

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

Abstract

Large language models (LLMs) excel in program synthesis, yet their ability to autonomously navigate neural architecture design—balancing syntactic reliability, performance, and structural novelty—remains underexplored. We address this by placing a code-oriented LLM within a closed-loop synthesis framework, analyzing its evolution over 22 supervised fine-tuning cycles. The model synthesizes PyTorch convolutional networks which are validated, evaluated via low-fidelity performance signals (single-epoch accuracy), and filtered using a MinHash–Jaccard criterion to prevent structural redundancy. High-performing, novel architectures are converted into prompt–code pairs for iterative fine-tuning via parameter-efficient LoRA adaptation, initialized from the LEMUR dataset. Across cycles, the LLM internalizes empirical architectural priors, becoming a robust generator. The valid generation rate stabilizes at 50.6% (peaking at 74.5%), while mean first-epoch accuracy rises from 28.06% to 50.99%, and the fraction of candidates exceeding 40% accuracy grows from 2.04% to 96.81%. Analyses confirm the model moves beyond replicating existing motifs, synthesizing 455 high-performing architectures absent from the original corpus. By grounding code synthesis in execution feedback, this work provides a scalable blueprint for transforming stochastic generators into autonomous, performance-driven neural designers, establishing that LLMs can internalize empirical, non-textual rewards to transcend their training data.

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 (Zoph and Le, 2017; White et al.,

2023; Kang et al., 2023). While successful, traditional NAS often incurs prohibitive computational costs. Consequently, the CIFAR-10 classification task (Krizhevsky, 2009) 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 (Nasir et al., 2023; Rahman et al., 2025). 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 (Krizhevsky, 2009). 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 (White et al., 2023; Kang et al., 2023).

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, monotonically improves both generator reliability and model performance while maintaining significant structural diversity, effectively reshaping the LLM into a robust architectural prior.

In summary, This work advances the intersection of automated program synthesis and neural architecture design through three primary contributions. First, 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. Second, we introduce a code-level novelty filter utilizing MinHash–Jaccard similarity to programmatically ensure meaningful design-space expansion. Third, we provide a 22-cycle longitudinal analysis demonstrating that iterative fine-tuning significantly enhances generation reliability and model quality without sacrificing architectural diversity.

2 Related Work

The development of Neural Architecture Search (NAS) has significantly automated network design through reinforcement learning, evolutionary algorithms, and differentiable optimization, though often at a prohibitive computational cost (Elsken et al., 2019; White et al., 2023; Kang et al., 2023). 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 (Domhan et al., 2015; Ru et al., 2020; Zela et al., 2020a). 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 Large Language Model (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 (Nasir et al., 2023), while others have integrated iterative refinement to satisfy stringent deployment constraints (Rahman et al., 2025). More recent self-improving systems, such as *SEKI* and *RZ-NAS*, leverage performance-guided evolution or reflective reasoning to improve design outcomes (Cai et al., 2025; Ji et al., 2025). 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 (Chen et al., 2021; Jiang et al., 2024), 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. This ensures that the fine-tuning corpus is augmented only with implementations that are both performant and structurally novel, fostering a diverse architectural prior that transcends simple memorization of existing training samples.

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 $c \in \{1, \dots, 22\}$. In each cycle, the LLM generates candidate PyTorch models; we execute validity checks, run a fixed first-epoch CIFAR-10 (Krizhevsky, 2009) 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. Figure 1 summarizes the generate–evaluate–select–fine-tune loop.

A code-focused LLM is used as a conditional sampler over PyTorch architectures. Prompts specify CIFAR-10 classification, input/output

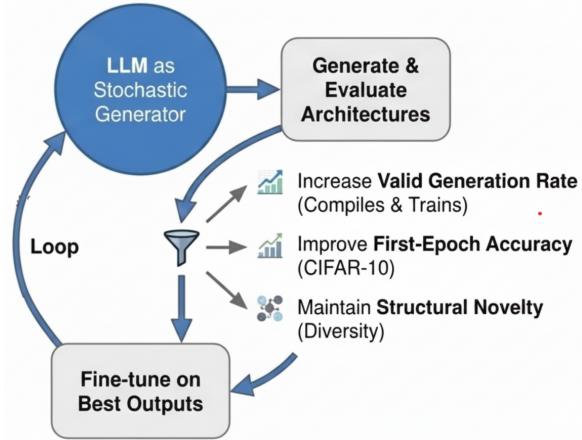


Figure 1: Overview of the iterative architecture-synthesis loop: the LLM generates architectures, candidates are evaluated and filtered (validity, first-epoch accuracy, novelty), and the LLM is fine-tuned on selected outputs.

shapes (e.g., $(N, 3, 32, 32) \rightarrow 10$ logits), and constraints on permissible operations (standard conv/pool/norm/activations; no pretrained weights or external feature extractors). A parameter budget of at most 500,000 parameters is imposed alongside an implicit edge-friendly latency target. Each generated model is required to implement a fixed API contract: a class `Net(nn.Module)` with `__init__`, `forward`, `train_setup`, `learn`, and a function `supported_hyperparameters()` returning `{"lr", "momentum"}`. Prompts require `import torch` and `import torch.nn as nn` and instruct the model to output a single `nn.Module` definition (no data loading or training loops). 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.

Drawing upon recent advances in LLM applications across diverse domains (Gado et al., 2025; Rupani et al., 2025; Khalid et al., 2025) and prior architectural synthesis studies within the NNGPT framework (Kochnev et al., 2025a; Jesani et al., 2025; Vysyaraju et al., 2025; Mittal et al., 2025; Shrestha et al., 2026), we initialize supervision using the LEMUR Neural Network Dataset as part of the NNGPT ecosystem (Kochnev et al., 2025b). This dataset constitutes the foundational knowledge base of NNGPT and encompasses a broad range of both high-capacity and edge-optimized neural network models (Goodarzi et al., 2025; Uzun et al., 2026; Din et al., 2025). Leveraging this diverse col-

lection of performance-annotated metadata enables the model to internalize empirical performance signals and to transition from stochastic code generation toward informed, task-aligned architectural design. A total of 109,913 model records are obtained and then pruned by removing text-level near-duplicates using MinHash/LSH over token shingles. Specifically, each code snippet is tokenized, k -shingles are formed over tokens, and MinHash signatures with locality-sensitive hashing (LSH) are used to retrieve near-neighbor candidates efficiently; For every successfully parsed and trained candidate, we compute two Jaccard similarities over token-level shingles of the model code. The first, denoted J_{train} , measures similarity to the deduplicated supervised training set (LEMUR-derived models plus all previously accepted generations); the second, denoted $J_{\text{gen}}^{(c)}$, measures similarity to the set of all code samples generated earlier in the same cycle c .

In the implementation, these quantities are logged as `nn_jaccard_train` and `nn_jaccard_gen`, respectively, and are accompanied by Boolean flags `near_dup_text_train` and `near_dup_text_gen` that indicate whether either similarity exceeds a predefined near-duplicate threshold. A typical log entry for a valid model therefore contains fields such as

```
"nn_jaccard_train": 0.0,
"near_dup_text_train": false,
"nn_jaccard_gen": 0.8047,
"near_dup_text_gen": false,
"rejection_count": 1
```

where `rejection_count` records how many previously sampled candidates were rejected for being too similar to existing code before accepting the current model.

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 of shingle sets for all models generated in cycle c prior to the current candidate. For a new candidate with shingle set A , we estimate

$$\begin{aligned} J_{\text{train}}(A) &= \max_{B \in S_{\text{train}}} J(A, B), \\ J_{\text{gen}}^{(c)}(A) &= \max_{B \in S_{\text{gen}}^{(c)}} J(A, B), \end{aligned} \quad (1)$$

via their MinHash signatures and LSH index. If either similarity exceeds a near-duplicate threshold τ (e.g., $\tau \approx 0.9$), the candidate is marked as a

text-level near-duplicate and 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.

This procedure yields 1,065 unique records. These records are converted to instruction-following chat examples via a ChatPrepConfig; 216 records are dropped due to conversion/format issues, leaving 849 converted records. For each converted record, two training examples are created with identical assistant code but different user descriptions (MNIST vs. CIFAR-10), yielding $849 \times 2 = 1,698$ chat-style prompt–code pairs for the initial training split.

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 CIFAR-10 accuracy threshold of 40%, and (iii) pass a near-duplicate filter against both the current training corpus and earlier generations from the same cycle using the same MinHash–Jaccard approach described above. For novelty filtering, each trained candidate with code-shingle set S is assigned an estimated maximum Jaccard similarity to the supervised corpus and to other candidates in the same cycle; near-duplicates above a threshold $\tau \approx 0.9$ are rejected. Each qualifying model is converted into a chat-format prompt–code pair and appended to the training set for the next cycle. Across 22 cycles, this process adds 455 unique high-accuracy models, expanding the training corpus from 1,698 to 2,153 examples by Cycle 22 (Table 1).

Source	Prompt–code pairs
LEMUR (deduplicated, chat-format train split)	1,698
Self-generated (22 cycles, $\geq 40\%$ acc., novel)	455
Total used for training by cycle 22	2,153

Table 1: Supervised training corpus by the end of cycle 22.

Each generated code snippet is executed in an isolated environment. Candidates are rejected if Python parsing fails, if Net cannot be instantiated (e.g., missing arguments or incompatible dimen-

sions), or if a dummy forward pass raises an exception (e.g., tensor shape mismatch). To ensure comparability across experimental cycles, all remaining candidates are subjected to a standardized training protocol on the CIFAR-10 dataset. Key hyperparameters—including the training/validation split, input resolution, optimization schedule, and batch size—are held constant. Advanced data augmentation techniques derived from [Aboudeshish et al. \(2025\)](#) are utilized to maintain a consistent baseline. Implementation details for MinHash/LSH near-duplicate detection and novelty filtering (including shingle size, signature length, and thresholds) are provided in Appendix A.

For each syntactically valid model m , the top-1 validation accuracy after a single epoch, denoted as $A(m)$, is recorded. This approach serves as a low-fidelity performance proxy, a common practice in NAS to balance computational efficiency with evaluative reliability ([Ru et al., 2020](#); [Zela et al., 2020b](#)). This standardized evaluation allows for the efficient ranking of model candidates while minimizing the computational overhead of full convergence training. Per-cycle summaries include the valid generation rate.

The valid generation rate is

$$p_{\text{valid}}^{(c)} = \frac{N_{\text{valid}}^{(c)}}{N_{\text{gen}}^{(c)}}. \quad (2)$$

Let $N_{\text{gen}}^{(c)}$ denote the number of candidate architectures sampled from the LLM in cycle c , and let $N_{\text{valid}}^{(c)}$ denote the subset that compile and train successfully.

For all valid models in cycle c , let

$$\mathcal{A}^{(c)} = \{A(m) : m \in \mathcal{M}_{\text{valid}}^{(c)}\}.$$

For the sample mean $\bar{A}^{(c)}$ and standard deviation $s^{(c)}$ computed over $n_c = |\mathcal{A}^{(c)}|$, we use the usual t -based 95% confidence interval:

$$\bar{A}^{(c)} \pm t_{0.975, n_c - 1} \frac{s^{(c)}}{\sqrt{n_c}}. \quad (3)$$

For any proportion $\hat{p} = k/n$ (e.g., $p_{\text{valid}}^{(c)}$ or the proportion with $A(m) \geq \tau$), we report Wilson score confidence intervals ([Wilson, 1927](#)).

3.1 Fine-tuning and Generation Hyperparameters

Fine-tuning is performed using DeepSeek-Coder-7B-Instruct-v1.5 ([Guo et al., 2024](#)) adapted with

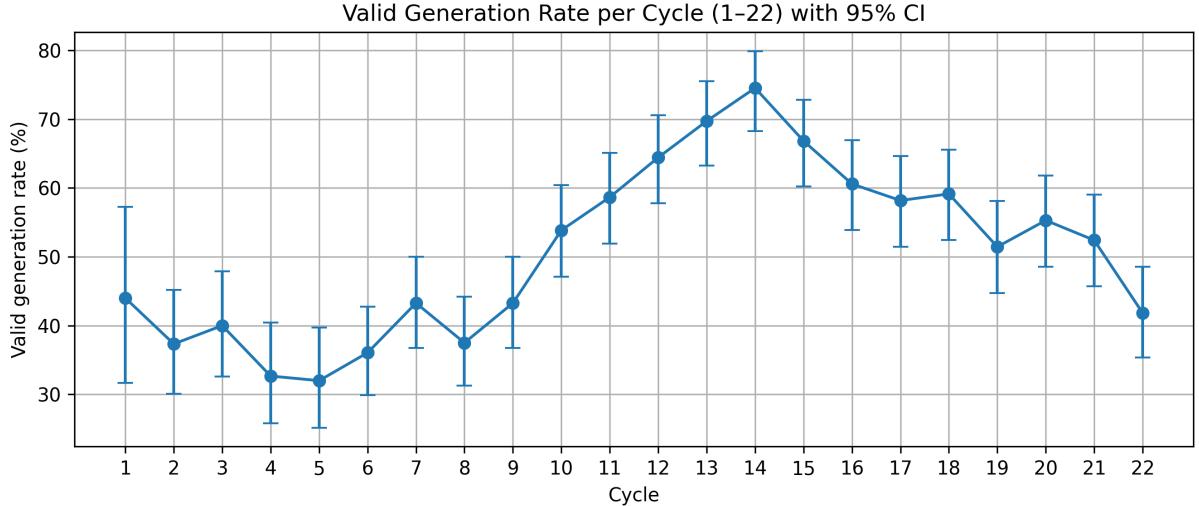


Figure 2: Valid generation rate per cycle (1–22) with Wilson 95% confidence intervals.

LoRA (Hu et al., 2022). LoRA hyperparameters are fixed across all cycles: rank $r=32$, LoRA $\alpha=32$, dropout 0.05; LoRA is applied to attention projections (q/k/v/o) and MLP projections (up_proj, down_proj, gate_proj) in all 24 Transformer layers (0–23). The task is causal language modeling over chat-format prompt–response pairs. In each cycle, fine-tuning runs for 5 epochs with learning rate 1×10^{-5} , per-device batch size 1, gradient accumulation 4 (effective batch size 4), paged AdamW in 8-bit, cosine schedule with 20 warmup steps, weight decay 0.01, max grad norm 1.0, and bfloat16 precision. The only changing factor across cycles is the size of the training data, which grows by adding accepted self-generated models from prior cycles.

Generation is performed in a chat interface. The system message casts the model as an expert PyTorch architecture designer optimizing for first-epoch accuracy under constraints; the user message specifies dataset, shapes, parameter budget, and the API contract. Decoding is held constant across cycles: temperature 0.20, top- k 50, nucleus p 0.9, do_sample=True, and max new tokens 2,048 (pad with EOS). Keeping prompts and decoding fixed isolates the effect of iterative fine-tuning and data growth on the generator.

4 Results

The 22-cycle synthesis loop is evaluated using the metrics defined in Section 3. In each cycle, the LLM produces candidate architectures; candidates are filtered for validity, each valid model is trained for one epoch on CIFAR-10, and high-accuracy,

structurally novel architectures are retained for the subsequent fine-tuning round. Representative checkpoints are reported in Table 2, while Figure 3 summarizes the joint evolution of reliability (valid generation), proxy performance (first-epoch accuracy), novelty-based selection, and training-set growth. Additional per-cycle plots are provided in Appendix B.

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	2154

Table 2: Selected cycle statistics: valid generation rate, best and mean first-epoch accuracy on CIFAR-10, proportion of models with accuracy $\geq 40\%$, number of structurally unique models selected, and cumulative training-set size.

We report reliability in terms of the valid generation rate. Cycle 1 begins at 44.0% validity (22/50). Following early fluctuations in the low-to-mid 30% range (cycles 2–5; e.g., 32.0% at cycle 5 in Table 2), validity increases and reaches a peak of 74.5% in cycle 14 (155/208). 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 (Appendix B.2); across all 22 cycles, the mean valid generation rate is 50.6% with a 95% confidence interval of [45.0%,

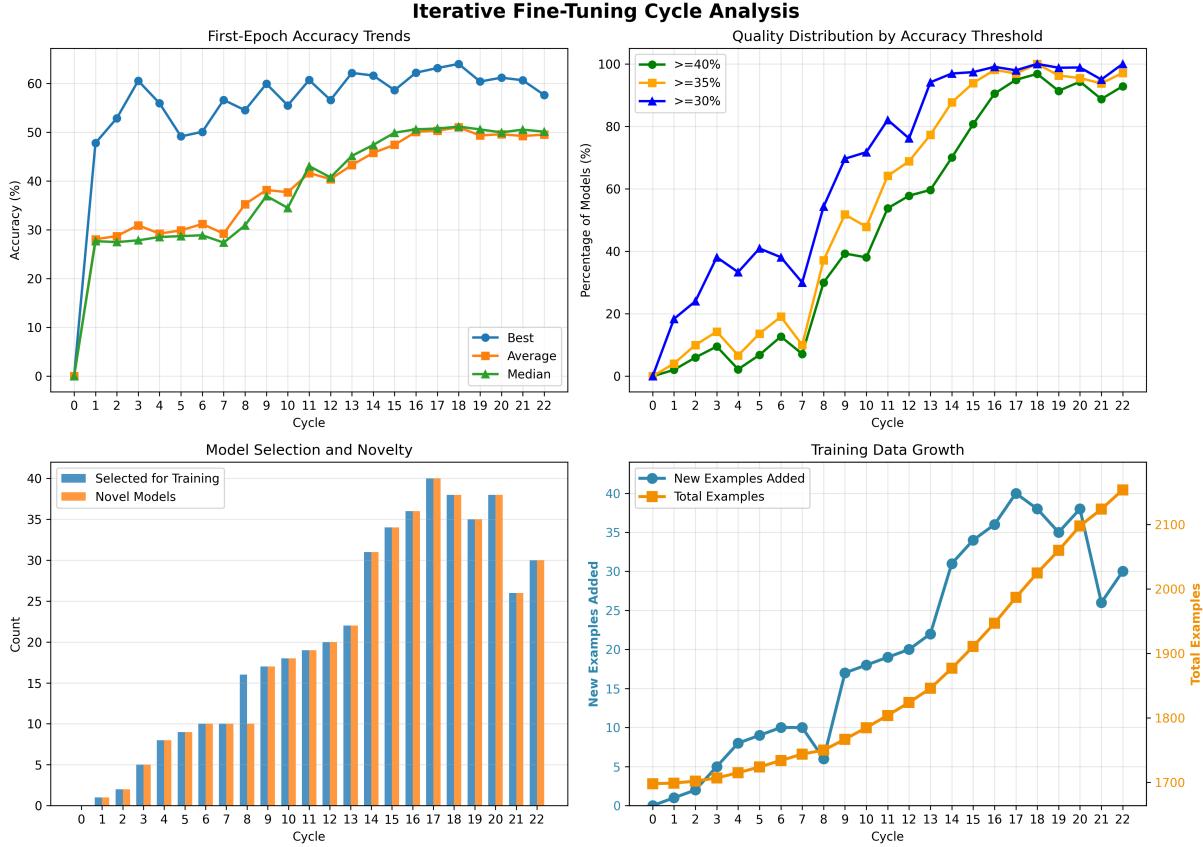


Figure 3: Overall analysis of the 22 fine-tuning cycles: (top-left) first-epoch accuracy trends; (top-right) quality distribution by accuracy threshold; (bottom-left) model selection and novelty; (bottom-right) training-data growth.

56.1%].

Proxy quality, measured by first-epoch CIFAR-10 validation accuracy, exhibits a marked upward shift over the course of training. In cycle 1, the best model reaches 47.78% and the mean accuracy is 28.06% (median 27.67%), indicating that most early-cycle valid generations learn weakly under the one-epoch protocol. By cycle 10, the best model improves to 55.48% and the mean rises to 37.70% (Table 2), reflecting a substantial redistribution of generated architectures toward faster early learning. 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 in central tendency (best 57.62%, mean 49.48%, median 50.10%), despite the lower validity reported above. When all 1,754 trained architectures are pooled, the overall mean first-epoch accuracy is 42.32% with a 95% confidence interval of [41.80%, 42.83%], showing that the typical sampled-and-trained model improves considerably relative to early cycles.

A complementary view of quality is provided by the mass above the fixed 40% accuracy threshold

used for selection into the next cycle. Only 2.04% of trained models exceed 40% in cycle 1, increasing to 6.82% by cycle 5 and reaching 38.04% by cycle 10 (Table 2). After this transition, the fraction above 40% 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 does not merely improve the best-of-sample outcome; instead, it 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 and fluctuate within a narrower band, suggesting diminishing returns and possible overfitting to self-generated data.

Novelty and corpus growth persist throughout training rather than collapsing into repeated templates. 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. Consistent with Table 2, the cumulative training set grows steadily,

from 1,699 prompt–code pairs at initialization to 2,154 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: the upward shift in the accuracy distribution, the increase in the fraction exceeding 40%, continued admission of unique models, and the monotonic increase in training data.

4.1 Ablation Study

The proposed synthesis loop combines three components: (i) a MinHash–Jaccard novelty filter that removes text-level near-duplicates, (ii) a first-epoch accuracy threshold of 40% for selecting high-quality models, and (iii) iterative LoRA fine-tuning of the LLM on accepted self-generated architectures. To isolate the contribution of each component to the behavior reported in Section 4, three ablation variants are considered, each removing one ingredient: generation without the novelty filter, generation without the performance-based accuracy threshold, and a non-iterative baseline without cycle-wise fine-tuning. Across all ablations, the prompt template, decoding configuration, and evaluation protocol are kept identical to the main setting.

First, the novelty filter is disabled while the 40% accuracy threshold and LoRA fine-tuning remain unchanged. Under this condition, any architecture that compiles, trains for one epoch, and exceeds the 40% first-epoch accuracy threshold becomes eligible for inclusion in the training corpus, even if its code closely matches models already present. Across cycles, the valid generation rate and first-epoch accuracy trends remain qualitatively similar to the full method, and the generator still transitions into a regime where a large fraction of valid models exceed the 40% threshold. However, the composition of the training corpus changes substantially: without the novelty constraint, the corpus accumulates repeated motifs and near-duplicate architectures, and the number of genuinely distinct code patterns added per cycle is noticeably lower than in the full method. Generated models therefore retain acceptable accuracy but exhibit reduced code-level diversity, indicating that the novelty filter is central for sustaining exploration of new regions of the design space rather than repeatedly reinforcing a narrow family of designs.

Second, the MinHash-based novelty filter is retained, but the 40% first-epoch accuracy threshold is removed. In this variant, all architectures that

(i) compile and train for one epoch and (ii) are non-duplicates with respect to the accepted set and earlier generations are added to the training corpus regardless of early-epoch performance. Improvements in valid generation rate are still observed as the LLM is exposed to more executable code, and the generator continues to produce architectures with reasonable first-epoch accuracy. Nonetheless, the shift in the overall accuracy distribution is clearly weaker than in the full method. The training set contains a broader mixture of low- and high-performing architectures, and low-accuracy yet valid models are regularly promoted into the corpus. Consequently, the fraction of models exceeding the 40% accuracy threshold increases more slowly and stabilizes at a lower level than when performance filtering is applied. This outcome indicates that novelty alone does not suffice to steer refinement toward rapidly learning architectures under the early-epoch proxy; performance-based selection is required to align corpus growth with empirical learning behavior.

Third, the iterative refinement mechanism is removed entirely. The base LLM is fine-tuned once on the initial LEMUR-derived training split, and architectures are generated from this fixed model without adding self-generated examples back into the training corpus. The evaluation protocol (validity checking, single-epoch training, and accuracy measurement) is unchanged, but the feedback loop is disabled. Under this non-iterative baseline, the valid generation rate and first-epoch accuracies match the behavior observed in early cycles of the full loop and do not exhibit the progressive improvements obtained under cycle-wise updates. The proportion of models surpassing the 40% accuracy threshold remains substantially below the levels achieved in later cycles of the iterative run, and the distribution of architectures does not shift into the regime where high-accuracy models dominate. Although code-level novel architectures can still be produced, these discoveries do not influence future generations, leaving the generator effectively static.

Taken together, the ablations decompose the synthesis loop into three interacting mechanisms. The MinHash-based novelty filter prevents collapse of the training corpus onto near-duplicate codes and sustains structural exploration; the 40% accuracy threshold ties inclusion of new examples to empirical learning behavior and strengthens the shift toward rapidly learning architectures; and itera-

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 3: Ablation study of the synthesis loop. *Valid rate* is the proportion of generated architectures that compile and train for one epoch. *Mean acc.* denotes the mean first-epoch CIFAR-10 accuracy over all valid models, with 95% confidence intervals reported in brackets (Wilson score for proportions, t -based for means). $\geq 40\%$ *acc.* is the proportion of valid models whose first-epoch accuracy is at least 40%. *Novel models* counts the number of self-generated architectures that pass the MinHash–Jaccard novelty filter and are added to the training corpus over the entire run. The full method combines MinHash-based novelty filtering, a 40% accuracy threshold, and iterative fine-tuning; the three ablations each remove one of these components.

tive fine-tuning closes the feedback loop, enabling the generator to internalize accumulated successes. Removing any single element degrades diversity, the performance distribution, or the long-horizon improvement trajectory, highlighting the need for their combination in the full method.

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 behavior is tracked across 22 supervised fine-tuning cycles using its own high-quality, structurally novel generations. Under a controlled CIFAR-10 image-classification setting, the generate–evaluate–select–fine-tune procedure induces a pronounced shift in the model’s output distribution over architectures, improving both the likelihood of producing executable models and the early-epoch performance of sampled networks, while retaining non-trivial structural diversity.

Across cycles, the generator moves from an initial regime in which valid, rapidly learning architectures are relatively uncommon to one in which they occur much more frequently. Although validity is not monotonic, it stabilizes for much of the run in a regime where a substantial fraction of sampled models compile and train successfully, which can reduce the overhead associated with repairing LLM-generated code. In parallel, the distribution of first-epoch accuracies shifts upward: the fraction of models exceeding a moderate performance threshold after a single epoch rises from only a small minority early on to a large majority in later cycles. Importantly, this improvement is achieved even though performance information affects learn-

ing only through the data-selection step; the fine-tuning objective remains standard next-token prediction.

Second outcome is the sustained admission of code-level novel architectures. By enforcing a MinHash–Jaccard-based novelty criterion and discarding near-duplicate implementations, the procedure constructs an expanding archive of architectures that are both above-threshold under the early-epoch proxy and diverse at the source-code level. Overall, the results suggest that code-capable LLMs can serve as increasingly useful architectural priors when trained within a lightweight, self-referential feedback loop that combines low-fidelity evaluation with novelty filtering. While the present study is intentionally constrained (single dataset, single base model, simple selection policy), it provides evidence that the generator’s behavior can be shaped in a systematic and measurable manner under controlled conditions.

Future work entails. First, integrating the refined generator with explicit optimization frameworks (e.g., LL Matic, SEKI, or RZ-NAS) could leverage the learned distribution as an architectural prior for downstream search (Nasir et al., 2023; Cai et al., 2025; Ji et al., 2025). Second, generalizing beyond CIFAR-10 via interleaved multi-task prompting could promote transferable architectural regularities over task-specific heuristics, utilizing repositories such as LEMUR as a foundation (Goodarzi et al., 2025). Third, incorporating granular feedback signals—replacing binary thresholds with performance-weighted sampling or reinforcement learning—could better align the generator. Furthermore, integrating multi-objective constraints (e.g., parameter count, latency) would allow the model to internalize the accuracy–efficiency trade-offs critical to edge-oriented NAS (Rahman et al., 2025; Barradas-Palmeros et al., 2025).

6 Limitations

Despite the observed improvements, several limitations and threats to validity remain. First, the study is restricted to a single dataset and task, namely CIFAR-10 image classification with a fixed input resolution. This choice enables controlled analysis, but it remains unclear how well the observed trends transfer to substantially different datasets (e.g., higher-resolution images or non-visual domains), input modalities, or task formulations such as segmentation or detection. The induced architectural prior may therefore be partly specialized to the CIFAR-10 setting.

Second, first-epoch validation accuracy is used as the sole performance signal. While early-epoch metrics and learning-curve extrapolation can correlate with eventual performance (Domhan et al., 2015; Ru et al., 2020), the relationship is imperfect and may favor architectures that learn quickly early but plateau later. In addition, longer-horizon training of the best discovered architectures is not evaluated here, and direct comparisons to human-designed baselines or classical NAS methods under matched computational budgets are not included. As a result, the conclusions primarily concern the *relative* evolution of the generator under a fixed proxy evaluation protocol rather than absolute state-of-the-art performance.

Third, the refinement strategy and selection policy are intentionally simple. LoRA adaptation is performed with fixed hyperparameters across cycles, and the acceptance threshold (40% first-epoch accuracy) is held constant after selection. Although this design isolates the effect of dataset evolution, it is not necessarily optimal; the observed plateauing behavior in later cycles suggests that alternative curricula or additional regularization could be beneficial.

Finally, novelty is defined using MinHash-Jaccard similarity over token-level shingles of source code and therefore operates at the text level rather than on explicit computation graphs. Different implementations can encode functionally similar architectures, while functionally equivalent models may be treated as novel if expressed in sufficiently different styles. Although the text-level filter removes obvious near-duplicates and trivial edits, it does not capture deeper semantic equivalence between architectures.

7 Ethical Considerations

This study employs a code-oriented large language model (LLM), specifically DeepSeek-Coder-7B-Instruct-v1.5, within a closed, controlled, and iterative architecture-synthesis framework. The research is strictly methodological in nature and focuses on neural architecture search. It does not involve human participants, personal data, or the deployment of an end-user-facing system.

Data Usage and Privacy. The initial training corpus is derived from the publicly available LEMUR Neural Network Dataset. The dataset consists exclusively of source code and associated technical metadata and does not contain personally identifiable information (PII) or sensitive user data. Architectures generated during the iterative synthesis process are programmatically produced code artifacts within an isolated execution environment. As a result, no private, confidential, or user-generated data are introduced into the training or evaluation loop.

Security and Integrity of Generated Code. The LLM is used to generate executable source code as part of the architecture synthesis process. While syntactic and semantic validity checks are applied during experimentation, we note that any real-world deployment or reuse of LLM-generated code would necessitate comprehensive security reviews. Such audits would be required to mitigate risks related to unsafe coding practices or the inadvertent reproduction of vulnerable code patterns.

Transparency and Reproducibility. To support transparency and reproducibility, we provide a detailed account of the experimental setup, including the base LLM, fine-tuning strategy (LoRA), data filtering mechanisms (MinHash-Jaccard novelty filtering), and evaluation protocols. This level of documentation is intended to facilitate independent verification of the results and to enable replication or extension of the proposed methodology by the research community.

References

- Nada Aboudeshish, Dmitry Ignatov, and Radu Timofte. 2025. *Augmentgest: Can random data cropping augmentation boost gesture recognition performance?* *arXiv preprint*, arXiv:2506.07216.
Massih-Reza Amini, Vasilii Feofanov, Loïc Pauletto,

- Emilie Devijver, and Yury Maximov. 2022. **Self-training: A survey**. *CoRR*, abs/2202.12040.
- Jesús-Arnulfo Barradas-Palmeros, Carlos-Alberto López-Herrera, Erick Mezura-Montes, Héctor-Gabriel Acosta-Mesa, and Ana-Leticia López-Lobato. 2025. **Testing neural architecture search efficient evaluation methods in deepgta**. *Mathematical and Computational Applications*, 30(4):74.
- Zicheng Cai, Yaohua Tang, Yutao Lai, Hua Wang, Zhi Chen, and Hao Chen. 2025. **SEKI: Self-evolution and knowledge inspiration based neural architecture search via large language models**. *arXiv preprint*, arXiv:2502.20422.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, and 1 others. 2021. **Evaluating large language models trained on code**. *arXiv preprint*.
- Saif U Din, Muhammad Ahsan Hussain, Mohsin Ikram, Dmitry Ignatov, and Radu Timofte. 2025. **Ai on the edge: An automated pipeline for pytorch-to-android deployment and benchmarking**. *Preprints*.
- Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. 2015. 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.
- Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.
- Mohamed Gado, Towhid Taliee, Muhammad Danish Memon, Dmitry Ignatov, and Radu Timofte. 2025. **Vist-gpt: Ushering in the era of visual storytelling with llms?** *arXiv preprint*, arXiv:2504.19267.
- 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. 2025. **Lemur neural network dataset: Towards seamless automl**. *arXiv preprint*, arXiv:2504.10552.
- Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, and 1 others. 2024. **Deepseek-coder: When the large language model meets programming – the rise of code intelligence**. *arXiv preprint*, arXiv:2401.14196.
- Edward J. Hu, Yelong Shen, Phillip 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)*.
- Krunal Jesani, Dmitry Ignatov, and Radu Timofte. 2025. **Llm as a neural architect: Controlled generation of image captioning models under strict api contracts**. *arXiv preprint*, arXiv:2512.14706.
- 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 42nd International Conference on Machine Learning (ICML)*. To appear.
- Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. **A survey on large language models for code generation**. *arXiv preprint*.
- Jeon-Seong Kang, JinKyu Kang, Jung-Jun Kim, Kwang-Woo Jeon, Hyun-Joon Chung, and Byung-Hoon Park. 2023. **Neural architecture search survey: A computer vision perspective**. *Sensors*, 23(3):1713.
- Waleed Khalid, Dmitry Ignatov, and Radu Timofte. 2025. **A retrieval-augmented generation approach to extracting algorithmic logic from neural networks**. *arXiv preprint*, arXiv:2512.04329.
- Roman Kochnev, Arash Torabi Goodarzi, Zofia Antonina Bentyn, Dmitry Ignatov, and Radu Timofte. 2025a. **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.
- 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. 2025b. **Nngpt: Rethinking automl with large language models**. *arXiv preprint*, arXiv:2511.2033.
- Alex Krizhevsky. 2009. **Learning multiple layers of features from tiny images**. Technical Report.
- Yash Mittal, Dmitry Ignatov, and Radu Timofte. 2025. **Preparation of fractal-inspired computational architectures for advanced large language model analysis**. *arXiv preprint*, arXiv:2511.07329.
- Muhammad U. Nasir, Sam Earle, Christopher Cleghorn, Steven James, and Julian Togelius. 2023. **LL-Matic: Neural architecture search via large language models and quality-diversity optimization**. *CoRR*, abs/2306.01102. Also in GECCO 2024.
- Md Hafizur Rahman, Zafaryab Haider, and Prabuddha Chakraborty. 2025. **An automated multi parameter neural architecture discovery framework using ChatGPT in the backend**. *Scientific Reports*, 15(16871).
- Binxin Ru, Rui Shu, Xinyi Dong, and 1 others. 2020. **Speedy performance estimation for neural architecture search**. In *Proceedings of the International Conference on Machine Learning (ICML) Workshop on AutoML*.
- Bhavya Rupani, Dmitry Ignatov, and Radu Timofte. 2025. **Exploring the collaboration between vision models and llms for enhanced image classification**. *Preprints*.

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

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicholas Papernot, Robert D. Anderson, Yoav Ganin, and Ross J. Anderson. 2024. [AI models collapse when trained on recursively generated data](#). *Nature*, 631(8022):755–759.

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. 2026. Lemur 2: Unlocking neural network diversity for ai. *arXiv preprint*.

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

Colin White, Mahmoud Safari, Rhea Sukthanker, Binxi Ru, Thomas Elsken, Arber Zela, Debadeepa Dey, and Frank Hutter. 2023. [Neural architecture search: Insights from 1000 papers](#). *arXiv preprint arXiv:2301.08727*.

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

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

Arber Zela, Julien Siems, and Frank Hutter. 2020b. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations (ICLR)*.

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

A Additional Method Details

A.1 MinHash/LSH configuration for near-duplicate detection

Tokenization and shingling. We perform lexer-based tokenization of Python/PyTorch source code into a sequence of syntactic tokens (e.g., keywords, identifiers, literals, operators, and delimiters). We then construct token-level shingles using a shingle size of $k = 10$ (i.e., contiguous 10-grams over the token stream), yielding a set representation for each architecture.

MinHash signatures. Each shingle set is mapped to a MinHash signature using $N_{\text{perm}} = 256$ permutations (hash functions). MinHash signatures are used to efficiently approximate Jaccard similarity between shingle sets.

LSH candidate retrieval. To accelerate retrieval, we index MinHash signatures using locality-sensitive hashing (LSH) with a retrieval threshold of 0.85, producing a candidate set of potentially similar architectures via band collisions. Candidates are subsequently verified using the MinHash-estimated Jaccard similarity.

Acceptance threshold for near-duplicates. A pair of architectures is marked as a near-duplicate if the estimated Jaccard similarity exceeds $\tau = 0.90$. We use the same τ consistently for lexical/structural duplicate checks, including dataset curation and novelty filtering during sampling.

A.2 Operational novelty filtering during sampling

Order of evaluation. Novelty filtering is applied only after a candidate satisfies execution validity (successful parse, instantiation, and forward pass) and completes one epoch of training. This ordering avoids expending LSH queries on invalid candidates.

Cycle-local archive. Within each sampling cycle, we maintain an in-memory archive of MinHash signatures for all previously accepted candidates in that cycle to enable efficient computation of $J_{\text{gen}}^{(c)}$.

Rejection accounting. We record the number of rejected candidates encountered before accepting a non-duplicate architecture (`rejection_count`), quantifying the propensity of the generator to propose near-duplicates under fixed decoding settings.

B Additional Results

B.1 First-epoch accuracy trends

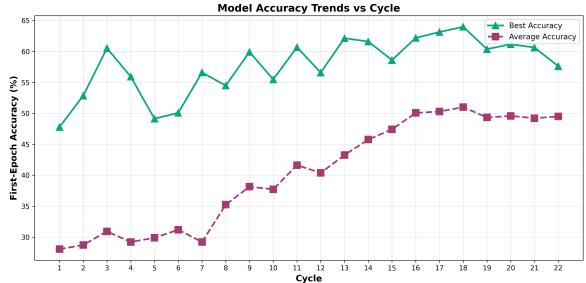


Figure 4: Best, mean, and median first-epoch accuracy per cycle on CIFAR-10 (see Section 4).

B.2 Proportion exceeding the 40% first-epoch accuracy threshold

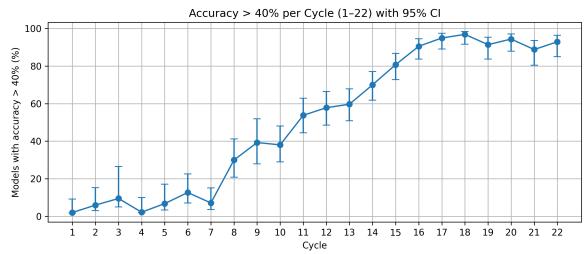


Figure 5: Proportion of models with first-epoch accuracy $\geq 40\%$ per cycle (see Section 4).

Late-cycle saturation. After cycle 18, both best and mean first-epoch accuracies plateau and fluctuate within a narrow band, and the fraction of models above 40% stabilizes. This saturation suggests diminishing returns from continued self-training and is consistent with known failure modes of iterative self-training pipelines, where selection bias and distributional narrowing can limit further gains when the training set becomes increasingly dominated by model-generated samples (Amini et al., 2022; Shumailov et al., 2024).