architecture search (NAS); however, they typically rely on in-context learning, struggle to turn past evaluations into accumulated knowledge, and often suffer from structural hallucinations. Motivated by the observation that a search process should systematically accumulate past experience to guide future exploration, we propose CoDA-NAS, a cooperative dual-agent framework that investigates natural language as a medium for structural knowledge. At the core of our framework is a pattern knowledge base (PKB) that extracts, validates, and maintains design patterns from evaluated architectures. These patterns are retrieved as soft priors to guide the searcher agent instead of memoryless generation. The searcher agent then employs a structured reasoning constraint to tightly align the generated topological outputs with these semantic concepts, thereby mitigating hallucinations. To ensure empirical reliability, an integrated lightweight predictor agent provides independent numerical validation. By applying a t-test, statistically verified knowledge enters the PKB, effectively curbing the spread of incorrect rules. Experimental results show that under strict search limits, our framework achieves competitive performance using only a local open-weights model. Furthermore, the extracted text-based patterns generalize well to new tasks, establishing natural language as a reliable and highly robust medium for transferring architectural knowledge.

1 Introduction

Large language models (LLMs) are increasingly being employed as autonomous agents for complex optimization tasks (Yao et al., 2023; Yang et al., 2024). Neural architecture search (NAS) is a representative example, as finding optimal network structures fundamentally requires searching through a massive and discrete search space (Zoph and Le, 2017; Elsken et al., 2019). This makes NAS a practical testbed to examine how well LLM-driven agents can reason and generalize. Recent studies often treat architectural components as linguistic tokens and use the prior knowledge inside LLMs to guide the search more effectively and efficiently (Chen et al., 2023; Zheng et al., 2023; Nasir et al., 2024).

Despite these efforts, there is still room for improvement, as many existing methods lack mechanisms to systematically accumulate and verify the knowledge gained during the search process. Many methods rely heavily on short-term in-context learning(Zheng et al., 2023; Ji et al., 2025; Nasir et al., 2024). As context windows are limited, models tend to forget past search steps, thereby failing to turn isolated trials into long-term and accumulated knowledge. Generative models also suffer from hallucinations and sometimes confidently suggest architectural rules that are mathematically unsound (Ji et al., 2023). A few recent studies attempt to extract textual design principles from past searches (Zhou et al., 2025), typically applying them as offline and rigid pruning constraints. Hard constraints like these can accidentally eliminate promising regions of the search space, which traps the optimization process in local optima.

To address these challenges, we present cooperative dual-agent NAS (CoDA-NAS), a framework that shifts the focus from memoryless trial-and-error to knowledge-grounded search. The core component is a pattern knowledge base (PKB) that translates architectural features into natural language patterns. The searcher agent uses retrieval-augmented generation (RAG) (Lewis et al., 2020) to retrieve relevant patterns from the PKB as contextual soft priors. Instead of acting as strict filters that permanently ban operations, these patterns are appended to the searcher's prompt so that the LLM can autonomously weigh historical sugges-

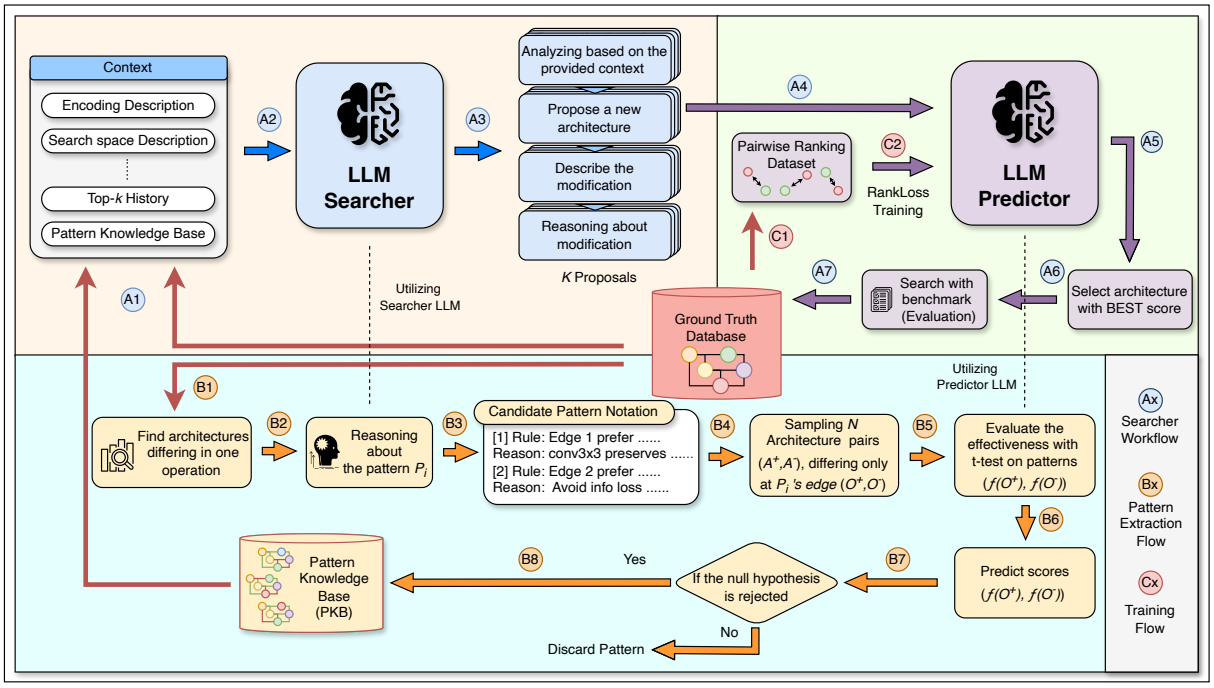

Figure 1: The CoDA-NAS framework. The searcher retrieves PKB patterns and generates candidates via structured reasoning; the predictor ranks them and validates new patterns before they enter the PKB.

tions without losing the freedom to explore new areas. To further prevent inference hallucinations, the searcher employs a structured generation flow that takes advantage of the autoregressive nature of LLMs. The model must state its reasoning and expected effects before generating the topological array, and describe its modification immediately after. This closed-loop sequence keeps the model focused within the massive search space and naturally maintains structural diversity during batch proposal generation.

Bridging LLM reasoning with objective numerical grounding is essential for reliability. To achieve this, we introduce a lightweight predictor agent: a language-driven surrogate model that provides transferable cross-task performance estimation while natively interpreting structural architecture encodings. Rather than remaining static, the predictor is continuously fine-tuned online during the search process and gradually adapts to the target task. As the searcher proposes new candidates, the predictor provides quick numerical feedback to filter out poor designs early. Beyond performance estimation, the predictor also plays a critical role in knowledge verification. Before any new textual pattern enters the PKB, the predictor evaluates the pattern's validity by comparing the predicted performance distribution of architectures with and without the pattern through a t-test. Only patterns showing a statistically significant advantage are admitted into the PKB, forming a self-correcting loop that effectively filters out noise.

Our evaluations demonstrate that CoDA-NAS can achieve convergence rates comparable to much heavier methods while running entirely on a local open-weights model. It avoids relying on closed-source and black-box LLMs, which often raise concerns about reproducibility and data privacy (Zheng et al., 2023; Ji et al., 2025). To rigorously evaluate this cross-task capability, we specifically benchmark CoDA-NAS on established computer vision datasets (e.g., NAS-Bench-201 (Dong and Yang, 2020) and TransNAS-Bench-101 (Duan et al., 2021)). These benchmarks provide absolute mathematical ground truth without retraining, offering an objective testbed to verify structural reasoning. Our main contributions are summarized below:

- **Pattern Knowledge Base:** We demonstrate that translating topological structures into natural language patterns effectively breaks the domain barrier for cross-task NAS.
- **Statistically Grounded Dual-Agent Framework:** By pairing a dynamically updated surrogate with rigorous statistical testing, CoDA-NAS systematically mitigates LLM hallucinations during text-based architectural search.
- **Empirical Superiority:** Our tests indicate

Method	LLM Model	Local	LLM Tuning	Surrogate	Retrieval
GENIUS (Zheng et al., 2023) (arXiv '23)	GPT-4	✗	✗	✗	✗
LLMatic (Nasir et al., 2024) (GECCO '24)	CodeGen-6.1B	✓	✗	✗	✗
RZ-NAS (Ji et al., 2025) (ICML '25)	GPT-4	✗	✗	✗	✗
LAPT (Zhou et al., 2025) (AAAI '25)	GPT-4	✗	✗	✗	✗
LM-Searcher (Hu et al., 2025) (EMNLP '25)	LLaMA-3.1-8B	✓	✓	✗	✗
CoDA-NAS (Ours)	**LLaMA-3.3-70B-Instruct**	✓	✗	✓	✓

Table 1: Comparison of key capabilities across LLM-driven NAS frameworks.

that treating RAG-based textual patterns as soft priors helps avoid search space collapse. Under strict evaluation limits, the local-model approach shows competitive efficiency while avoiding the reproducibility issues of closed APIs.

2 Related Work

2.1 Neural Architecture Search with LLMs

Conventional NAS methods often suffer from high computational cost due to repeated architecture evaluations, along with limited ability to discover transferable patterns within a search space (Elsken et al., 2019). LLMs address these challenges through semantic reasoning, offering a natural mechanism for structural transfer that is highly suited to the cross-task NAS challenges formalized by TransNAS-Bench-101 (Duan et al., 2021). Pioneers such as GPT-NAS (Yu et al., 2025) pre-train a GPT on architecture data, framing NAS as autoregressive sequence generation, while LLMatic (Nasir et al., 2024) employs LLMs as mutation operators within a quality-diversity (MAP-Elites) framework. LLMs, meanwhile, are also employed as linguistic coordinators in GENIUS (Zheng et al., 2023), RZ-NAS (Ji et al., 2025), and the concurrent trajectory-guided CoLLM-NAS (Li et al., 2026), which iteratively optimize architectures through prompting and self-reflection. LAPT (Zhou et al., 2025) further acts as an offline pre-processor that distills design principles from proxy datasets, enabling cross-task transfer of linguistic design patterns.

2.2 LLM as Performance Predictor

Extensive research has established the effectiveness of machine learning methods such as XGBoost or standard graph neural networks for estimating architecture performance (White et al., 2021). Neural Predictor (Wen et al., 2020) demonstrates strong in-domain correlations, thereby showing great promise in reducing the computational overhead for NAS. With the advancement of predictors, TransNAS-Bench-101 (Duan et al., 2021) further highlights the crucial demand for transferability in predictors and their promising prospects on diverse tasks, thus promoting the research on LLM-driven predictors. Recent studies have shown that LLMs demonstrate strong capability in cross-task transferability when serving as surrogates for NAS (Qin et al., 2025). For instance, LM-Searcher (Hu et al., 2025) predicts performance through numerical encoding and instruction-tuned ranking. While these methods advance the field, several fundamental challenges remain.

2.3 Summary

Based on the literature review in the LLM-driven NAS field, existing studies (summarized in Table 1) exhibit several critical pitfalls: (1) The greedy nature of optimization in GENIUS (Zheng et al., 2023) risks trapping the search in local optima. (2) Mutation operators in LLMatic (Nasir et al., 2024) restrict the model to local changes without reasoning about the overall architecture. (3) The reasoning process in RZ-NAS (Ji et al., 2025) and the concurrent work CoLLM-NAS (Li et al., 2026) rely on unverified proxies or trajectory memory. Without rigorous numerical verification, these approaches (1–3) struggle to accumulate reusable knowledge across tasks. (4) LAPT (Zhou et al., 2025) attempts to guide the search by partitioning the solution space via design principles, but this rigid approach severely restricts exploration and often causes optimal solutions to be inadvertently pruned during knowledge transfer. (5) LM-Searcher (Hu et al., 2025) encodes architectures as numerical sequences, ignoring the rich structural semantics provided by natural language, and requires expensive offline training while still struggling to achieve high precision on individual tasks.

Unlike methods that discard evaluations or lack statistical verification (1–3), the CoDA-NAS framework accumulates validated design patterns in a knowledge base and retrieves them as soft priors. This strategy preserves full search space cover-

age instead of rigid pruning (4). By grounding architectural knowledge in natural language and adapting online without costly offline training (5), CoDA-NAS enables effective knowledge-grounded search.

3 Methodology

3.1 Problem Formulation

Standard NAS over a discrete search space $\mathcal{A}$ is fundamentally a bilevel optimization problem (Liu et al., 2019). The goal is to find an optimal architecture $\mathbf{a}^*$ that minimizes the validation loss $\mathcal{L}_{val}$, subject to the network weights $\mathbf{w}$ being optimized on the training loss $\mathcal{L}_{train}$:

$$\begin{aligned} \mathbf{a}^* &= \arg\min_{\mathbf{a}\in\mathcal{A}} \mathcal{L}_{val}(\mathbf{a}, \mathbf{w}^*(\mathbf{a})), \\ \text{s.t.} \quad \mathbf{w}^*(\mathbf{a}) &= \arg\min_{\mathbf{w}} \mathcal{L}_{train}(\mathbf{a}, \mathbf{w}). \end{aligned} \tag{1}$$

In a cross-task scenario involving semantically different source ($\mathcal{T}_{src}$) and target ($\mathcal{T}_{tgt}$) domains, solving this objective (Eq. 1) from scratch is computationally expensive. Instead of discarding past evaluations, our framework extracts transferable structural rules from the search trajectory on $\mathcal{T}_{src}$ into a pattern knowledge base $\mathcal{P}$. We then reframe the target search by introducing an LLM-driven generation policy $\mathbf{a} \sim \pi(\cdot \mid \mathcal{P})$. This policy guides the exploration by treating $\mathcal{P}$ as a set of soft priors, rather than modifying the actual objective landscape. The main technical challenge is to ensure that this guided sampling speeds up convergence when the domains share similarities while using rigorous statistical validation to prevent negative transfer when the source priors clash with the target semantics.

3.2 CoDA-NAS

A practical challenge when using LLMs for iterative optimization is that each call is independent. The model retains no memory of what it proposed in the previous round or how that candidate performed. NAS, however, depends on exactly this kind of feedback, as future decisions need to be based on previous evaluations. CoDA-NAS closes this gap with three components that interact throughout the search, as shown in Figure 1 and Algorithm 1. The pattern extraction procedure is formalized in Algorithm 2 (Appendix A).

At each step, a searcher agent (Llama-3.3-70B (Dubey et al., 2024)) retrieves validated design patterns from the PKB, treats them as soft priors,

Algorithm 1: CoDA-NAS

Input: Benchmark $\mathcal{B}$, budget T, searcher $\mathcal{M}$, predictor f_θ, sample size K, mining interval ratio ρ, (opt.) PKB_0
Output: Best architecture $\mathbf{a}^*$

```
PKB ← PKB_0 or ∅;  H ← ∅;  a* ← ∅
for t ← 1 to T do
    P ← validated patterns from PKB as soft priors
    {a_i}_{i=1}^K ← M(a*, P, TopK(H))
    a_t ← arg max_i f_θ(a_i)
    v_t ← B(a_t);  H ← H ∪ {(a_t, v_t)}
    Update f_θ and a* with (a_t, v_t)
    if t mod ⌊ρT⌋ = 0 then
        PatternExtract(M, H, PKB, f_θ)
return a*
```

and proposes K candidate architectures through a structured generation flow. A lightweight predictor agent (DeBERTaV3-Base (He et al., 2023)) scores these candidates and selects the top-ranked one for actual benchmark evaluation. Every few iterations, the system runs PATTERNEXTRACT (Algorithm 2) to mine architecture pairs from the search history that differ by exactly one operation. It then prompts the searcher to explain the observed performance gap in natural language and uses the predictor to statistically verify the explanation before storing it in the PKB. Over time, an otherwise memoryless process gradually accumulates verified structural knowledge that carries over into future iterations.

Searcher Agent. Neural architectures are directed acyclic graphs (DAG), but LLMs are sequence models. To bridge this gap, we encode each architecture as a compact integer array $[o_0, o_1, \ldots, o_{|\mathcal{E}|-1}]$, where position i indexes edge e_i in the cell DAG and the value selects an operation from the candidate set. The structural semantics of the DAG are described once in the system prompt (Appendix B).

At each iteration, the searcher constructs a prompt that combines the current best architecture, a ranked list of top-performing candidates from $\mathcal{H}$, and the validated patterns retrieved from the PKB. Each pattern is injected in the form *"Edge e: prefer o^+ over o^- (reason: ...)"*. The prompt also specifies diversity requirements across the K candidates, asking for a mix of strategies such as applying a validated principle, exploring an unconstrained edge, combining changes, or attempting more aggressive mutations. We enforce a structured grammar with four sequential fields via constrained decoding: `reasoning`, `proposed_spec`, `change_description`, and `expected_effect`.

Because LLM generation is autoregressive, the ordering of these fields matters. The model must first write its mutation strategy in `reasoning`, which becomes part of the context when producing the actual integer array in `proposed_spec`. It then describes the modification it just made in `change_description`, tightening the link between stated intent and generated output. In batch mode (K=5), all candidates are produced in a single autoregressive pass, and the `expected_effect` of earlier candidates becomes visible context for later ones, naturally encouraging different exploration directions without explicit deduplication logic.

This design differs from the post-hoc reflection used in RZ-NAS (Ji et al., 2025), where a separate LLM call reviews the output after evaluation. Our constraint operates within the same inference call, forcing the model to articulate reasoning before generating the architecture rather than reflecting afterward. Full prompt templates and a representative output example are in Appendix B.

Pattern Knowledge Base. The PKB stores structural knowledge extracted from the search history. Every $\lfloor \rho T \rfloor$ iterations, the system scans $\mathcal{H}$ for architecture pairs $(\mathbf{a}^+, \mathbf{a}^-)$ that differ at exactly one edge ($d_{\text{edit}} = 1$). Restricting the comparison to single-edge differences serves as a controlled experiment. When two architectures share all but one edge, any performance gap is more attributable to the differing operation than if multiple edges had changed simultaneously. This contrastive formulation also aligns with recent findings that LLMs extract stronger task signals from paired positive and negative demonstrations than from positive examples alone (Mo et al., 2024). For each such pair, we prompt the searcher to explain why operation o^+ outperforms o^- at edge e, producing a short natural-language design principle (e.g., *"conv3x3 preserves spatial resolution better than skip at this aggregation stage"*). Grounding the generation in observed outcomes rather than open-ended prediction limits the scope for hallucinated explanations. Each candidate principle is stored as a tuple $(e,\ o^+,\ o^-,\ \textit{reason})$ and enters the PKB only after passing statistical validation by the predictor agent.

Unlike approaches that permanently prune operations from the search space (Zhou et al., 2025), our framework preserves the full candidate set. A purely numerical rule like "prefer op 3 over op 0 at edge 2" carries no explanation of why this preference holds. By storing the reasoning in natural language, a principle like "conv3x3 preserves spatial resolution" remains interpretable and applicable across semantically different tasks, which we evaluate in Section 5.2. As the search progresses, the PKB may accumulate principles that partially contradict each other. Rather than resolving these conflicts heuristically, we present all validated principles together and let the LLM weigh them based on the accompanying reasoning.

Predictor Agent. The predictor serves two roles during the search. First, it acts as a candidate ranker. At each iteration, the searcher proposes K=5 candidates, but only one is sent to the actual benchmark. The predictor f_θ scores all K candidates and selects the top-ranked one for ground-truth evaluation (Algorithm 1, line 5), effectively exploring five times more candidates per budget unit and ensuring that the budget is spent on the most promising candidate. Second, the predictor validates candidate patterns before they enter the PKB (Algorithm 2, lines 5 to 10). For a candidate pattern $p = (e, o^+, o^-)$, we sample n random architectures with edge e fixed to o^+ and another n with edge e fixed to o^-, keeping all other edges uniformly random. The predictor scores all $2n$ architectures, and we apply a one-sided Student's t-test to determine whether $\mu^+ > \mu^-$. The pattern is accepted into the PKB only if $p_{\text{val}} < \alpha$ and $\bar{f}_\theta(\mathbf{A}^+) > \bar{f}_\theta(\mathbf{A}^-)$.

We use predicted scores rather than ground-truth evaluations for this test. The statistical power of a t-test depends on sample size. With n=100 per group, validating a single pattern would cost 200 evaluations, far exceeding the total budget. The predictor provides these scores at zero cost. The test does not require the predicted values to be accurate in absolute terms. It only requires the predictor to produce a consistent ranking, so that architectures with the better operation receive, on average, higher scores than those without. We train the predictor with a pairwise ranking loss specifically to encourage this property (Appendix C). A DeBERTaV3-Base (He et al., 2023) ($\sim$184M parameters) is used as the predictor backbone. Since the predictor is called repeatedly during search (K scores for ranking, $2n$ for validation), a lightweight model keeps this overhead negligible. The predictor is fine-tuned online with LoRA adapters (Hu et al., 2022) and discriminative learning rates to preserve cross-task representations while adapting to new evaluations (Appendix C).

4 Experiments

4.1 Experimental Setup

Search Spaces and Datasets. We evaluate CoDA-NAS on two diverse search spaces: NAS-Bench-201 (NB201) (Dong and Yang, 2020) and TransNAS-Bench-101 (Trans101) (Duan et al., 2021). For NB201, we report validation and test performance on CIFAR-100 and ImageNet16-120, as the base CIFAR-10 dataset is reserved exclusively for establishing the foundational prior. For Trans101, we evaluate across all seven distinct computer vision tasks to assess the framework's cross-domain semantic generalization. Comprehensive details regarding the topological structures, candidate operators, and dataset characteristics for both search spaces are provided in Appendix D.

Baselines and Search Protocol. We benchmark CoDA-NAS against two groups of baselines: classic NAS algorithms (e.g., DARTS (Liu et al., 2019), REA (Real et al., 2019), HEBO (Cowen-Rivers et al., 2022)) and state-of-the-art LLM-driven frameworks (e.g., GENIUS (Zheng et al., 2023), LLMatic (Nasir et al., 2024), RZ-NAS (Ji et al., 2025), LAPT (Zhou et al., 2025), LM-Searcher (Hu et al., 2025); summarized in Table 1). Unlike prior works relying on providers' APIs (e.g., GPT-4) or extensive offline instruction-tuning, our framework utilizes a tuning-free and open-weights model (Llama-3.3-70B-Instruct (Dubey et al., 2024)) as the searcher agent. For a fair comparison, all LLM-based methods are restricted to a search budget of 50 architecture evaluations. We report the optimal validation metrics averaged across five independent random seeds per task. As tabular benchmarks represent architectures as integer vectors without model instances, evaluation metrics that require model instantiation (e.g., zero-cost proxy scores, FLOPs-based descriptors) are replaced with equivalent structure-based estimators computable directly from the vector representation. For any baseline relying on offline prior extraction, we standardize its source dataset to CIFAR-10, ensuring all methods start from identical foundational knowledge. To eliminate confounding capability differences, all evaluated LLM baselines (except LM-Searcher) share GPT-4o as their common reasoning backbone.

Implementation Details and Computation Cost. Our dual-agent framework supports distributed deployment across commodity GPUs; the searcher runs Llama-3.3-70B-Instruct (Dubey et al., 2024) and the predictor is initialized from DeBERTaV3-Base (He et al., 2023) (hardware details in Appendix B). The predictor is initially pre-trained on a limited set of CIFAR-10 evaluations to establish the foundational structural prior. When transferring to downstream tasks, it adapts online, removing the need for dataset-specific pre-training. This highly efficient approach requires only $\sim$0.04 GPU-days to construct this foundational prior, and about $\sim$0.02 GPU-days per online search trial. Consequently, our entire main evaluation (9 tasks across 5 seeds) consumed only $\sim$1.1 GPU-days. We defer further details on hyperparameters, prompts, and software environments to Appendix B.

4.2 Main Results

As shown in Table 2, CoDA-NAS achieves highly competitive results on NAS-Bench-201. On CIFAR-100, our framework consistently discovers the theoretical global optimum of the search space on both validation and test sets with zero variance across seeds, matching specialized gradient-based methods like DrNAS (Chen et al., 2021) and β-NAS (Ye et al., 2022), while previous LLM-driven methods (e.g., RZ-NAS (Ji et al., 2025), GENIUS (Zheng et al., 2023)) fall short of this upper bound. On the more challenging ImageNet-16-120, CoDA-NAS achieves the highest validation accuracy among all methods and remains comparable to the best LLM baseline on the test set, demonstrating consistent competitiveness under the strict 50-evaluation budget. On TransNAS-Bench-101 (Table 3), CoDA-NAS achieves a state-of-the-art Average Rank (11.0), outperforming recent LLM-driven methods such as LAPT (20.5) and RZ-NAS (26.2). While baseline methods may excel in specific tasks (e.g., normal estimation or jigsaw), their performance is often inconsistent across diverse semantic domains. By formulating architectural knowledge as natural language patterns within the PKB, CoDA-NAS maintains robust performance across both classification and dense-prediction tasks (e.g., semantic segmentation).

5 Analysis

5.1 Ablation study

To isolate the mechanisms within CoDA-NAS, we evaluate five additive variants (Base, +PKB, +CoT, PKB+CoT, and Full) across TransNAS-Bench-101. Here, "CoT" denotes the structured autoregressive

Method	CIFAR-100		ImageNet-16-120	
	Valid Acc	Test Acc	Valid Acc	Test Acc
DARTS (Liu et al., 2019) (ICLR '19)	15.03 ± 0.00	15.61 ± 0.00	16.43 ± 0.00	16.32 ± 0.00
SNAS (Xie et al., 2019) (ICLR '19)	69.69 ± 2.39	69.34 ± 1.98	42.84 ± 1.79	43.16 ± 2.64
PC-DARTS (Xu et al., 2020) (ICLR '20)	67.12 ± 0.39	67.48 ± 0.89	40.83 ± 0.08	41.31 ± 0.22
RLNAS (Zhang et al., 2021) (CVPR '21)	70.98 ± 0.00	70.71 ± 0.00	46.86 ± 0.00	43.70 ± 0.00
DrNAS (Chen et al., 2021) (ICLR '21)	73.49 ± 0.00	73.51 ± 0.00	46.37 ± 0.00	46.34 ± 0.00
β-NAS (Ye et al., 2022) (CVPR '22)	73.49 ± 0.00	73.51 ± 0.00	46.37 ± 0.00	46.34 ± 0.00
REA (Real et al., 2019) (AAAI '19)	71.45 ± 1.59	71.56 ± 1.48	45.38 ± 1.06	45.51 ± 0.91
MetaD2A (Lee et al., 2021) (ICLR '21)	72.53 ± 0.63	72.35 ± 0.62	46.09 ± 0.33	45.97 ± 0.35
HEBO (Cowen-Rivers et al., 2022) (JAIR '22)	73.15 ± 0.43	73.30 ± 0.29	46.21 ± 0.52	46.25 ± 0.75
TNAS-BO (Shala et al., 2023) (ICLR '23)	72.10 ± 0.89	72.12 ± 1.11	46.01 ± 0.38	46.20 ± 0.58
GENIUS (Zheng et al., 2023) (arXiv '23)	72.37 ± 0.63	72.20 ± 0.72	45.99 ± 0.60	45.88 ± 0.80
LLMatic (Nasir et al., 2024) (GECCO '24)	71.25 ± 0.49	71.46 ± 0.84	45.83 ± 0.48	45.43 ± 0.76
RZ-NAS (Ji et al., 2025) (ICML '25)	73.28 ± 0.42	73.38 ± 0.27	46.23 ± 0.21	**46.50 ± 0.11**
LM-Searcher (Hu et al., 2025) (EMNLP '25)	71.40 ± 0.52	71.27 ± 0.76	45.17 ± 0.99	45.44 ± 0.95
CoDA-NAS (Ours)	**73.49 ± 0.00**	**73.51 ± 0.00**	**46.56 ± 0.15**	**46.46 ± 0.22**

Table 2: Top-1 performance comparison on NAS-Bench-201. Accuracy (%) is reported as Mean ± Std over 5 seeds.

Method	Cls.O Acc↑	Cls.S Acc↑	Room L2↓	Auto SSIM↑	Nor SSIM↑	Seg mIoU↑	Jig Acc↑	Avg Rank↓
RS (Li and Talwalkar, 2019) (UAI '19)	45.16	54.41	61.48	55.94	56.85	25.21	94.47	58.0
REA (Real et al., 2019) (AAAI '19)	45.39	54.62	61.75	**56.96**	57.22	25.52	94.62	31.1
BONAS (Shi et al., 2020) (NeurIPS '20)	45.50	54.46	61.10	56.73	57.46	25.32	94.81	25.1
weakNAS (Wu et al., 2021) (NeurIPS '21)	45.66	54.72	**60.31**	56.77	57.21	25.90	94.63	17.1
Arch-Graph-single (Huang et al., 2022) (CVPR '22)	45.48	54.70	61.05	56.52	57.53	25.71	94.66	20.0
LLMatic (Nasir et al., 2024) (GECCO '24)	45.51 ± 0.17	54.88 ± 0.05	61.88 ± 0.90	**56.88 ± 0.43**	56.84 ± 0.09	25.69 ± 0.17	94.49 ± 0.07	46.9
RZ-NAS (Ji et al., 2025) (ICML '25)	45.78 ± 0.33	54.71 ± 0.18	61.49 ± 0.42	56.36 ± 0.66	57.31 ± 0.23	25.93 ± 0.18	94.97 ± 0.35	26.2
LM-Searcher (Hu et al., 2025) (EMNLP '25)	45.46 ± 0.34	54.61 ± 0.18	60.72 ± 0.63	56.66 ± 0.41	56.91 ± 0.05	25.59 ± 0.23	94.54 ± 0.13	34.0
LAPT (Zhou et al., 2025) (AAAI '25)	45.58 ± 0.17	54.87 ± 0.05	60.44 ± 0.64	56.17 ± 0.99	**58.33 ± 1.08**	25.83 ± 0.37	**95.37 ± 0.00**	20.5
CoDA-NAS (Ours)	**46.00 ± 0.19**	**54.91 ± 0.03**	60.37 ± 0.61	56.70 ± 0.29	57.73 ± 0.96	**26.13 ± 0.18**	94.78 ± 0.09	**11.0**
Global Best	46.32	54.94	59.38	57.72	59.62	26.27	95.37	1

Table 3: Top-1 performance on TransNAS-Bench-101. Traditional baselines from Huang et al. (2022); LLM methods report Mean ± Std over 5 seeds. Avg Rank: global architecture rank across all 7 tasks.

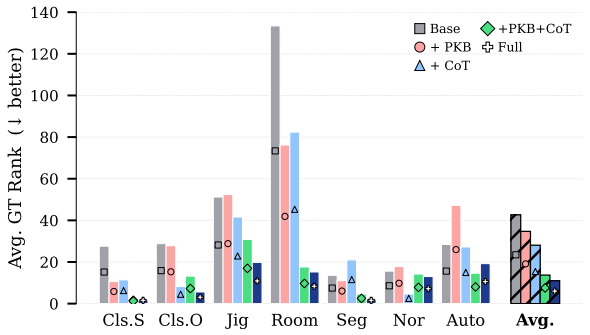

Figure 2: Ablation on TransNAS-Bench-101. Average ground-truth rank ↓ across five seeds.

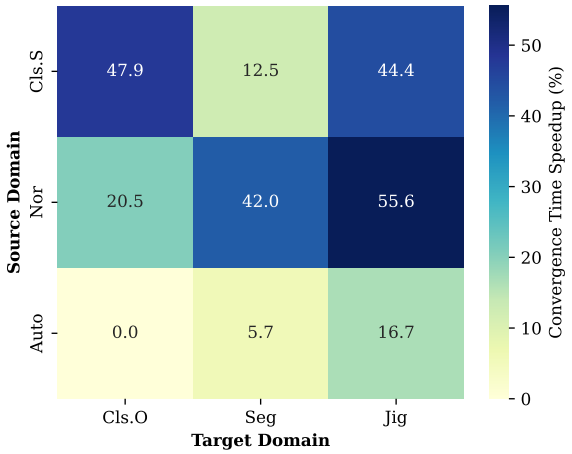

Figure 3: Convergence speedup matrix for cross-domain pattern transfer. All nine entries are non-negative.

generation flow (Section 3.2), and the "Full" system further incorporates the statistical predictor. The results in Figure 2 show a consistent improvement in average rank from 42.7 to 11.0. A central finding is how explicit reasoning activates historical priors. Providing the LLM with isolated rules (+PKB) yields only a small gain, from 42.7 to 34.7, because the model functions as a basic parser applying fixed constraints. However, when these rules are combined with a structured generation sequence (PKB+CoT), the average rank drops by 21.0 positions to 13.7. This explicit generative constraint effectively bridges the semantic-to-structural gap. Without it, the model might retrieve a rule like "use `conv3x3` at edge 2" yet output a topological array that fails to incorporate it. By enforcing a closed-loop generation flow, the autoregressive sequence firmly anchors the final decision. It compels the searcher to articulate why `conv3x3` preserves spatial resolution for dense reconstruction, and strictly translates this semantic understanding into a discrete topological output. This confirms that historical patterns alone (+PKB) are insufficient without structural alignment. Although this reasoning is effective on its own, task-level

Task	# Pats	Win(%)	ΔP(%)	r
CIFAR-100	25	83.2	+16.4	+0.21
ImageNet-16-120	14	84.5	+17.4	+0.29
Cls.S	10	74.3	+15.5	+0.24
Seg	20	68.5	+13.8	+0.21
Nor	9	73.7	+13.5	+0.20
Jigsaw	7	68.5	+13.1	+0.18
Cls.O	7	67.4	+14.5	+0.24
Room	7	62.2	+13.2	+0.25
Auto	6	56.5	+9.0	+0.12

Table 4: Global evaluation of consensus patterns. **# Pats**: rules found by $\geq$2 seeds; **Win**: minimal-pair win rate; ΔP: avg. percentile rank improvement; r: point-biserial correlation.

variance confirms the predictor's role as a conservative regularizer. In extreme cases like *Scene* or *AE*, pure (PKB+CoT) variant alone achieves the best results, reaching Rank 1.0 in *Scene*. Because the predictor enforces a strict statistical threshold, the Full system occasionally deviates slightly from these peak efficiencies. We consider this a deliberate trade-off where we sacrifice minor flexibility for a strong safeguard of stability. This grounding prevents degradation in volatile tasks like *Room Layout* (from 17.6 to 15.2) and resolves inductive bias collisions in *Normal* (from 17.8 to 13.0). The predictor verifies each hypothesis to stabilize these final ranks at 15.2 and 13.0 respectively.

5.2 Reliability and Transferability of Patterns

To ensure the PKB captures genuine structural preferences, we evaluate all consensus patterns (discovered by $\geq$2 seeds) via exhaustive minimal-pair comparison against the full architecture pool. Isolating architecture pairs that differ *only* at the target edge, Table 4 reports strict win rates between 56.5% and 84.5%, with average percentile rank improvements of +9.0 to +17.4 points. While the point-biserial correlations ($r \approx 0.12$–0.29) appear low, each pattern governs exactly one of six independently varying edges; assuming equal structural contribution, a single edge accounts for at most $1/6$ of the total performance variance, capping the theoretical correlation ceiling at $r_{\max} = \sqrt{1/6} \approx 0.41$. Our linguistically extracted rules therefore reach 29% to 71% of this theoretical ceiling across both benchmarks.

Since CoDA-NAS encodes architectural knowledge as natural-language patterns rather than weight-level statistics, we also investigate whether these patterns generalize across semantically distant tasks. We construct a 3×3 transfer matrix by selecting one source from each task family (Scene, Normal, Autoencoder) and transferring their validated PKB into three unseen targets (Object, Segmentation, Jigsaw). Each target is searched via both a cold start (no prior) and a warm start (source PKB). Convergence speedup is defined as the percentage reduction in steps to reach 98% of the cold-start's final performance, averaged over five seeds. As shown in Figure 3, all nine speedup values are non-negative. While conventional weight-sharing methods often suffer from negative transfer when domains are misaligned (Duan et al., 2021; Huang et al., 2022), our framework largely avoids this failure mode. Even in the Autoencoder row, where self-supervised reconstruction favors skip connections and shallow bottlenecks that could mislead a classification searcher, the online validation loop continuously mines locally grounded patterns that naturally dilute misaligned foreign priors, yielding a neutral +0.0% on Object and +5.7% on Segmentation. When source and target share structural similarities, gains are substantial: Scene $\rightarrow$ Object achieves 47.9%, Normal $\rightarrow$ Segmentation 42.0%, and Normal $\rightarrow$ Jigsaw yields the highest speedup of 55.6% despite crossing task families, suggesting that certain low-level spatial priors can generalize across task-specific semantics when expressed in natural language.

6 Conclusion

In this work, we introduced CoDA-NAS, a novel cooperative framework that enables LLM agents to extract, verify, and accumulate structural knowledge during neural architecture search. By converting observed performance differences into natural-language design patterns and validating them through statistical testing, the search process accumulates verified knowledge that carries across iterations and tasks. Our results show that grounding LLM generation in verified linguistic knowledge is crucial for mitigating hallucinations and sustaining effective search under strict evaluation budgets, even when powered entirely by a local open-weights model. These findings suggest that natural language can function as a practical knowledge representation for discrete optimization, moving beyond its traditional role as a communication interface. As the knowledge cycle operates fully online without task-specific offline training, this design opens a promising path toward extending knowledge-guided LLM search to broader optimization problems.

Limitation

Our evaluation relies on tabular benchmarks where each query returns a precomputed metric instantly. While this gives us reliable ground truth for validating patterns, it hides the real cost of training neural networks from scratch, so how well our approach scales to compute-heavy settings remains untested. In addition, our cross-task experiments transfer patterns across different tasks but within the same search space. The current encoding assumes a fixed-topology cell, and applying the knowledge cycle to different search spaces, such as those with variable depth or macro-level topology, would require rethinking how architectures are represented. Developing such encodings while keeping the online mining and validation loop intact is a natural next step.

Acknowledgements

The authors acknowledge the use of the Gemini model family as an assistive tool during the development of this project. It was specifically employed to accelerate routine programming tasks and to polish the grammatical structure of the manuscript. The core methodology, experimental design, and data analysis were conducted entirely by the human authors. Following the use of this tool, all text and codebase elements were carefully validated, modified and tested. The authors bear full responsibility for the final manuscript and the accompanying software.

Ethics Statement

As an automated neural architecture search framework, CoDA-NAS is fundamentally an optimization tool. While there is a theoretical risk of amplifying inherent biases if applied to sensitive datasets, we assess the direct risk of harm to be minimal. Furthermore, although LLM inference carries an environmental cost, our methodology offsets this by significantly reducing the carbon footprint associated with traditional NAS evaluations. The datasets and models utilized in this work are open-source and applied strictly for academic research, aligning with the intended use cases specified by their creators.

References

Angelica Chen, David R. So, and David M. Dohan. 2023. EvoPrompting: Language models for code-level neural architecture search. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36.

Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. 2021. DrNAS: Dirichlet neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Alexander I Cowen-Rivers, Wenlong Lyu, Rasul Tutunov, Zhi Wang, Antoine Grosnit, Ryan Rhys Griffiths, Hao Jianye, Jun Wang, and Haitham Bou Ammar. 2022. HEBO: Pushing the limits of sample-efficient hyperparameter optimisation. *Journal of Artificial Intelligence Research (JAIR)*, 74:1269–1349.

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

Yawen Duan, Xin Chen, Hang Xu, Zewei Chen, Xiaodan Liang, Tong Zhang, and Zhenguo Li. 2021. TransNAS-bench-101: Improving transferability and generalizability of cross-task neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5251–5260.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023. DeBERTaV3: Improving DeBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Edward J Hu, Yelong Shen, Phil Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Yuxuan Hu, Jihao Liu, Ke Wang, Jinliang Zhen, Weikang Shi, Manyuan Zhang, Qi Dou, Rui Liu, Aojun Zhou, and Hongsheng Li. 2025. 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 (EMNLP)*, pages 9408–9421.

Minbin Huang, Zhijian Huang, Changlin Li, Xin Chen, Hang Xu, Zhenguo Li, and Xiaodan Liang. 2022. Arch-graph: Acyclic architecture relation predictor for task-transferable neural architecture search. In

Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 11881–11891.

Zipeng Ji, Guanghui Zhu, Chunfeng Yuan, and Yihua Huang. 2025. RZ-NAS: Enhancing LLM-guided neural architecture search via reflective zero-cost strategy. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 27237–27254.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Hao Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, pages 611–626.

Hayeon Lee, Eunho Lee, and Sung Ju Hwang. 2021. MetaD2A: a meta-learning framework for data-driven network architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33.

Liam Li and Ameet Talwalkar. 2019. Random search and reproducibility for neural architecture search. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 367–377.

Zhe Li, Zhiwei Lin, and Yongtao Wang. 2026. CoLLM-NAS: Collaborative large language models for efficient knowledge-guided neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Ying Mo, Jiahao Liu, Jian Yang, Qifan Wang, Shun Zhang, Jingang Wang, and Zhoujun Li. 2024. C-ICL: Contrastive in-context learning for information extraction. In *Findings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 10099–10114.

Muhammad U. Nasir, Sam Earle, Julian Togelius, Steven James, and Christopher Cleghorn. 2024. LLMatic: Neural architecture search via large language models and quality diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1–9.

Shiwen Qin, Gabriela Kadlecová, Martin Pilát, Shay B. Cohen, Roman Neruda, Elliot J. Crowley, Jovita Lukasik, and Linus Ericsson. 2025. Transferrable surrogates in expressive neural architecture search spaces. In *Proceedings of the International Conference on Automated Machine Learning (AutoML)*.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 4780–4789.

Gresa Shala, Thomas Elsken, Frank Hutter, and Josif Grabocka. 2023. Transfer NAS with meta-learned bayesian surrogates. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James Kwok, and Jianfei Chen. 2020. Bridging the gap between sample-based and one-shot neural architecture search with BONAS. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33.

Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. 2020. Neural predictor for neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 660–676.

Colin White, Arber Zela, Binxin Ru, Yang Liu, and Frank Hutter. 2021. How powerful are performance predictors in neural architecture search? In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34.

Junru Wu, Xiyang Dai, Dongdong Chen, Yinpeng Chen, Mengchen Liu, Ye Yu, Zhangyang Wang, Zicheng Liu, Mei Chen, and Lu Yuan. 2021. Stronger NAS with weaker predictors. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2019. SNAS: Stochastic neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. 2020. PC-DARTS: Partial channel connections for memory-efficient architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Peng Ye, Baopu Li, Yikang Li, Tao Chen, Jiayuan Fan, and Wanli Ouyang. 2022. β-DARTS: Beta-decay regularization for differentiable architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10874–10883.

Caiyang Yu, Xianggen Liu, Yifan Wang, Yun Liu, Wentao Feng, Xiong Deng, Chenwei Tang, and Jiancheng Lv. 2025. GPT-NAS: Neural architecture search meets generative pre-trained transformer model. *Big Data Mining and Analytics*, 8:45–64.

Xuanyang Zhang, Pengfei Hou, Xiangyu Zhang, and Jian Sun. 2021. Neural architecture search with random labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10907–10916.

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

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

Barret Zoph and Quoc V Le. 2017. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

A Pattern Extraction Procedure

Algorithm 2 formalizes the pattern mining and statistical validation procedure described in Section 3.2.

Algorithm 2: PATTERNEXTRACT

Input: Searcher $\mathcal{M}$, history $\mathcal{H}$, PKB, predictor f_θ, sample size n, level α
Output: Updated PKB

$Pairs \leftarrow \{(\mathbf{a}^+, \mathbf{a}^-) \in \mathcal{H}^2 \mid d_{\text{edit}}(\mathbf{a}^+, \mathbf{a}^-) = 1\}$
foreach $(\mathbf{a}^+, \mathbf{a}^-) \in$ *Pairs differing at e by* o^+, o^- **do**
  $r \leftarrow \mathcal{M}.\text{explain}(\mathbf{a}^+, \mathbf{a}^-, e)$
  $\text{PKB.add}(e,\ o^+,\ o^-,\ r)$
foreach *unvalidated* $p = (e, o^+, o^-) \in \text{PKB}$ **do**
  $\mathbf{A}^+ \leftarrow$ Sample n archs fixing $\mathbf{a}_e = o^+$
  $\mathbf{A}^- \leftarrow$ Sample n archs fixing $\mathbf{a}_e = o^-$
  $p_{\text{val}} \leftarrow$ one-tail-t-test$(f_\theta(\mathbf{A}^+),\ f_\theta(\mathbf{A}^-))$
  if $p_{\text{val}} < \alpha \wedge \bar{f}_\theta(\mathbf{A}^+) > \bar{f}_\theta(\mathbf{A}^-)$ **then** accept p
  else reject p
return PKB

B Implementation Details

B.1 Search Hyperparameters

Table 5 lists all hyperparameters used in the search loop. The temperature adapts during search via $t = \min(1.2,\ 0.7 + 0.1 \cdot s)$, where s counts consecutive iterations without improvement. This gradually encourages more diverse outputs when the search stalls.

Category	Parameter	Value
Search Loop	Total budget T	50
	Batch size K	5
	Mining interval $\lfloor \rho T \rfloor$	5 evals
	Max pairs per mining round	3
	Min. performance gap	0.005
LLM Inference	Temperature (base)	0.7
	Temperature (adaptive max)	1.2
	Top-p	0.9
	Repetition penalty	1.05
Pattern Validation	Samples per group n	100
	Significance level α	0.05
Cold Start	Initial groundings	5
	Candidates before grounding	5

Table 5: Search loop hyperparameters. All values are fixed across tasks and random seeds.

B.2 Searcher Agent

The searcher is served via vLLM (Kwon et al., 2023) with a maximum context length of 8,192 tokens and GPU memory utilization of 90%. The `lm-format-enforcer` backend handles constrained decoding. Maximum output length is 1,000 tokens for batch proposals (K=5) and 256 tokens for single proposals and pattern comparisons.

B.3 Prompt Templates

The searcher operates with three prompt types. The *system prompt* is instantiated once per task and describes the target domain, the DAG topology with the operation-to-index mapping, the evaluation metric with its expected range, and a concise role instruction. The *contrastive comparison prompt* presents a minimal pair of architectures (edit distance = 1) with their performance gap and asks the model to state a generalizable design principle in one sentence.

System Prompt

```
You are a neural architecture search expert.
Target Domain: {task_info}

{dag_description_and_information_flow}

Evaluation metric: {metric_name}
{metric_info}
ALL scores follow the convention: HIGHER is BETTER.

Your job is to analyze architectures, compare designs,
and propose improvements for {task_info}.
MANDATORY: Keep 'reasoning' concise and technically
focused (under 80 words).
NEVER repeat the same phrase multiple times. Stop
immediately once the point is made.
```

Contrastive Comparison Prompt

```
Compare these two architectures that differ only at
edge {e}:

Architecture A: {spec_a} -> {metric_name} = {val_a}
Architecture B: {spec_b} -> {metric_name} = {val_b}

The only difference is edge {e}: A uses {op_a}, B uses
{op_b}.
A scores {gap} higher on the numerical optimization
scale.

Why does {op_a} work better than {op_b} at edge {e}?
State a generalizable design principle in one
sentence.
Be concise.
```

The *batch proposal prompt* is the primary generation prompt used at each search iteration. It supplies the current best architecture, a ranked top-5 history, all validated PKB patterns, and explicit diversity requirements:

Batch Proposal Prompt

```
Current best architecture: {best_spec}
 -> {metric_name}: {best_val}

Top architectures found so far:
  {spec_1} -> {val_1}
  ...
  {spec_5} -> {val_5}

Validated design principles:
{PKB patterns in natural language}

Propose 5 DIFFERENT architectures that each attempt to
improve the {metric} beyond {best_val}.
Rules:
- Each candidate MUST differ from every other
candidate in at least one edge.
- Each candidate MUST differ from the current best.
- Cover diverse mutation strategies: (1) apply a
validated principle, (2) explore an unconstrained
edge, (3) combine multiple small changes, (4-5)
creative or aggressive mutations.
- For each candidate, provide concise reasoning (under
60 words), the proposed_spec, change_description, and
expected_effect.
CRITICAL: Return exactly 5 candidates in a JSON object
with a "candidates" array.
```

Each candidate is constrained to a four-field JSON schema enforced via Pydantic: `reasoning` (max 15 words), `proposed_spec` (integer array), `change_description` (max 10 words), and `expected_effect` (max 10 words).

B.4 Representative LLM Output

Below is an excerpt from a TransNAS-Bench-101 Scene Classification search. The current best architecture is `[3, 0, 3, 0, 1, 1]` with a score of 54.1%.

Representative Output (2 of 5 candidates shown)

```
{"candidates": [
  {"reasoning": "Apply PKB: conv3x3 preserves spatial
features at edge 2.",
   "proposed_spec": [3, 2, 3, 0, 1, 1],
   "change_description": "Edge 1: none->conv3x3",
   "expected_effect": "Better feature flow"},
  {"reasoning": "Edge 4 skip avoids premature
downsampling per PKB.",
   "proposed_spec": [3, 0, 3, 0, 3, 1],
   "change_description": "Edge 4: skip->skip_conn",
   "expected_effect": "Preserve spatial dims"},
  ... (3 more candidates)
]}
```

B.5 Software Environment

The searcher agent is hosted on two NVIDIA RTX A6000 GPUs and the predictor is concurrently trained and served on a single NVIDIA RTX 5080 GPU. All experiments use Python 3.11, PyTorch 2.5, vLLM 0.6, HuggingFace Transformers 4.47, and PEFT 0.14 for LoRA adapters. The predictor server communicates with the searcher via a REST API over localhost. All random seeds are controlled at both the Python and PyTorch levels. All datasets and models used in this work are publicly available for research purposes. Specifically, Llama-3.3 and DeBERTaV3 are licensed under the Llama 3 Community License and MIT License, respectively. TransNAS-Bench-101 and NAS-Bench-201 are utilized under their respective open-source terms.

C Predictor Agent Training Details

The predictor uses DeBERTaV3-Base ($\sim$184M parameters) with a two-layer regression head ($768 \rightarrow 384 \rightarrow 1$, ReLU activation, dropout 0.1). Each architecture is encoded as a structured natural-language instruction that describes the target domain and dataset characteristics, followed by a graph-notation string that names every operation and its input node. An example is shown below. To support zero-shot transfer, the TransNAS-Bench-101 operation names are mapped to the NAS-Bench-201 vocabulary (e.g., `zero`$\rightarrow$`none`, `conv3x3`$\rightarrow$`nor_conv_3x3`) so that the predictor sees a consistent token distribution across both benchmarks.

Predictor Input Example (CIFAR-10)

```
**Task Description**
- Domain: Image Classification
- Benchmark: NAS-Bench-201
- Dataset: CIFAR-10
- Dataset Characteristics: 10 object classes in 32x32
 pixel color images.
- Architectural Hints:
  1. Convolutional operations are essential to capture
 local spatial patterns.
  2. Avoid excessive downsampling early in the
 network.

**Architecture String**
n1=nor_conv_3x3(n0) | n2=skip_connect(n0)
+nor_conv_1x1(n1) | n3=nor_conv_3x3(n0)
+avg_pool_3x3(n1)+nor_conv_3x3(n2)
```

The predictor is first pre-trained on all 15,625 NAS-Bench-201 CIFAR-10 architectures using pairwise ranking loss (MarginRankingLoss), since the search loop requires only consistent relative ordering rather than accurate absolute predictions. CIFAR-10 is used exclusively for this pre-training phase and is never included as a search target in our experiments. During the search, the predictor is fine-tuned online with LoRA adapters attached to the query and value projections. Because only a few dozen ground-truth evaluations are available during search, full fine-tuning risks destabilizing the pre-trained ranking quality. We therefore freeze the base weights and apply discriminative learning rates, with a higher rate for the regression head and a much smaller rate for the LoRA parameters. The online objective adds a small MSE anchor term (weight = 0.05) to the ranking loss so that predicted scores remain calibrated for the t-test validation described in Section 3.2. All hyperparameters are listed in Table 6. Upon acceptance, we will release the pre-trained checkpoint to support reproducibility.

Phase	Parameter	Value
Pre-train	Epochs	15
	Learning rate	1×10^{-5}
	Batch size	16
	Ranking margin	0.1
	Pair multiplier	2.0
Online	Epochs	3
	Head learning rate	2×10^{-4}
	LoRA learning rate	5×10^{-6}
	Batch size	8
	Ranking margin	0.2
	Pair multiplier	3.0
	LoRA r / α	8 / 16
	LoRA dropout	0.1
Shared	Weight decay	0.01
	Gradient clip	1.0

Table 6: Predictor training hyperparameters.

D Search Space and Dataset Characteristics

NAS-Bench-201. NAS-Bench-201 (Dong and Yang, 2020) defines a cell-based search space with 4 nodes and 6 directed edges forming a DAG. Each edge selects one operation from a candidate set of 5 operators: `none`, `skip_connect`, `nor_conv_1x1`, `nor_conv_3x3`, and `avg_pool_3x3`. An architecture is encoded as a 6-dimensional integer array $[o_0, \ldots, o_5]$, yielding a total of $5^6 = 15{,}625$ unique architectures. The information flow follows a densely connected pattern:

$$
\begin{aligned}
s_1 &= e_0(s_0), \\
s_2 &= e_1(s_0) + e_2(s_1), \\
s_3 &= e_3(s_0) + e_4(s_1) + e_5(s_2),
\end{aligned}
$$

where s_0 is the input node and s_3 is the output node. Each architecture is evaluated on three image classification datasets: CIFAR-10, CIFAR-100, and ImageNet-16-120. All tasks report validation and test accuracy (%). In our experiments, CIFAR-10 is reserved exclusively for pretraining the predictor's foundational prior and is not used as a search target.

TransNAS-Bench-101. TransNAS-Bench-101 (Duan et al., 2021) also uses a cell-based DAG with 4 nodes and 6 edges, but with a smaller operation set of 4 operators: `zero`, `skip`, `conv1x1`, and `conv3x3`. This yields $4^6 = 4{,}096$ unique architectures per task. The benchmark spans seven semantically diverse computer vision tasks, covering classification, self-supervised learning, and dense prediction. Table 7 summarizes each task, its learning objective, and the evaluation metric used. This semantic diversity makes

Task	Objective	Metric
Scene	Image Classification	Top-1 Acc. (%)
Object	Image Classification	Top-1 Acc. (%)
Jigsaw	Self-supervised Learning	Top-1 Acc. (%)
Room Layout	Spatial Reasoning	Loss
Segmentation	Semantic Segmentation	mIoU
Normal	Surface Normal Est.	SSIM
Autoencoder	Image Reconstruction	SSIM

Table 7: TransNAS-Bench-101 tasks, their learning objectives, and evaluation metrics.

TransNAS-Bench-101 particularly suitable for evaluating cross-task transferability, as the seven tasks span fundamentally different loss landscapes despite sharing the same topological search space.

Task	C10	C100	ImN	Nor	Cls.S
τ	.617	.604	.610	.549	.526
Task	Seg	Cls.O	Jig	Room	Auto
τ	.452	.447	.407	.300	.120

Table 8: Zero-shot rank correlation (τ) of the base predictor.

E Zero-Shot Reliability of Predictor Agent

Table 8 reports the Kendall rank correlation (τ) between the predictor's zero-shot predictions and the ground-truth performance on each task, measured before any online fine-tuning. The predictor achieves moderate correlation on tasks whose evaluation structure resembles the pre-training source (CIFAR-10), while tasks with fundamentally different objectives, such as Room Layout (spatial regression) and Autoencoder (image reconstruction), yield considerably lower initial correlations. Despite this variance, CoDA-NAS achieves the best average rank across all seven TransNAS-Bench-101 tasks (Table 3), remaining competitive even on the two tasks where zero-shot τ is weakest. This suggests that the framework's robustness is primarily from the knowledge accumulation in the PKB rather than from the initial accuracy of the predictor.

F Hyperparameter Sensitivity of Statistical Threshold

To evaluate the impact of the validation strictness on the pattern knowledge base (PKB), we ablate the t-test significance level (α) in the PATTERNEXTRACT procedure. Table 9 presents the performance across TransNAS-Bench-101 under $\alpha \in \{0.01, 0.05, 0.10, 0.20\}$.

A strict threshold ($\alpha = 0.01$) rejects many useful structural priors, which limits the domain knowledge provided to the LLM and drops the average global rank to 25.9. On the other hand, larger thresholds ($\alpha \geq 0.10$) fail to filter out random performance noise. This causes the LLM to learn incorrect patterns (for example, the noticeable performance drop on the Jigsaw task at $\alpha = 0.20$) and worsens the average rank to 21.9 and 25.9. The default threshold of $\alpha = 0.05$ provides the best balance. By effectively separating useful patterns from statistical noise, it achieves a much better average global rank of 11.0, showing that the framework relies heavily on proper statistical validation to accumulate reliable knowledge.

G Impact of Searcher Model Size

To assess whether the CoDA-NAS framework is highly tied to large-scale LLMs, we replace the default Llama-3.3-70B searcher with Llama-3.1-8B-Instruct while keeping all other components identical. Table 10 reports the results on TransNAS-Bench-101. While the 8B model remains comparable on certain individual tasks (e.g., Surface Normal Prediction), its average rank degrades substantially (27.2 vs. 11.0), with the largest drops on tasks that demand complex spatial or combinatorial reasoning (e.g., Jigsaw, Room Layout). This confirms that the structured autoregressive reasoning and pattern extraction mechanisms in CoDA-NAS benefit significantly from the stronger capabilities of larger models.

H Search Space Diversity Analysis

While integrating RAG provides valuable historical priors, it introduces a potential risk of losing search diversity. To quantify this, we measure topological diversity in Table 11, defined as the average percentage of differing edges among proposed architectures within a search budget. As expected, the unconstrained *Base* exploration exhibits a high variance of 71.4%. Integrating isolated rules (+*PKB*) only slightly reduces this to 69.5%. This minimal drop confirms that the natural-language patterns function as soft conceptual priors, guiding the search without the rigid structural lock-in seen in hard-constrained algorithms. The actual cause of the diversity reduction is the reasoning mechanism, as the +*CoT* variant drops to 51.5%. Rather than a flaw, this is an expected outcome of the thinking constraint. Forcing the LLM to justify its modifications naturally shifts its behavior from unconstrained global exploration to targeted local refinement. The *Full* system settles at 50.1%. By ensuring evaluated architectures still differ by half of their edges, the framework retains enough variance to avoid local optima while keeping the search efficient.

α Threshold	Cls.O Acc↑	Cls.S Acc↑	Room L2↓	Auto SSIM↑	Nor SSIM↑	Seg mIoU↑	Jig Acc↑	Avg Rank↓
$\alpha = 0.01$	45.51 ± 0.36	54.83 ± 0.21	60.71 ± 1.08	56.40 ± 0.23	57.46 ± 0.18	25.91 ± 0.11	94.70 ± 0.13	25.9
$\alpha = 0.05$ (Default)	$\mathbf{46.00 \pm 0.19}$	54.91 ± 0.03	$\mathbf{60.37 \pm 0.61}$	56.70 ± 0.29	57.73 ± 0.96	$\mathbf{26.13 \pm 0.18}$	$\mathbf{94.78 \pm 0.09}$	**11.0**
$\alpha = 0.10$	45.37 ± 0.10	$\mathbf{54.92 \pm 0.03}$	60.75 ± 0.35	56.08 ± 0.49	$\mathbf{57.80 \pm 0.93}$	26.03 ± 0.20	94.72 ± 0.05	21.9
$\alpha = 0.20$	45.55 ± 0.34	54.91 ± 0.03	60.46 ± 0.56	$\mathbf{56.79 \pm 0.32}$	57.21 ± 0.15	26.04 ± 0.19	94.55 ± 0.44	25.9

Table 9: Sensitivity of the search framework to the T-test significance level (α). Performance represents Mean $\pm$ Std across 5 random seeds on TransNAS-Bench-101. The default threshold ($\alpha = 0.05$) provides the best balance between learning verified structural knowledge and filtering out random noise, achieving the best average global rank.

Model Size	Cls.O Acc↑	Cls.S Acc↑	Room L2↓	Auto SSIM↑	Nor SSIM↑	Seg mIoU↑	Jig Acc↑	Avg Rank↓
Llama-3.1-8B	45.46 ± 0.08	54.80 ± 0.21	60.71 ± 0.77	56.39 ± 0.19	57.71 ± 0.54	25.61 ± 0.24	94.61 ± 0.30	27.2
Llama-3.3-70B	$\mathbf{46.00 \pm 0.19}$	$\mathbf{54.91 \pm 0.03}$	$\mathbf{60.37 \pm 0.61}$	$\mathbf{56.70 \pm 0.29}$	$\mathbf{57.73 \pm 0.96}$	$\mathbf{26.13 \pm 0.18}$	$\mathbf{94.78 \pm 0.09}$	**11.0**

Table 10: Impact of searcher model scale on TransNAS-Bench-101. The performance shows Mean $\pm$ Std across 5 seeds. The 70B model consistently outperforms the 8B model, with the performance gap being most significant in tasks requiring complex structural reasoning (e.g., Jigsaw, Room Layout).

Variant	**Topological Div. (%)**
Base	71.4 ± 2.4
+ PKB	69.5 ± 3.4
+ CoT	51.5 ± 2.3
PKB+CoT	50.9 ± 3.0
Full	50.1 ± 2.2

Table 11: Topological diversity across ablation variants.