Scaling Closed-Loop Feature Channel Configuration with LLMs

Tolgay Atinc Uzun(✉), Radu Timofte, and Dmitry Ignatov

Computer Vision Lab, CAIDAS & IFI, University of Würzburg, Germany
t.atincuzun@gmail.com

Abstract. Promising initial results in closed-loop large-language-model-based channel-configuration search demonstrated that neural-network widths can be optimized directly through executable code generation and accuracy feedback. However, those results were obtained from a relatively sparse set of valid evaluations, leaving open whether the observed optimization behavior transfers to a denser sampling regime and whether additional architectural regularities emerge when more generated networks are evaluated. To test this, the same search setting is scaled to 250 candidate networks per fine-tuning cycle. The analysis covers 2000 generated candidates from 8 complete cycles, yielding 462 verified CIFAR-100 evaluations after task and metadata filtering. The expanded sample strengthens the evidence for a performance-guided channel-search signal. Per-cycle mean accuracy exhibits a positive linear trend with slope 9.87×10^{-4} ($p = 0.043$), while the high-performing frontier improves more strongly: the best observed accuracy increases from 0.3144 to 0.3676, and both the top-5 and top-10 cycle-level means exhibit positive trends. The scaled run also reveals improved parameter efficiency. The best model reaches 0.3676 with 11.8M parameters, compared with an early high-performing model at 0.3144 with 166.5M parameters. Beyond accuracy, the larger sample exposes architectural regularities that were difficult to assess from sparse observations. Non-power-of-two channel widths occur in 41.8% of verified candidates, and the strongest models share structured channel-allocation patterns characterized by moderate early widths and expanded middle or later blocks. These findings indicate that the channel-search signal observed in the initial study transfers to a larger-data setting and that increased sampling scale turns isolated candidate discovery into a measurable distributional and architectural analysis of LLM-generated channel priors.

Keywords: Large Language Models · Neural Architecture Search · Channel Configuration · CIFAR-100 · Proxy Evaluation

1 Introduction

Neural architecture search (NAS) is usually formulated as optimization over a constrained graph or supernet [11, 24]. Channel-number search is a narrower

but practically important case: it changes per-layer widths while preserving the executable network skeleton, and it is closely related to efficient-model design and channel-pruning literature [7,12,13,19]. The preceding study asked whether large language models (LLMs) can perform such search directly over neural-network source code [16]. In that framework, the LLM generates a PyTorch model, a training harness evaluates it, and successful generations are fed back into later fine-tuning cycles.

The preceding result established feasibility on a smaller evaluation sample. Many generated programs may fail, some successful executions may correspond to non-target settings, and the best candidate is an order statistic over a sampled architecture population. Reporting only the final best architecture therefore leaves open the question of whether the search behavior carries over to a larger generation budget.

The present study analyzes a denser run of the same channel-configuration loop. Each complete cycle contains 250 generated candidates, compared with a substantially smaller sample in the preceding experiment. The resulting data support three observations: the per-cycle mean accuracy has a positive trend, the high-performing frontier improves under scaling, and later cycles identify substantially more parameter-efficient channel configurations than the early incumbent. The contribution of the present study is therefore a scaled empirical analysis of LLM-driven channel search and the architectural patterns that become visible with more generated candidates.

Contributions. First, the study provides a strict candidate-accounting protocol for LLM-generated architecture experiments, distinguishing failed code, missing logs, non-target datasets, and target-valid evaluations. Second, the scaled run recovers and strengthens the accuracy and frontier signals discussed in the preceding study. Third, channel patterns are analyzed at scale using both programmatic extraction and dynamic PyTorch instrumentation of generated networks. Fourth, a self-contained analysis script rebuilds all tables and figures from the raw folders without relying on pre-existing processed data frames.

2 Related Work

Neural architecture search and channel configuration. Classical NAS formulates architecture design as an optimization problem over a discrete or continuous search space. Reinforcement-learning-based NAS demonstrated that architectures can be learned from validation feedback [24], while differentiable methods such as DARTS reduced the cost of search by relaxing architectural decisions into continuous parameters [11]. For efficient convolutional networks, a closely related line of work searches or prunes channel dimensions rather than entire macro-architectures. Network Slimming uses sparsity-induced channel selection [13]; AMC frames compression as automated policy search [7]; MetaPruning predicts weights for pruned channel configurations [12]; and AutoSlim searches channel widths in a slimmable network [19]. These methods usually operate within predefined search spaces or supernets. The present study instead ana-

lyzes channel configurations generated as executable code by a language model, while keeping the scientific focus on channel allocation, parameter efficiency, and proxy accuracy under a scaled candidate budget.

Language models for architecture generation. Recent work has explored LLMs as optimizers, code editors, or knowledge sources for architecture design. GPT-style models have been tested for NAS prompting and iterative architecture proposal [18, 21]. EvoPrompting and LLMatic use language-model code generation in evolutionary or quality-diversity search loops [4, 14]. Other systems use LLMs to transfer design principles [23], perform self-evolution and knowledge-inspired search [2], or coordinate multiple LLM roles for knowledge-guided search [10]. A complementary line of work studies the infrastructure needed for generated neural networks to be stored, evaluated, and reused. NNGPT rethinks AutoML around LLM-generated neural-network code under executable API constraints [9], while LEMUR and LEMUR2 provide large neural-network datasets and diversity-oriented resources for automated neural design exploration [6, 17]. The closest prior work to the present study is the closed-loop channel-prior paper, which introduced feedback-driven LLM channel-configuration search and reported promising non-standard channel priors [16]. Compared with these broader architecture-generation and dataset efforts, the present analysis isolates a narrower question: whether the channel-search behavior observed in that prior study persists when the number of generated networks per cycle is increased, and what architectural regularities become measurable from the larger generated population.

Proxy evaluation and short-horizon signals. NAS systems often rely on reduced-cost evaluations before committing to expensive full training. Proxy signals may come from short training schedules [22], weight sharing through supernetworks [11], zero-cost indicators computed from a single minibatch [1], or reduced search spaces. Systematic comparisons show that proxies vary widely in how well they preserve the true ranking of architectures [20, 22]. Such approximations are useful for ranking many candidates but must be interpreted as search-time evidence rather than final trained performance. The present work follows this logic: one-epoch CIFAR-100 accuracy is treated as a fixed proxy for early learning behavior. The dynamic PyTorch analysis therefore asks which generated channel configurations learn quickly under this proxy, which stage-wise allocations are associated with short-horizon accuracy, and which models improve the accuracy-parameter frontier before full-training confirmation.

3 Closed-Loop Pipeline and Scaled Evaluation Setting

The pipeline treats channel search as conditional code generation [16]. The input prompt contains a baseline network, task metadata, hyperparameters, and a target improvement. The LLM then emits a complete neural-network program and training-related blocks. The candidate is evaluated under a fixed one-epoch proxy protocol on CIFAR-100. Valid, evaluated candidates are stored and later used to update the model through parameter-efficient fine-tuning with LoRA [5, 8].

The experiment uses the OlympicCoder-7B code model [15] and an AirNet-like residual convolutional skeleton [3]. Generated networks and metadata are stored in the LEMUR/NNGPT ecosystem [6, 9, 17]. LoRA hyperparameters follow the standard adapter settings: rank 32, $\alpha = 32$, dropout 0.05, applied to the q, k, v projection modules of the base model; fine-tuning restarts from a fresh copy of the base model at the start of each cycle rather than continuing from the previous adapter.

The study scales the same closed-loop pipeline to 250 candidates per cycle, a substantial increase over prior sampling density. The raw experiment directory contains cycles `A0` through `A8`. Cycles `A0`–`A7` are complete, each with 250 candidate folders. Cycle `A8` is partial (seven CIFAR-100 evaluations) and is excluded from the main analysis.

Compute footprint. Across the 462 strict CIFAR-100 evaluations, the per-candidate one-epoch training took a median of 35 s (IQR 26–48 s) and a total of approximately 7.4 GPU-hours, with a median peak GPU memory of 21.5 GB. The 12.8%–34.4% strict success rate means roughly two thirds of the per-cycle wall-clock was spent on candidates that did not enter the analysis. This cost is part of why 8 cycles of 250 candidates each were needed to accumulate the 462 valid CIFAR-100 evaluations, and it frames the search as a sample-efficiency problem as well as an accuracy problem.

4 Data Accounting Protocol

A target-valid success is defined as a candidate in `A0`–`A7` with a valid `{epoch_number}.json` file (under the one-epoch protocol, `1.json`) and an `eval_info.json` confirming evaluation on `cifar-100`. The dataset check removes false-positive successful evaluations on other tasks. This rule yields 462 strict CIFAR-100 evaluations out of 2000 generated candidates.

Table 1 and Fig. 1 summarize the complete accounting. Across complete cycles, 1348 candidates terminate with an error trace, 185 have no logged output, and 5 are successful evaluations on a dataset other than CIFAR-100. The non-target evaluations are not failures of code execution, but they are excluded from the CIFAR-100 analysis because they do not answer the target channel-search question.

Overall, 23.1% of generated candidates become target-valid CIFAR-100 evaluations under this filtering rule. The unsuccessful candidates are used for error analysis, while the successfully trained CIFAR-100 models are used for the subsequent accuracy, efficiency, channel-configuration, and dynamic PyTorch analyses.

5 Accuracy Trends Under Scaling

Table 2 summarizes the strict CIFAR-100 results. The means lie in a narrow band from 0.2107 to 0.2203. A linear fit to the per-cycle means gives a positive slope of 9.87×10^{-4} accuracy points per cycle ($p = 0.043$), indicating an upward

Table 1. Per-cycle candidate accounting for the complete cycles. Each cycle contains 250 generated candidates. Only the strict CIFAR-100 column is used for accuracy and architecture claims.

Cycle	CIFAR-100	Other	Code/err	No output	Rate (%)
A0	69	4	153	24	27.6
A1	32	1	189	28	12.8
A2	47	0	179	24	18.8
A3	42	0	183	25	16.8
A4	71	0	166	13	28.4
A5	61	0	164	25	24.4
A6	54	0	172	24	21.6
A7	86	0	142	22	34.4

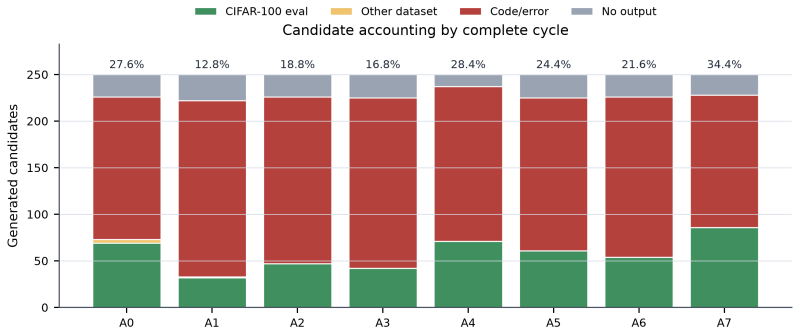

Fig. 1. Candidate accounting over the 8 complete cycles. The figure separates target-valid CIFAR-100 evaluations from non-target evaluations, code errors, and missing outputs.

trend at the cycle-mean level across eight cycle-level observations. Individual candidates are highly variable, so the stronger empirical signal is found in the upper tail of the distribution rather than in a per-candidate regression.

The primary scaling signal is in the frontier. The best strict CIFAR-100 model in `A0` reaches 0.3144. Later cycles improve the high-water mark to 0.3676 at A6/B190. Fig. 2 shows this distinction: the central mass of the distribution remains broad and noisy, while the upper tail improves more clearly. Linear fits to the top-5 and top-10 per-cycle means give slopes of 4.81×10^{-3} ($p = 0.017$) and 4.02×10^{-3} ($p = 0.016$), respectively. This supports the interpretation that scaling the candidate budget primarily reveals improved high-performing candidates rather than a uniform shift of the entire generated population.

These results show that the search improves the high-performing frontier over successive cycles. The best model is an order statistic that depends on sampling budget, and the scaled experiment makes that dependence visible: later samples include rare high-performing configurations that were less accessible from a smaller sample.

Table 2. Accuracy statistics for strict CIFAR-100 candidates. Top-5 mean is computed within each cycle and highlights the high-performing tail.

Cycle	N	Mean	Median	Q90	Max	Top-5 mean
A0	69	0.2122	0.2148	0.2616	0.3144	0.2852
A1	32	0.2134	0.2175	0.2676	0.2953	0.2788
A2	47	0.2107	0.2109	0.2568	0.2992	0.2747
A3	42	0.2166	0.2115	0.2745	0.3061	0.2819
A4	71	0.2180	0.2246	0.2578	0.3276	0.2971
A5	61	0.2187	0.2165	0.2646	0.3495	0.3103
A6	54	0.2203	0.2193	0.2708	0.3676	0.3133
A7	86	0.2156	0.2138	0.2652	0.3068	0.3008

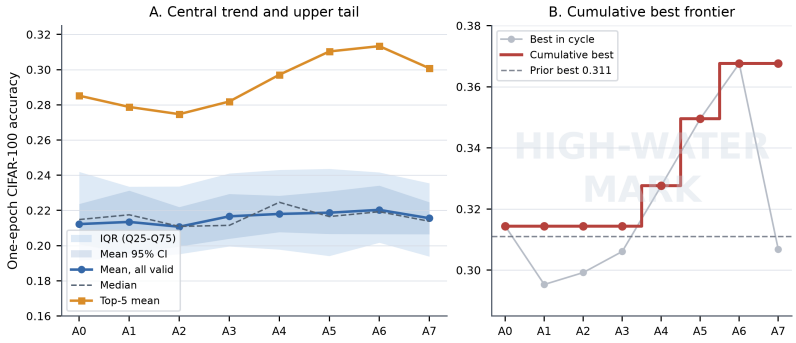

Fig. 2. Accuracy trends and frontier dynamics. Left: per-cycle mean accuracy for all strict CIFAR-100 candidates with a 95% confidence band, interquartile shading, median line, and top-5 mean. Right: the per-cycle best and cumulative high-water mark; the dashed reference is the best accuracy reported in the preceding study [16].

The trend p-values are computed on eight cycle-level points; bootstrap 95% confidence intervals on the per-cycle means and maxima are wide and overlap across cycles. The most stable statistical evidence is the positive linear trend of the top-5 and top-10 per-cycle means ($p < 0.02$), rather than the per-cycle mean alone.

6 Efficiency of the Discovered Frontier

The scaled run also shows a parameter-efficiency effect. A0/B134 reaches the highest A0 accuracy (0.3144) but is the largest A0 model at 166.5M parameters — the early cycle's best raw score came at a high parameter cost (the A0 median is 6.5 M, the 75th percentile 19 M). Late cycles recover the same or higher accuracy at much smaller scales: A6/B102 reaches 0.3400 with 5.0M parameters, and the global best A6/B190 reaches 0.3676 with 11.8M parameters. The shift is also visible at the population level: 26 of 69 A0 strict candidates are Pareto-

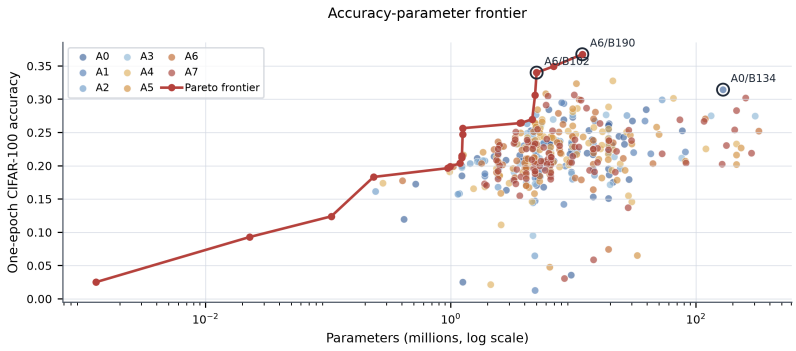

Fig. 3. Accuracy versus parameter count. Points are strict CIFAR-100 candidates, colored by generation cycle. The red curve is the empirical Pareto frontier. Late cycles include candidates that improve over the early high-parameter incumbent while using far fewer parameters.

dominated by A6/B190 (lower accuracy at higher parameter counts), and the per-cycle maximum at matched parameter budgets is consistently higher in cycles A4–A7 than in A0–A3. Thus the later frontier does not merely raise raw accuracy; it identifies channel allocations that dominate the early frontier across the full parameter range.

Fig. 3 shows the empirical Pareto frontier. This analysis supports comparison beyond the single best accuracy value. A scaled search should be evaluated not only by whether it finds a numerically similar maximum, but also by whether it recovers comparable trade-offs between accuracy and model size.

7 Channel Configuration Patterns

The analysis script extracts convolutional output widths by executing each generated `new_nn.py` under a lightweight `torch.nn`-compatible stub module, recording constructor arguments for `Conv2d`, `BatchNorm2d`, and `Linear` without GPU dependencies. Extraction succeeds for all 462 strict CIFAR-100 candidates, yielding 4961 convolutional width tokens with 94 distinct values.

7.1 Non-standard Widths

Non-power-of-two widths appear in 193 of 462 strict architectures, or 41.8% at the architecture level; at the token level, 20.2% of convolutional widths are non-power-of-two. Power-of-two values remain dominant — the most frequent widths are conventional values such as 256, 512, 64, 128, and 192. Architectures with at least one non-power-of-two width have mean accuracy 0.2144, compared with 0.2169 for all-power-of-two architectures, a negligible mean difference; the distinction is visible at the frontier, where the best all-power-of-two architecture

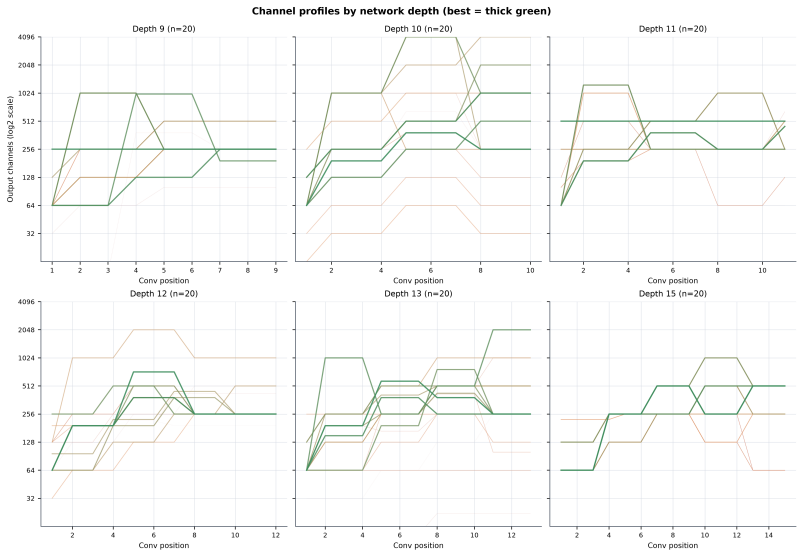

Fig. 4. Channel profiles faceted by network depth. Within each depth group, lines are colored by accuracy (thin red = low, thick green = high). The pattern of wider middle blocks for better models holds across depth values.

reaches 0.3276 and the best architecture with at least one non-power-of-two width reaches 0.3676.

7.2 Width-Position Patterns

Table 3 lists the top-performing candidates. The best model (A6/B190, 0.3676) has channel sequence [64, 192, 192, 192, 724, 724, 724, 256, 256, 256, 256, 256] — a 64-channel stem, three 192-channel layers, three 724-channel layers, and five 256-channel layers in run-length form. The pattern is blockwise: moderate early capacity, a wider middle section, and a controlled later representation. Fig. 4 shows that this pattern — better trajectories tend toward wider middle blocks — holds across network depths, and Fig. 5 shows the corresponding layer-local statistics. Non-power-of-two widths concentrate in early-middle positions, especially layers 2–7, and the association between width and accuracy is weaker near the stem and stronger in later positions. The per-layer tests are not independent, but the result shows that the scaled run produces structured, non-random width variation.

7.3 Dynamic PyTorch Evidence for Capacity Placement

The generated code permits a direct dynamic analysis with real PyTorch. Each strict CIFAR-100 network is instantiated, evaluated on a dummy CIFAR-shaped tensor, and instrumented with hooks on convolutional, linear, normalization,

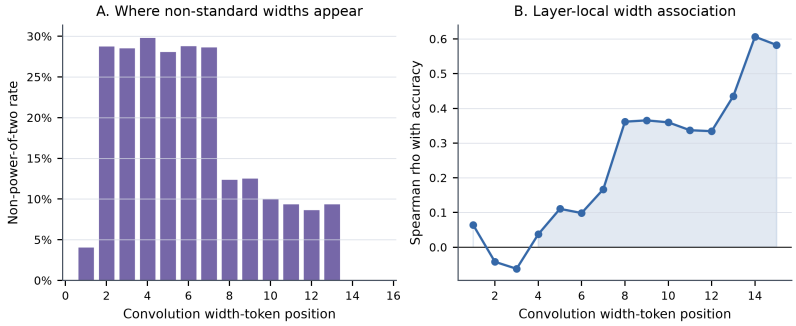

Fig. 5. Layer-local channel statistics. Left: non-power-of-two width rate by convolutional layer index. Right: Spearman correlation between layer width and validation accuracy. Later layers show stronger positive association with accuracy than the input stem.

and pooling layers. This yields observed tensor shapes, parameter counts, and approximate multiply-accumulate counts (MACs) for all 462 verified networks. These measurements are architectural properties of the generated programs, while their association with accuracy should be interpreted under the fixed one-epoch proxy protocol.

Two complementary splits of the convolutional layers are used below. **Stage-wise** labels a conv layer as *early*, *middle*, or *late* by its position (first, middle, and last third of layers, respectively). **Resolution-wise** labels a layer by the spatial resolution of its output feature map (e.g., 32×32, 16×16, 8×8). The distinction matters because a layer's *width* (number of output channels) and its *feature map* (the spatial tensor (channels × height × width)) are different objects — a 192-channel layer at 32×32 is far more expensive than the same 192 channels at 8×8, because the convolution is applied at every spatial position. Resolution-wise grouping is therefore the more informative split for capacity analysis.

The resolution-wise analysis is shown in Fig. 6. Top-decile proxy models allocate substantially less primary-convolution parameter share to `32x32` layers and substantially more to lower-resolution layers. In the full verified set (N=462), the share of primary-convolution parameters located at `8x8` or lower resolution has Spearman correlation $\rho = 0.477$ with one-epoch accuracy ($p < 10^{-26}$). Among the N=119 models that contain `4x4` convolutions, the mean `4x4` channel width has a similar-magnitude association ($\rho = 0.516$, $p < 10^{-8}$); the larger coefficient reflects the more restricted parameter range of this subset, not a stronger underlying signal.

To test whether the resolution effect is a proxy for larger models, candidates are compared within matched parameter-count quartiles. Quartile here means the 462 strict candidates are sorted by parameter count and split into four equal-sized groups Q1–Q4 (Q1 = smallest 25%, Q4 = largest 25%); within each quartile, the candidates are then split into a *lower* and a *higher* half by their low-resolution primary-convolution parameter share. Fig. 7 A shows the

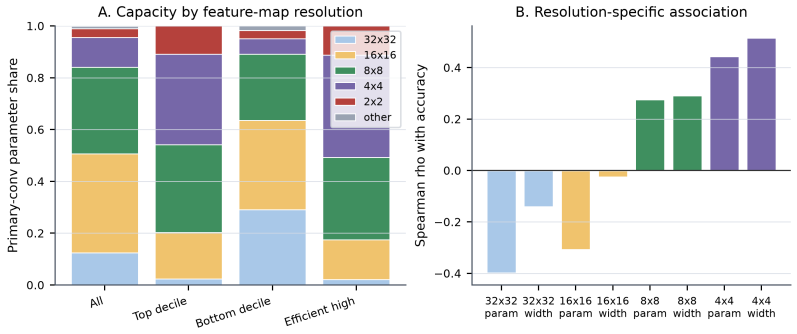

Fig. 6. Resolution-aware dynamic analysis. Left: primary-convolution parameter allocation by output feature-map resolution. Top-decile and efficient high-accuracy candidates allocate more capacity to lower-resolution stages than bottom-decile candidates. Right: resolution-specific Spearman associations with one-epoch accuracy.

result: the higher low-resolution group wins in all four quartiles, with a gap of approximately 0.030 accuracy points in Q2, Q3, and Q4 and one-sided Welch tests giving $p < 10^{-3}$ in those three bins. If the resolution effect were only a size artifact, the two bars within each quartile would be roughly equal; they are not, so the resolution effect carries information beyond total parameter count.

A complementary question is whether architecture features extracted from generated code can *predict* one-epoch proxy performance at all. A random-forest regressor is trained on dynamic architecture features (resolution shares, layer-position ratios, log-parameter count, depth, and so on) and asked to predict each candidate's one-epoch accuracy, with leave-cycle-out cross-validation so that the model is tested only on cycles it has not seen. Fig. 7 B shows the result: each dot is one candidate, the x-coordinate is the observed accuracy and the y-coordinate is the predicted accuracy, with the red diagonal marking perfect prediction. The points follow the diagonal loosely: leave-cycle-out $R^2 = 0.219$ with Pearson $r = 0.476$ and Spearman $\rho = 0.592$ (RMSE 0.040, roughly 4 accuracy points). The most important features are low-resolution parameter share, log-parameter count, middle-over-early MAC allocation, late-stage mean width, and depth. The remaining $\sim$78% of the variance is consistent with the noise floor of one-epoch CIFAR-100 training and LLM sampling: the signal is informative about short-horizon learning without fully determining the proxy outcome.

7.4 Confound Audit for Training and Hidden Code Changes

The performance signal could in principle come from three places: (i) the channel configuration; (ii) drift in the training protocol, evaluation metadata, or non-channel architectural choices (stride, downsampling, pooling, dropout, weight decay, Nesterov); or (iii) the one-epoch training stochasticity itself. This section audits (ii) and shows that the channel signal in (i) survives the controls.

The audit is summarized in Fig. 8. **Panel A** plots the partial Spearman ρ between each of four channel features and one-epoch accuracy under progressively stronger controls — *Raw*, *Size+cycle*, *Full* (size, cycle, depth, MACs, layer-count

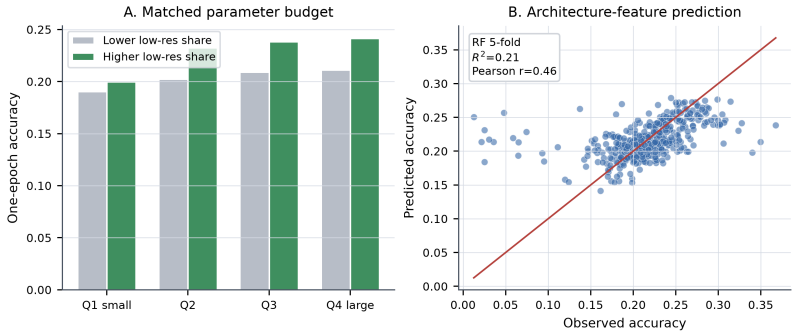

Fig. 7. Matched-budget and predictive analyses. **A**: within each parameter-count quartile, candidates with higher low-resolution primary-convolution parameter share (green) outperform the lower half (gray), so the resolution effect is not a size artifact. **B**: predicted vs. observed one-epoch accuracy from a leave-cycle-out random forest on architecture features ($R^2 = 0.22$, Pearson $r = 0.48$).

features, stride, downsampling, pooling, evaluation settings, and hidden-code flags), and *+train diag.* (full controls plus train loss and gradient norm). The blue line for low-resolution share starts at $\rho \approx 0.48$ under *Raw* and only drops to $\rho \approx 0.17$ under *+train diag.*; the other channel lines follow a similar pattern, so the signal does not collapse as controls tighten. **Panel B** compares three feature sets for leave-cycle-out rank prediction: non-channel *Controls*, *Controls + channel*, and *Channel only*. The *Channel only* bar (green) is taller than the *Controls* bar (gray), and adding controls on top of channels does not improve further. **Panel C** splits the 462 strict candidates by accuracy into *deciles* (10% groups, ∼46 candidates each) and compares the bottom-decile mean (gray) to the top-decile mean (blue) for two channel features (*Low-res*, *Mid/early*) and three hidden-code flags (*Nest.* = Nesterov, *WD* = weight decay, *Drop.* = dropout). The channel features differ sharply between top and bottom; the hidden-code features do not. The accuracy gap is in the channels, not in the training code.

Under the controls described in Panel A, primary low-resolution parameter share remains positively associated with one-epoch accuracy ($\rho = 0.139$, $p = 0.003$), and the middle-over-early channel-allocation ratio remains similarly positive ($\rho = 0.170$, $p = 2.5 \times 10^{-4}$). Adding post-training diagnostics such as train loss and gradient norm leaves the low-resolution association positive ($\rho = 0.170$, $p = 3.8 \times 10^{-4}$); these diagnostics are partly a consequence of the architecture and serve as a sensitivity check rather than as primary controls.

8 Execution Validity and Failure Modes

Failure modes are part of the generated-code search process. Among the 1348 code/error candidates, the most frequent categories are generic runtime errors (426), API signature mismatches (370), missing required functions (192), checksum duplicates (185), and shape mismatches (121); these labels are deterministic regex categories over error traces and are descriptive rather than a complete

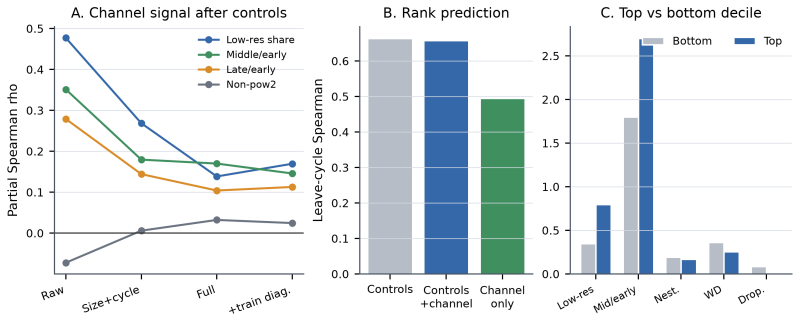

Fig. 8. Confound audit for the channel-configuration claims. **A**: partial Spearman ρ of four channel features under progressively stronger control sets. **B**: leave-cycle-out rank prediction for non-channel controls, controls plus channel features, and channel features only. **C**: top vs. bottom accuracy decile for two channel features and three hidden-code flags. Panel details are described in the text.

Table 3. Top strict CIFAR-100 candidates. Channel configurations are shown in run-length form.

Candidate	Accuracy	Params	Depth	Channel configuration
A6/B190	0.3676	11.8M	12	64, 192x3, 724x3, 256x5
A5/B99	0.3495	6.9M	12	64, 192x3, 384x3, 256x5
A6/B102	0.3400	5.0M	10	64, 192x3, 384x3, 256x3
A4/B152	0.3276	20.9M	10	128, 256x3, 512x3, 1024x3
A5/B146	0.3235	10.6M	13	64, 192x3, 576x3, 384x3, 256x3
A0/B134	0.3144	166.5M	16	256, 512x3, 1024x3, 2048x3, 1024x3, 2048x3
A5/B145	0.3084	6.0M	13	64, 150x3, 384x3, 256x6
A7/B43	0.3068	10.1M	15	64x3, 256x3, 512x3, 256x3, 512x3

compiler taxonomy. Scaled studies of LLM-based NAS should preserve raw generated code, training logs, evaluation metadata, rejected error traces, and enough identifiers to reconstruct candidate lineage. In this run, strict filtering removed non-target dataset evaluations and `1.json`-only records that would otherwise distort the accuracy trajectory.

9 Reproducibility Statement

Every quantitative claim in this paper is regenerated from the raw experiment directory by a single self-contained analysis script. The script parses the candidate folders of cycles `A0`–`A8`, applies the strict filtering rule of Sect. 4 (`1.json` plus `eval_info.json` verification of `cifar-100`), and rebuilds all tables, all figures, and the numeric macros injected into this document through `generated/rrpr_numbers.tex` and `generated/confound_numbers.tex`, with-

out relying on any pre-existing processed data frames. The generation stack is fixed and stated in Sect. 3: OlympicCoder-7B [15] as the base code model; LoRA adapters with rank 32, $\alpha = 32$, dropout 0.05 on the q, k, v projection modules; a fresh copy of the base model at the start of each cycle; and a fixed one-epoch CIFAR-100 proxy evaluation for every candidate. Generated networks, evaluation metadata, and candidate lineage identifiers are stored in the LEMUR/NNGPT ecosystem [6,9,17]. The preserved artifact comprises the raw candidate folders (generated `new_nn.py` programs, training logs, error traces, `1.json` and `eval_info.json` records for all 2000 candidates, including failures and non-target evaluations), the analysis script, and the generated macro files, so that the complete accounting of Table 1 — not only the successes — can be independently re-derived. The compute footprint of the strict evaluations is reported in Sect. 3 (approximately 7.4 GPU-hours in total; median 35 s per one-epoch candidate; median peak GPU memory 21.5 GB).

10 Limitations and Future Work

Scope of the experiment. The experiment is one execution of the closed loop with OlympicCoder-7B, CIFAR-100, and a one-epoch proxy. Other code LLMs may yield different channel priors, and other datasets or longer schedules may re-rank candidates. The confound audit controls for non-channel code variation, but not for LLM identity, dataset identity, or training horizon. The generated samples are also not i.i.d., since later cycles depend on LoRA fine-tuning from earlier cycles.

Sampling-budget considerations. A0's best (0.3144) and A6's best (0.3676) differ by 0.053 absolute accuracy. With the observed A0 mean of 0.212 and per-cycle standard deviation of about 0.030, the expected maximum of a 250-sample distribution is roughly 0.31, while the expected maximum of the 54-sample A6 cycle is about 0.27. A second run with the same prompt, LLM, and protocol would produce a different frontier and would help separate a true accuracy shift from chance on a single realization.

Partial signal from architecture features. Resolution-specific Spearman values (0.48–0.52) and the leave-cycle-out random-forest model ($R^2 = 0.219$, Spearman $\rho = 0.592$) capture an informative but partial signal: roughly three quarters of one-epoch accuracy variance is not explained by the extracted features, consistent with one-epoch training stochasticity and LLM sampling being substantial noise sources. Width tokens are extracted from constructor arguments under a `torch.nn`-compatible stub, and tensor shapes and MACs are measured on a single dummy forward pass, so they approximate rather than replicate the trained network's behavior. No candidate was trained for more than one epoch, and a full-training run on the top candidates would clarify whether one-epoch rankings correlate with final rankings.

Future work. Future work should test whether one-epoch rankings correlate with final rankings, and should perform controlled channel-allocation interventions in which depth and parameter budget are held fixed while channels are

moved from early to later stages or between power-of-two and non-power-of-two middle widths. The partial `A8` folder also indicates that robust experiment orchestration remains an open practical concern.

11 Conclusion

The scaled experiment shows that closed-loop LLM channel search improves both accuracy and parameter efficiency with increased candidate budget. From 2000 generated candidates in 8 complete cycles, 462 are strict CIFAR-100 evaluations; the high-performing frontier improves from 0.3144 to 0.3676, with A6/B102 reaching 0.3400 at 5.0M parameters and A0/B134 at 166.5M. The confound audit confirms that the main training protocol is effectively fixed for 461 of 462 strict evaluations. For LLM-driven NAS, scaled reporting should include the full generation ledger: successes, failures, target alignment, architecture statistics, confound audits, and sampling budget.

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

References

1. Abdelfattah, M.S., Mehrotra, A., Dudziak, Ł., Lane, N.D.: Zero-cost proxies for lightweight NAS. arXiv preprint arXiv:2101.08134 (2021)
2. Cai, Z., Tang, Y., Lai, Y., Wang, H., Chen, Z., Chen, H.: SEKI: Self-evolution and knowledge inspiration based neural architecture search via large language models. arXiv preprint arXiv:2502.20422 (2025)
3. Chee, E., Wu, Z.: Airnet: Self-supervised affine registration for 3d medical images using neural networks. arXiv preprint arXiv:1810.02583 (2018)
4. Chen, A., Dohan, D.M., So, D.R.: EvoPrompting: Language models for code-level neural architecture search. In: Advances in Neural Information Processing Systems (2023)
5. Dettmers, T., Pagnoni, A., Holtzman, A., Zettlemoyer, L.: QLoRA: Efficient finetuning of quantized LLMs. In: Advances in Neural Information Processing Systems (2023)
6. Goodarzi, A.T., Kochnev, R., Khalid, W., Goudarzi, H.T., Qin, F., Uzun, T.A., Dhameliya, Y.S., Kathiriya, Y.K., Bentyn, Z.A., Ignatov, D., Timofte, R.: LEMUR neural network dataset: Towards seamless AutoML. arXiv preprint arXiv:2504.10552 (2025), https://arxiv.org/abs/2504.10552
7. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: AMC: AutoML for model compression and acceleration on mobile devices. In: Proceedings of the European Conference on Computer Vision (ECCV) (2018)
8. Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W.: LoRA: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685 (2021)
9. Kochnev, R., Khalid, W., Uzun, T.A., Zhang, X., Dhameliya, Y.S., Qin, F., Vysyaraju, C., Duvvuri, R., Goyal, A., Ignatov, D., Timofte, R.: NNGPT: Rethinking AutoML with large language models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops. pp.

3262–3272 (June 2026), https://openaccess.thecvf.com/content/CVPR2026W/CVPR-NAS26/html/Kochnev_NNGPT_Rethinking_AutoML_with_Large_Language_Models_CVPRW_2026_paper.html
10. Li, Z., Lin, Z., Wang, Y.: CoLLM-NAS: Collaborative large language models for efficient knowledge-guided neural architecture search. arXiv preprint arXiv:2509.26037 (2025)
11. Liu, H., Simonyan, K., Yang, Y.: DARTS: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)
12. Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.T., Sun, J.: MetaPruning: Meta learning for automatic neural network channel pruning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (2019)
13. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE International Conference on Computer Vision (2017)
14. Nasir, M.U., Earle, S., Cleghorn, C.W., James, S., Togelius, J.: LLMatic: Neural architecture search via large language models and quality diversity optimization. arXiv preprint arXiv:2306.01102 (2023)
15. Penedo, G., Lozhkov, A., Kydlíček, H., Allal, L.B., Beeching, E., Lajarín, A.P., Gallouédec, Q., Habib, N., Tunstall, L., von Werra, L.: Olympiccoder. https://huggingface.co/open-r1/OlympicCoder-7B (2025)
16. Uzun, T.A., Ignatov, D., Timofte, R.: Closed-loop LLM discovery of non-standard channel priors in vision models. In: Proceedings of the International Conference on Pattern Recognition (ICPR) (2026), https://arxiv.org/abs/2601.08517, to appear
17. Uzun, T.A., Khalid, W., Din, S.U., Mulukuledu, S.R., Singh, A., Vysyaraju, C., Duvvuri, R., Goyal, A., Lukhi, Y.R., A Hussain, M., Jesani, K., Shrestha, U., Mittal, Y., Kochnev, R., Kadam, P., Ikram, M., Moradiya, H., Arslanian, A., Ignatov, D., Timofte, R.: LEMUR 2: Unlocking neural network diversity for AI. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops. pp. 3291–3300 (June 2026), https://openaccess.thecvf.com/content/CVPR2026W/CVPR-NAS26/html/Uzun_LEMUR_2_Unlocking_Neural_Network_Diversity_for_AI_CVPRW_2026_paper.html
18. Yu, C., Liu, X., Wang, Y., Liu, Y., Feng, W., Xiong, D., Tang, C., Lv, J.: GPT-NAS: Evolutionary neural architecture search with the generative pre-trained model. arXiv preprint arXiv:2305.05351 (2023)
19. Yu, J., Huang, T.S.: AutoSlim: Towards one-shot architecture search for channel numbers. arXiv preprint arXiv:1903.11728 (2019)
20. Yu, K., Sciuto, C., Jaggi, M., Musat, C., Salzmann, M.: Evaluating the search phase of neural architecture search. arXiv preprint arXiv:1902.08142 (2019)
21. Zheng, M., Su, X., You, S., Wang, F., Qian, C., Xu, C., Albanie, S.: Can GPT-4 perform neural architecture search? arXiv preprint arXiv:2304.10970 (2023)
22. Zhou, D., Zhou, X., Zhang, W., Loy, C.C., Yi, S., Zhang, X., Ouyang, W.: EcoNAS: Finding proxies for economical neural architecture search. arXiv preprint arXiv:2001.01233 (2020)
23. Zhou, X., Wu, X., Feng, L., Lu, Z., Tan, K.C.: Design principle transfer in neural architecture search via large language models. arXiv preprint arXiv:2408.11330 (2024)
24. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)