From Memorization to Creativity: LLM as a Designer of Novel Neural Architectures

Waleed Khalid Dmitry Ignatov* Radu Timofte
Computer Vision Lab, CAIDAS & IFI, University of Würzburg, Germany · *dmytro.ignatov@uni-wuerzburg.de

Abstract

Large language models (LLMs) excel in program synthesis, yet their capacity for neural architecture design—balancing syntactic reliability, performance, and structural novelty—remains underexplored. We present a closed-loop architecture synthesis pipeline within the NNGPT framework, in which a code-oriented LLM evolves over 22 supervised fine-tuning cycles. At each cycle, the LLM synthesizes PyTorch convolutional networks, validated via low-fidelity performance signals and filtered via a MinHash–Jaccard criterion to prevent structural redundancy before being incorporated into the LEMUR dataset. High-performing candidates with novel architectures are converted into prompt–code pairs for parameter-efficient LoRA fine-tuning. This feedback loop drives a measurable distributional shift, progressively internalizing empirical architectural priors such that valid and high-performing outputs evolve from scarce to dominant across cycles. On CIFAR-10, the valid generation rate stabilizes at 50.6% (peaking at 74.5%), mean first-epoch accuracy rises from 28.1% to 51.0%, and candidates exceeding 40% accuracy grow from 2.0% to 96.8%. Cross-dataset transfer to CIFAR-100 and SVHN confirms that improved validity, shifted accuracy distributions, and sustained novelty generalize across benchmarks of varying difficulty and visual domain. Across 22 cycles, 455 unique architectures absent from the original corpus are admitted under the novelty filter. By grounding synthesis in execution feedback and novelty filtering, we demonstrate that iterative self-supervised fine-tuning reshapes an LLM into a task-specialized architectural prior—improving generation reliability, proxy performance, and structural diversity—offering a reproducible, annotation-free alternative to hand-crafted search spaces. All code artifacts are publicly released under https://github.com/ABrain-One, including the code retrieval engine (NN-RAG), prompt generation (NN-Dup), and LLM fine-tuning (NNGPT) pipelines, as well as the generated models (NN Dataset); the fine-tuned LLM is available at https://huggingface.co/ABrain/NNGPT-UniqueArch-Rag.

1. Introduction

Designing effective neural network architectures remains a central bottleneck in modern deep learning. Neural Architecture Search (NAS) emerged to automate this process through reinforcement learning, evolutionary algorithms, and differentiable optimization [16, 31, 34]. While successful, traditional NAS often incurs prohibitive computational costs. Consequently, the CIFAR-10 classification task [20] has become a canonical benchmark for evaluating these automated design strategies.

In parallel, large language models (LLMs) have revolutionized program synthesis, enabling the generation of complex source code from natural-language instructions. Recent frameworks like *LLMatic* and *LEMONADE* have begun leveraging LLMs to emit full network definitions, demonstrating their potential as architecture generators [22, 24]. However, existing studies primarily focus on final model accuracy and search efficiency, offering limited insight into the generator's reliability. Specifically, it remains unclear how LLM-driven synthesis evolves under iterative refinement, particularly regarding syntactic *validity*, structural *novelty*, and the ability to maintain diversity as the model specializes.

In this work, we therefore consider an LLM purely as an *architecture synthesizer* and address the following central question: if we repeatedly fine-tune an LLM on its own successful generations, does its ability to produce valid, high-quality, and structurally novel network architectures measurably improve over time? Rather than optimizing for final test accuracy after long training runs, we deliberately adopt a low-cost performance proxy: the classification accuracy achieved after a *single* training epoch on CIFAR-10 [20]. This early-epoch accuracy is inexpensive to obtain and directly reflects how well the generated architectures support fast initial learning. At the same time, we treat structural uniqueness as a first-class objective, because practical NAS workflows benefit not only from strong individual models but also from diverse candidates that explore different regions of the design space [16, 31].

Concretely, we execute an LLM-driven synthesis loop

over 22 cycles where candidate architectures are filtered for compilation validity, trained for a single epoch, and subjected to MinHash–Jaccard novelty analysis. Our results demonstrate that this iterative generate–evaluate–select–fine-tune process, guided by low-fidelity signals, produces a pronounced upward shift in the first-epoch accuracy distribution across cycles, while maintaining significant structural diversity as measured by code-level novelty filtering. The valid generation rate, though not monotonic, stabilizes at levels substantially above early-cycle baselines for much of the run, effectively reshaping the LLM into a task-specialized architectural prior.

It is important to delineate the scope of this study: we do *not* propose a complete NAS method competing on final test accuracy against established search algorithms, nor do we claim that the generated architectures surpass hand-designed baselines after full training. Instead, we study *the generator itself*—how its output distribution over architectures changes under iterative self-refinement—using first-epoch accuracy as a low-cost, interpretable proxy. The contribution is therefore a characterization of LLM-as-generator dynamics, providing a foundation upon which future work can build by coupling the refined generator with downstream optimization.

In summary, this work advances the intersection of automated program synthesis and neural architecture design through four primary contributions:

1. We establish an LLM-driven synthesis framework that treats the generator as a trainable architectural prior, optimizing for a triad of objectives: syntactic validity, early-epoch performance, and structural novelty.
2. We introduce a code-level novelty filter utilizing MinHash–Jaccard similarity to programmatically ensure meaningful design-space expansion.
3. We provide a 22-cycle longitudinal analysis demonstrating that iterative fine-tuning induces a substantial distributional shift toward higher-performing architectures, with net improvements in generation reliability, without sacrificing architectural diversity.
4. We validate the generality of the observed trends across three benchmarks—CIFAR-10, CIFAR-100, and SVHN—showing that the iterative refinement loop induces consistent distributional shifts regardless of task difficulty, number of classes, or visual domain.

2. Related Work

The development of NAS has significantly automated network design through reinforcement learning, evolutionary algorithms, and differentiable optimization, though often at a prohibitive computational cost [8, 16, 31]. To ameliorate the expense of repeated candidate training, the field has increasingly relied on low-fidelity proxies, including early-stopped training, learning-curve extrapolation, and training-free zero-cost signals [6, 25, 33]. While our work utilizes single-epoch accuracy as an efficient performance proxy, we diverge from traditional NAS by employing these signals to shape the behavioral priors of a generative LLM rather than optimizing within a static, handcrafted search space.

In parallel, the advent of code-capable LLMs has introduced a paradigm shift toward synthesizing complete model implementations from natural language. Frameworks such as *LLMatic* have demonstrated the efficacy of coupling LLM-driven mutation with quality-diversity search [22], while others have integrated iterative refinement to satisfy stringent deployment constraints [24]. More recent self-improving systems, such as *SEKI* and *RZ-NAS*, leverage performance-guided evolution or reflective reasoning to improve design outcomes [3, 14]. Despite these advances, existing research typically evaluates success through final search efficiency or peak accuracy. Our approach provides a distinct longitudinal perspective by explicitly characterizing the evolution of the generator itself—tracking metrics of validity, performance distribution shifts, and code-level novelty across twenty-two successive cycles of supervised fine-tuning.

Crucially, the utility of LLM-generated architectures is predicated on both functional reliability and structural diversity. While standard code generation benchmarks prioritize functional correctness and unit-test pass rates [4, 15], the structured nature of PyTorch programs necessitates a more nuanced separation between executable validity—comprising parsing, instantiation, and forward passes—and downstream learning quality. To prevent the collapse of the generator into redundant motifs or trivial rewrites, we incorporate a MinHash–Jaccard near-duplicate filter, ensuring that the fine-tuning corpus is augmented only with implementations that are both performant and structurally novel. Table 1 summarizes the key methodological and empirical distinctions between our framework and the most closely related LLM-based architecture generation methods.

3. Method

We treat a code-oriented large language model (LLM) as a stochastic generator of neural network architectures and study how its behavior changes under an iterative refinement loop. We run 22 synthesis cycles indexed by $c \in \{1, \ldots, 22\}$. In each cycle, the LLM generates candidate PyTorch models; we execute validity checks, run a fixed first-epoch training protocol to obtain a low-cost performance proxy, filter for novelty, and then fine-tune the LLM on the accepted outputs before proceeding to the next cycle. The primary evaluation is conducted on CIFAR-10 [20]; cross-dataset generalization is assessed on CIFAR-100 [20] and SVHN [23] using the same loop with dataset-appropriate accuracy thresholds (Section 3.2). Figure 1

Method	Valid Rate	Proxy Metric	Novel Filter	Iter. FT	Corpus Growth	Best Accuracy
LLMatic [22]	N/R	Final	QD arch.	No	No	$\sim$94%[a]
LEMONADE [24]	$\approx$75–85%[†]	Constr.	None	No	No	N/A[b]
SEKI [3]	N/R	Final	Repo.	Yes	N/R	97.7%[a]
RZ-NAS [14]	$\approx$98%[‡]	Zero-cost	None	No	No	N/R[c]
Ours	**50.6%**[e]	**1-ep.**	**MinHash**	**Yes**	**Yes**[f]	**64.0%**[d]

Table 1. Comparison of LLM-based architecture generation frameworks (CIFAR-10). N/R = not reported; FT = fine-tuning. [a] Fully-trained test accuracy (DARTS/NB201 search space). [b] Optimizes deployment constraints, not peak accuracy. [c] Uses zero-cost ranking; final accuracy not reported. [d] **First-epoch validation accuracy only**; not directly comparable to fully-trained results above. [e] Explicitly tracked across 22 cycles; no other method reports longitudinal validity. [f] Training corpus grows from 1,698 to 2,153 prompt–code pairs across cycles. Our framework is the only one to jointly track valid generation rate, apply a code-level novelty filter, grow the training corpus iteratively, and fine-tune on low-fidelity proxy feedback—all without manual annotations.

summarizes the generate–evaluate–select–fine-tune loop.

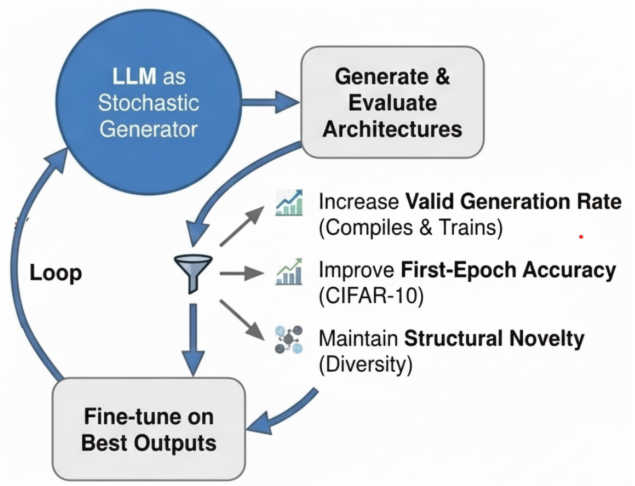

Figure 1. Overview of the iterative architecture-synthesis loop: the LLM generates architectures, candidates are evaluated and filtered (validity, first-epoch accuracy, novelty), and the LLM is fine-tuned on selected outputs.

A code-focused LLM is used as a conditional sampler over PyTorch architectures. Prompts specify CIFAR-10 classification, input/output shapes ($(N, 3, 32, 32) \rightarrow 10$ logits), and constraints on permissible operations (standard conv/pool/norm/activations; no pretrained weights). A parameter budget of at most 500,000 parameters is imposed. Each generated model must implement a fixed API contract (class `Net(nn.Module)` with `forward`, `train_setup`, `learn`); the full prompt specification is provided in Suppl. §I. The prompt template, decoding configuration, and maximum generation length are held fixed across all cycles to avoid prompt drift; consequently, changes in outputs are attributable to training data and fine-tuning.

We initialize supervision using the LEMUR Neural Network Dataset [9, 19], a broad collection of performance-annotated neural network implementations spanning high-capacity and edge-optimized designs [5, 29], drawing on prior work in LLM-driven architectural synthesis within the NNGPT ecosystem [10, 13, 17, 18, 21, 26, 28, 30]. After MinHash/LSH deduplication and chat-format conversion (Suppl. §A–C), the corpus yields 849 unique architectures paired with two task descriptions, producing 1,698 prompt–code training examples.

For every successfully parsed and trained candidate, we compute two Jaccard similarities over token-level shingles of the model code. The first, J_{train}, measures similarity to the deduplicated supervised training set; the second, $J_{\text{gen}}^{(c)}$, measures similarity to the set of all code samples generated earlier in the same cycle c. Formally, let S_{train} denote the collection of shingle sets corresponding to all models in the current supervised corpus, and let $S_{\text{gen}}^{(c)}$ denote the collection for all models generated in cycle c prior to the current candidate. For a new candidate with shingle set A:

$$J_{\text{train}}(A) = \max_{B \in S_{\text{train}}} J(A, B), \qquad J_{\text{gen}}^{(c)}(A) = \max_{B \in S_{\text{gen}}^{(c)}} J(A, B), \tag{1}$$

estimated via MinHash signatures and LSH indexing. If either similarity exceeds a near-duplicate threshold $\tau \approx 0.9$, the candidate is rejected. Sampling continues until a valid architecture is found that is sufficiently dissimilar to both the current training set and earlier generations in the same cycle.

During the 22-cycle loop, the corpus is augmented with self-generated models. At the end of each cycle, candidate models are considered for inclusion if they (i) compile and train, (ii) exceed a first-epoch accuracy threshold on the target dataset (40% for CIFAR-10, 20% for CIFAR-100, 70% for SVHN), and (iii) pass the near-duplicate filter. This process adds 455 unique high-accuracy models across 22 cycles, expanding the training corpus from 1,698 to 2,153 examples (Table 2).

Source	N models
LEMUR (deduplicated, train split)	1,698
Self-generated ($\geq 40\%$ acc., novel)	455
Total used for training by cycle 22	2,153

Table 2. Supervised training corpus by the end of cycle 22 (CIFAR-10 setting).

Each generated code snippet is executed in an isolated environment. Candidates are rejected if Python parsing fails, if `Net` cannot be instantiated, or if a dummy forward pass raises an exception. All remaining candidates are subjected to a standardized training protocol on the target dataset, with hyperparameters—including the training/validation split, input resolution, optimization schedule, and batch size—held constant. Advanced data augmentation techniques derived from Aboudeshish et al. [1] are utilized to maintain a consistent baseline. Implementation details for MinHash/LSH near-duplicate detection are provided in Suppl. §A.

For each syntactically valid model m, the top-1 validation accuracy after a single epoch, $A(m)$, is recorded as the primary performance signal; a comparison with zero-cost proxy alternatives is given in Suppl. §E. Per-cycle summaries include the valid generation rate $p^{(c)}_{\text{valid}} = N^{(c)}_{\text{valid}}/N^{(c)}_{\text{gen}}$, sample mean and standard deviation of first-epoch accuracy, and the proportion of models exceeding a fixed threshold. We report t-based 95% confidence intervals for means and Wilson score intervals [32] for proportions; formal definitions are given in Suppl. §K.

3.1. Fine-tuning and Generation Hyperparameters

Fine-tuning is performed using DeepSeek-Coder-7B-Instruct-v1.5 [11] adapted with LoRA [12] (rank 32, applied to all attention and MLP projections). In each cycle, the model is fine-tuned for 5 epochs on chat-format prompt–response pairs; the only changing factor across cycles is the growing training set. Generation uses fixed decoding (temperature 0.20, top-k 50, nucleus p 0.9) across all cycles, isolating the effect of iterative fine-tuning and data growth. Full hyperparameter specifications for both fine-tuning and decoding are provided in Suppl. §J.

While the initial cycle employed a fixed generation budget of $N_{gen} = 50$ candidates, a dynamic sampling strategy was introduced from cycle 11 onward. To maintain a consistent influx of at least 30 structurally unique, above-threshold architectures per cycle for LoRA corpus augmentation, N_{gen} was scaled adaptively in response to the evolving acceptance rate imposed by the MinHash–Jaccard novelty filter. The compute budget is detailed in Suppl. §L.

The choice of 22 cycles reflects the point at which both mean accuracy and above-threshold fraction plateau, indicating diminishing returns from further fine-tuning; a formal analysis is provided in Suppl. §D.

3.2. Cross-Dataset Evaluation Setup

To evaluate whether the observed improvements are specific to CIFAR-10 or reflect a more general property of the iterative synthesis loop, we replicate the full 22-cycle procedure on CIFAR-100 [20] and SVHN [23]. CIFAR-100 shares the same 32×32 RGB format but increases output classes from 10 to 100; SVHN is a 10-class digit recognition task using street-view images, sharing the class count with CIFAR-10 but differing in visual domain. The prompt template is adapted only in the user message; the system message, decoding configuration, LoRA hyperparameters, novelty filtering, and single-epoch evaluation protocol remain identical. The sole dataset-specific parameter is the accuracy threshold for corpus inclusion: 40% for CIFAR-10, 20% for CIFAR-100, and 70% for SVHN. Each dataset maintains its own independent training corpus and cycle history.

4. Results

The 22-cycle synthesis loop is evaluated using the metrics defined in Section 3. We first present CIFAR-10 results (Sections 4.1–4.2), then report cross-dataset generalization (Section 4.3).

4.1. CIFAR-10: Primary Results

Representative checkpoints are reported in Table 3, while Figure 3 summarizes the joint evolution of reliability, proxy performance, novelty-based selection, and training-set growth. Additional per-cycle plots are provided in Suppl. §G.

Cycle	Valid (%)	Best (%)	Mean (%)	$\geq 40\%$ (%)	Unique models	Total train
1	44.0	47.78	28.06	2.04	1	1698
5	32.0	49.13	29.88	6.82	9	1724
10	53.8	55.48	37.70	38.04	18	1785
15	**66.8**	58.60	47.40	80.70	34	1911
18	59.1	**63.98**	**50.99**	**96.81**	**38**	2025
22	41.8	57.62	49.48	92.86	30	**2153**

Table 3. Selected cycle statistics on CIFAR-10: valid generation rate, best and mean first-epoch accuracy, proportion of models with accuracy $\geq 40\%$, number of structurally unique models selected, and cumulative training-set size.

Validity. Cycle 1 begins at 44.0% validity (22/50). Following early fluctuations in the low-to-mid 30% range (cycles 2–5), validity increases and reaches a peak of 74.5%

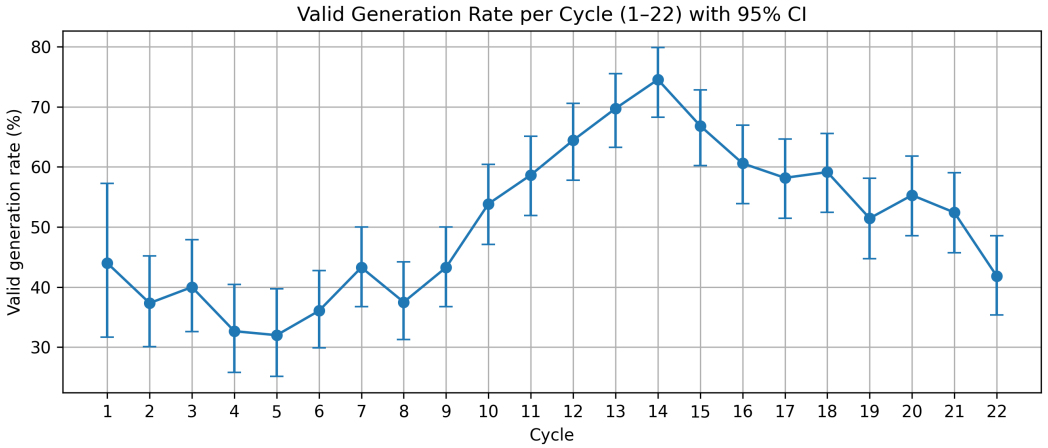

Figure 2. Valid generation rate per cycle (1–22) with Wilson 95% confidence intervals (CIFAR-10).

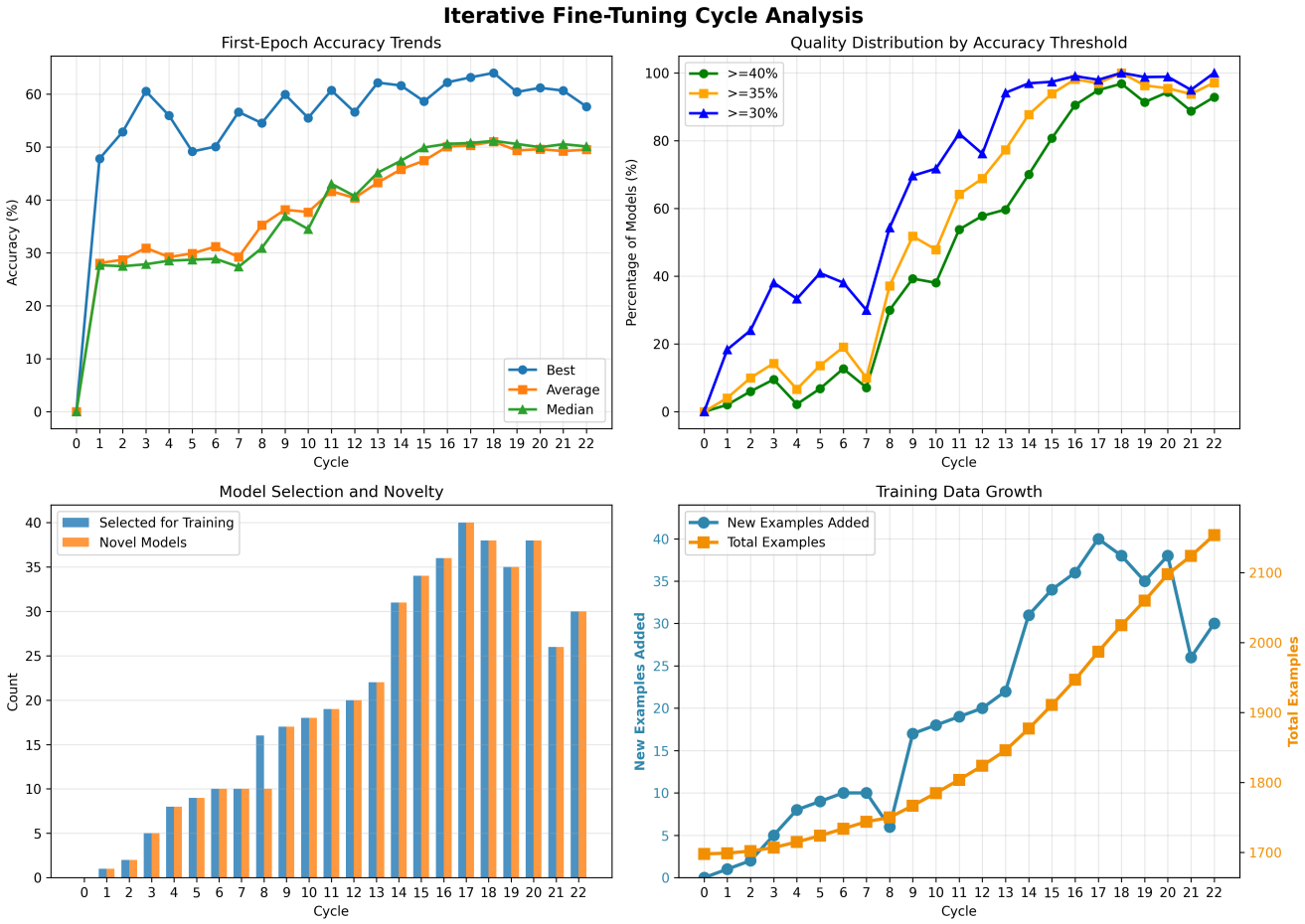

Figure 3. Overall analysis of the 22 fine-tuning cycles on CIFAR-10: (top-left) first-epoch accuracy trends; (top-right) quality distribution by accuracy threshold; (bottom-left) model selection and novelty; (bottom-right) training-data growth.

in cycle 14. In later cycles, the valid rate stabilizes mostly between 51% and 60%, before dropping to 41.8% in cycle 22. Figure 2 shows the per-cycle trajectory together with Wilson-score 95% confidence intervals; across all 22 cycles, the mean valid generation rate is 50.6% with a 95% confidence interval of [45.0%, 56.1%]. Notably, the validity trajectory is non-monotonic: the late-cycle decline likely reflects increased specialization of the generator toward high-

accuracy architectures at the expense of syntactic diversity, a trade-off consistent with the distributional narrowing documented in iterative self-training settings [27].

Proxy quality. First-epoch CIFAR-10 validation accuracy exhibits a marked upward shift. In cycle 1, the best model reaches 47.78% and the mean accuracy is 28.06% (median 27.67%). By cycle 10, the best improves to 55.48% and the mean rises to 37.70%. The strongest checkpoint occurs at cycle 18, where the best reaches 63.98% and the mean reaches 50.99% (median 51.15%); cycle 22 remains comparably strong (best 57.62%, mean 49.48%, median 50.10%). The monotonic improvement in mean accuracy (28.06% $\rightarrow$ 49.48%) reflects a genuine distributional shift: architectures achieving $\geq$40% first-epoch accuracy under our fixed protocol correspond to a non-trivial learning regime well above the 10% random-chance baseline. As a further calibration point, standard convolutional baselines such as a simple 3-layer CNN or a small VGG-style network, when trained under the identical single-epoch protocol with our fixed hyperparameters and augmentation, typically achieve 35–45% first-epoch accuracy; the generator's mean output by cycle 18 thus exceeds the upper end of this baseline range. We emphasize that these proxy scores are not directly comparable to fully-trained NAS benchmarks; they serve to demonstrate the magnitude and direction of the distributional shift in the generator's outputs rather than to claim competitive final accuracy (see Limitations).

Above-threshold mass. Only 2.04% of trained models exceed 40% in cycle 1, increasing to 38.04% by cycle 10. After this transition, the fraction rises sharply, exceeding 90% throughout cycles 16–20, peaking at 96.81% in cycle 18, and ending at 92.86% in cycle 22. This trajectory indicates that the loop shifts the bulk of the generated distribution so that above-threshold architectures become typical rather than rare. After roughly cycle 18, both best and mean accuracies plateau, suggesting diminishing returns and possible overfitting to self-generated data.

4.2. CIFAR-10: Novelty and Corpus Growth

Structurally novel, above-threshold architectures continue to be admitted across cycles under the MinHash–Jaccard novelty constraint. Across all cycles, 459 structurally novel architectures are discovered and 455 are ultimately added to the supervised fine-tuning corpus. The cumulative training set grows steadily from 1,698 prompt–code pairs at initialization to 2,153 by cycle 22, with intermediate growth visible at representative checkpoints (e.g., 1,785 by cycle 10, 2,025 by cycle 18). Figure 3 visualizes these coupled trends. A qualitative inspection of representative generated architectures (Suppl. §H) confirms that this code-level novelty corresponds to meaningful structural diversity: the admitted models range from ultra-compact single-block designs to deep multi-stage networks with residual identity shortcuts and dropout regularization, exhibiting both combinatorial and operational variety across design families.

4.3. Cross-Dataset Generalization

To assess whether the trends documented above reflect a general property of the iterative synthesis loop, we repeat the full 22-cycle procedure on CIFAR-100 and SVHN under the setup described in Section 3.2.

CIFAR-100. Table 4 reports representative cycle statistics. The overall trajectory mirrors the CIFAR-10 results: the valid generation rate rises from 42.0% in cycle 1 to a peak of 63.5% in cycle 15 before settling at 46.2% by cycle 22, while mean first-epoch accuracy increases from 9.0% to 20.5%. The proportion of valid models exceeding the 20% threshold grows from 1.5% to 63.0%, confirming that the iterative loop induces a distributional shift toward faster-learning architectures even under a 100-class setting. The best single-epoch accuracy reaches 32.5% by cycle 18. Across cycles, the training corpus expands from 1,698 to 2,038 prompt–code pairs.

Cycle	Valid (%)	Best (%)	Mean (%)	$\geq$ 20% (%)	Unique models	Total train
1	42.0	20.0	9.0	1.5	1	1698
5	29.3	21.5	10.2	3.0	6	1708
10	50.0	25.8	14.6	16.0	14	1748
15	**63.5**	29.5	18.8	52.0	26	1832
18	56.7	**32.5**	**21.2**	**67.0**	**30**	1906
22	46.2	31.0	20.5	63.0	24	**2038**

Table 4. Cycle statistics for CIFAR-100: valid generation rate, best and mean first-epoch accuracy, proportion of models with accuracy $\geq$ 20%, structurally unique models selected, and cumulative training-set size.

SVHN. Table 6 presents the corresponding results. Because SVHN is an easier recognition task, the inclusion threshold is set to 70%. The valid generation rate follows a qualitatively similar arc (46.0% $\rightarrow$ peak at 69.2% $\rightarrow$ 48.1% at cycle 22), while mean first-epoch accuracy rises from 51.0% to 70.5%. The fraction of models exceeding 70% grows from 18.0% to 77.0%, and the best model reaches 84.5% accuracy by cycle 18. The per-cycle count of admitted novel models is non-monotonic but subject to statistically insignificant fluctuations, reflecting the interplay between accuracy specialization and the stricter novelty constraint as the corpus grows. The training corpus expands to 2,110 examples by cycle 22.

Dataset	Threshold τ	Cycle 1			Cycle 22		
		Valid (%)	Mean acc. (%)	$\geq \tau$ (%)	Valid (%)	Mean acc. (%)	$\geq \tau$ (%)
CIFAR-10	40%	44.0	28.06	2.04	41.8	**49.48**	92.86
CIFAR-100	20%	42.0	9.0	1.5	46.2	**20.5**	63.0
SVHN	70%	46.0	51.0	18.0	48.1	**70.5**	77.0

Table 5. Cross-dataset comparison at cycle 1 and cycle 22. The quality threshold τ is adapted to task difficulty. All three benchmarks exhibit a very similar qualitative pattern: approximately stable valid generation rates, a pronounced increase in mean first-epoch accuracy, and a large increase in the proportion of models exceeding the task-specific threshold.

Cycle	Valid (%)	Best (%)	Mean (%)	$\geq 70\%$ (%)	Unique models	Total train
1	46.0	73.0	51.0	18.0	2	1698
5	34.7	74.5	53.0	24.0	8	1718
10	57.7	79.5	62.0	54.0	18	1778
15	**69.2**	83.0	69.0	70.0	**33**	1890
18	61.5	**84.5**	**71.5**	**79.0**	32	1965
22	48.1	83.5	70.5	77.0	28	**2110**

Table 6. Cycle statistics for SVHN: valid generation rate, best and mean first-epoch accuracy, proportion of models with accuracy $\geq 70\%$, structurally unique models selected, and cumulative training-set size.

Cross-dataset comparison. Table 5 directly compares the three datasets at cycle 1 and cycle 22. Despite differences in absolute accuracy levels, all three benchmarks exhibit a very similar qualitative pattern: (i) the valid generation rate remains comparable across datasets and cycles, confirming that validity is primarily a property of the generator's code-synthesis capability; (ii) mean first-epoch accuracy approximately doubles over the 22-cycle run in each case; and (iii) the proportion of models exceeding the dataset-specific quality threshold increases by one to two orders of magnitude. These consistent trends across datasets varying in number of classes (10 vs. 100), visual domain (natural images vs. street-view digits), and difficulty level provide evidence that the iterative procedure induces a robust distributional shift that is not an artifact of a single benchmark.

4.4. Ablation Study

The proposed synthesis loop combines three components: (i) a MinHash–Jaccard novelty filter, (ii) a first-epoch accuracy threshold of 40%, and (iii) iterative LoRA fine-tuning. To isolate the contribution of each, three ablation variants are considered, each removing one ingredient. Across all ablations, the prompt template, decoding configuration, and evaluation protocol are kept identical. Table 7 summarizes the results; detailed per-ablation analyses are provided in Suppl. §F.

Removing the novelty filter preserves accuracy trends but roughly halves the number of genuinely distinct architectures admitted (455 $\rightarrow$ 220), as the corpus accumulates near-duplicate motifs. Removing the accuracy threshold maintains novelty but weakens the distributional shift: the above-40% fraction drops from 51.1% to 34.0% and mean accuracy falls by 3.8 percentage points, since low-performing models are promoted into the training corpus. Removing iterative fine-tuning entirely (non-iterative baseline) yields the weakest outcome: the valid generation rate, mean accuracy, and above-threshold fraction all remain at the level observed in cycle 1, and only a single novel architecture is discovered because the feedback loop is absent. Taken together, the ablations demonstrate that all three components interact synergistically: the novelty filter sustains exploration, the accuracy threshold steers corpus quality, and iterative fine-tuning closes the feedback loop that enables progressive improvement.

5. Conclusion and Future Work

This work examines how a code-oriented large language model behaves when placed at the center of an iterative architecture-synthesis loop. Rather than treating the LLM as a fixed component within a neural architecture search pipeline, its output distribution over architectures is tracked across 22 supervised fine-tuning cycles using its own high-quality, structurally novel generations. Under a controlled image-classification setting, the generate–evaluate–select–fine-tune procedure induces a pronounced shift in this distribution, increasing both the likelihood of producing executable models and the early-epoch performance of sampled networks, while retaining non-trivial structural diversity as measured by a code-level novelty filter.

Across cycles, the generator moves from an initial regime in which valid, rapidly learning architectures are relatively uncommon to one in which they constitute the majority of outputs. The distribution of first-epoch accuracies shifts upward: the fraction of models exceeding a moderate performance threshold rises from a small minority early on to a large majority in later cycles. A second outcome is the sustained admission of code-level novel architectures via the MinHash–Jaccard novelty criterion, though we note that

Method	Valid rate (%)	Mean acc. (%)	$\geq 40\%$ acc. (%)	Novel models
Full method	50.6 [45.0, 56.1]	42.3 [41.8, 42.8]	51.1	455
No novelty filter	52.0 [46.0, 57.9]	42.0 [41.4, 42.6]	50.0	220
No accuracy threshold	51.0 [45.1, 56.8]	38.5 [37.8, 39.3]	34.0	470
No iteration	44.0 [31.2, 57.7]	28.06 [5.9, 50.2]	4.55	1

Table 7. Ablation study of the synthesis loop (CIFAR-10). 95% confidence intervals are reported in brackets (Wilson score for proportions, t-based for means). The full method combines all three components; each ablation removes one.

text-level novelty does not guarantee functional novelty (see Limitations). A third outcome is the demonstrated generality of these trends: cross-dataset experiments on CIFAR-100 and SVHN confirm that the same qualitative pattern holds across benchmarks varying in number of classes, visual domain, and task difficulty, suggesting that the iterative loop shapes the generator's code-synthesis priors rather than exploiting dataset-specific artifacts.

Several important questions remain open: whether the proxy ranking under single-epoch evaluation is preserved after full training, whether the discovered architectures are competitive with established NAS methods under matched budgets, and whether the structural diversity documented at the code level (Suppl. §H) extends to a broader set of generated models across cycles. These constitute concrete directions for future work.

Additional future directions include integrating the refined generator with explicit optimization frameworks (e.g., LLMatic, SEKI, or RZ-NAS) to leverage the learned distribution as an architectural prior for downstream search [3, 14, 22]; extending evaluation to higher-resolution datasets and non-classification tasks; incorporating more granular feedback signals such as performance-weighted sampling or reinforcement learning; and integrating multi-objective constraints (e.g., parameter count and latency) to internalize accuracy–efficiency trade-offs critical to edge-oriented NAS [2, 24].

6. Limitations

Despite the observed improvements, several limitations remain and should be weighed when interpreting the results.

Benchmark scope. Although we evaluate on three datasets (CIFAR-10, CIFAR-100, SVHN), all three are low-resolution (32×32) image classification benchmarks. It remains unclear how well the observed trends transfer to higher-resolution images, non-visual domains, or tasks such as segmentation or detection.

Proxy signal. First-epoch validation accuracy serves as the sole performance signal. While supported by recent work on early-discarding strategies [7] and contextualized by baseline calibration in Section 4.1, validating the proxy ranking under full training schedules remains an important next step (Suppl. §E).

Scope of comparisons. This study characterizes the generator's evolution rather than proposing a complete NAS pipeline; fully-trained accuracy comparisons against established NAS methods are therefore not included. Table 1 positions our proxy-level results alongside reported fully-trained accuracies from related LLM-based methods to provide context.

Qualitative coverage. A preliminary qualitative inspection of representative architectures (Suppl. §H) confirms structural diversity spanning compact single-block designs to deep residual networks. A more systematic analysis across a larger sample would further strengthen this evidence.

Fine-tuning rigidity. LoRA adaptation is performed with fixed hyperparameters, and the acceptance threshold is held constant; the observed plateauing in later cycles suggests that alternative curricula or additional regularization could be beneficial.

Text-level novelty definition. Novelty is defined over token-level shingles of source code rather than on explicit computation graphs, meaning functionally equivalent models expressed in different styles may be treated as novel, while structurally distinct models with similar token patterns may not.

7. Ethical Considerations

This study employs DeepSeek-Coder-7B-Instruct-v1.5 within a closed, controlled, iterative architecture-synthesis framework and is strictly methodological in nature. It does not involve human participants, personal data, or end-user-facing deployment. The initial training corpus (LEMUR) consists exclusively of source code and technical metadata containing no personally identifiable information. Any real-world deployment of LLM-generated code would necessitate comprehensive security reviews. Full details on data usage, security considerations, and reproducibility measures are provided in Suppl. §M.

Acknowledgments. This work was partially supported by the Alexander von Humboldt Foundation.

References

[1] Nada Aboudeshish, Dmitry Ignatov, and Radu Timofte. AUGMENTGEST: Can random data cropping augmentation boost gesture recognition performance? *arXiv preprint*, arXiv:2506.07216, 2025. 4

[2] Jesús-Arnulfo Barradas-Palmeros, Carlos-Alberto López-Herrera, Erick Mezura-Montes, Héctor-Gabriel Acosta-Mesa, and Ana-Leticia López-Lobato. Testing neural architecture search efficient evaluation methods in deepga. *Mathematical and Computational Applications*, 30(4):74, 2025. 8

[3] Zicheng Cai, Yaohua Tang, Yutao Lai, Hua Wang, Zhi Chen, and Hao Chen. SEKI: Self-evolution and knowledge inspiration based neural architecture search via large language models. *arXiv preprint*, arXiv:2502.20422, 2025. 2, 3, 8

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, et al. Evaluating large language models trained on code. *arXiv preprint*, 2021. 2

[5] Saif U Din, Muhammad Ahsan Hussain, Mohsin Ikram, Dmitry Ignatov, and Radu Timofte. AI on the edge: An automated pipeline for PyTorch-to-Android deployment and benchmarking. *Preprints*, 2025. 3

[6] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3460–3468, 2015. 2

[7] Romain Egele, Felix Mohr, Tom Viering, and Prasanna Balaprakash. The unreasonable effectiveness of early discarding after one epoch in neural network hyperparameter optimization. *Neurocomputing*, 597:127964, 2024. 8

[8] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019. 2

[9] Arash Torabi Goodarzi, Roman Kochnev, Waleed Khalid, Furui Qin, Tolgay Atinc Uzun, Yashkumar Sanjaybhai Dhameliya, Yash Kanubhai Kathiriya, Zofia Antonina Bentyn, Dmitry Ignatov, and Radu Timofte. LEMUR neural network dataset: Towards seamless AutoML. *arXiv preprint*, arXiv:2504.10552, 2025. 3

[10] Xiaojie Gu, Dmitry Ignatov, and Radu Timofte. Resource-efficient iterative LLM-based NAS with feedback memory. *arXiv preprint*, arXiv:2603.12091, 2026. 3

[11] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, et al. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *arXiv preprint*, arXiv:2401.14196, 2024. 4

[12] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022. 4

[13] Krunal Jesani, Dmitry Ignatov, and Radu Timofte. LLM as a neural architect: Controlled generation of image captioning models under strict API contracts. *arXiv preprint*, arXiv:2512.14706, 2025. 3

[14] Zipeng Ji, Guanghui Zhu, Chunfeng Yuan, and Yihua Huang. RZ-NAS: Enhancing llm-guided neural architecture search via reflective zero-cost strategy. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025. To appear. 2, 3, 8

[15] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. *arXiv preprint*, 2024. 2

[16] Jeon-Seong Kang, JinKyu Kang, Jung-Jun Kim, Kwang-Woo Jeon, Hyun-Joon Chung, and Byung-Hoon Park. Neural architecture search survey: A computer vision perspective. *Sensors*, 23(3):1713, 2023. 1, 2

[17] Waleed Khalid, Dmitry Ignatov, and Radu Timofte. A retrieval-augmented generation approach to extracting algorithmic logic from neural networks. *arXiv preprint*, arXiv:2512.04329, 2025. 3

[18] Roman Kochnev, Arash Torabi Goodarzi, Zofia Antonina Bentyn, Dmitry Ignatov, and Radu Timofte. Optuna vs Code Llama: Are LLMs a New Paradigm for Hyperparameter Tuning? In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 5664–5674, 2025. 3

[19] Roman Kochnev, Waleed Khalid, Tolgay Atinc Uzun, Xi Zhang, Yashkumar Sanjaybhai Dhameliya, Furui Qin, Chandini Vysyaraju, Raghuvir Duvvuri, Avi Goyal, Dmitry Ignatov, and Radu Timofte. NNGPT: Rethinking AutoML with large language models. *arXiv preprint*, arXiv:2511.2033, 2025. 3

[20] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. Technical Report. 1, 2, 4

[21] Yash Mittal, Dmitry Ignatov, and Radu Timofte. Preparation of fractal-inspired computational architectures for advanced large language model analysis. *arXiv preprint*, arXiv:2511.07329, 2025. 3

[22] Muhammad U. Nasir, Sam Earle, Christopher Cleghorn, Steven James, and Julian Togelius. LLMatic: Neural architecture search via large language models and quality-diversity optimization. *CoRR*, abs/2306.01102, 2023. Also in GECCO 2024. 1, 2, 3, 8

[23] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011. 2, 4

[24] Md Hafizur Rahman, Zafaryab Haider, and Prabuddha Chakraborty. An automated multi parameter neural architecture discovery framework using ChatGPT in the backend. *Scientific Reports*, 15(16871), 2025. 1, 2, 3, 8

[25] Binxin Ru, Rui Shu, Xinyi Dong, et al. Speedy performance estimation for neural architecture search. In *Proceedings of the International Conference on Machine Learning (ICML) Workshop on AutoML*, 2020. 2

[26] Usha Shrestha, Dmitry Ignatov, and Radu Timofte. From brute force to semantic insight: Performance-guided data transformation design with LLMs. *arXiv preprint*, 2026. 3

[27] Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicholas Papernot, Robert D. Anderson, Yoav Ganin, and Ross J. An-

derson. AI models collapse when trained on recursively generated data. *Nature*, 631(8022):755–759, 2024. 6

[28] Tolgay Atinc Uzun, Dmitry Ignatov, and Radu Timofte. Closed-loop LLM discovery of non-standard channel priors in vision models. *arXiv preprint*, arXiv:2601.08517, 2026. 3

[29] Tolgay Atincand Uzun, Waleed Khalid, Saif U Din, Sai Revanth Mulukuledu, Akashdeep Singh, Chandini Vysyaraju, Raghuvir Duvvuri, Avi Goyal, Yashkumar Rajeshbhai Lukhi, Ahsan Hussain, Krunal Jesani, Usha Shrestha, Yash Mittal, Roman Kochnev, Pritam Kadam, Mohsin Ikram, Harsh Rameshbhai Moradiya, Alice Arslanian, Dmitry Ignatov, and Radu Timofte. LEMUR 2: Unlocking neural network diversity for AI. *arXiv preprint*, 2026. 3

[30] Chandini Vysyaraju, Raghuvir Duvvuri, Avi Goyal, Dmitry Ignatov, and Radu Timofte. Enhancing LLM-based neural network generation: Few-shot prompting and efficient validation for automated architecture design. *arXiv preprint*, arXiv:2512.24120, 2025. 3

[31] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*, 2023. 1, 2

[32] Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927. 4

[33] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2020. 2

[34] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017. 1