From Brute Force to Semantic Insight: Performance-Guided Data Transformation Design with LLMs

Anonymous ACL submission

Abstract

Large language models (LLMs) have achieved notable performance in code synthesis; however, data-aware augmentation remains a limiting factor, handled via heuristic design or brute-force approaches. We introduce a performance-aware, closed-loop solution in the NNGPT ecosystem of projects that enables LLMs to autonomously engineer optimal transformations by internalizing empirical performance cues. We fine-tune LLMs with Low-Rank Adaptation on a novel repository of 6,000+ empirically evaluated PyTorch augmentation functions, each annotated solely by downstream model accuracy. Training uses pairwise performance ordering (better–worse transformations), enabling alignment through empirical feedback without reinforcement learning, reward models, or symbolic objectives. This reduces the need for exhaustive search, achieving up to 600× fewer evaluated candidates than brute-force discovery while maintaining competitive peak accuracy and shifting generation from random synthesis to task-aligned design. Ablation studies show that structured Chain-of-Thought prompting introduces syntactic noise and degrades performance, whereas direct prompting ensures stable optimization in performance-critical code tasks. Qualitative and quantitative analyses demonstrate that the model internalizes semantic performance cues rather than memorizing syntax. These results show that LLMs can exhibit task-level reasoning through non-textual feedback loops, bypassing explicit symbolic rewards. Upon acceptance, we will publicly release all code, prompts, and generated artifacts to support reproducibility and community use.

1 Introduction

A neural network's performance depends on both its architecture and how the data is preprocessed. Effective data preprocessing and augmentation are essential for model generalization and convergence. Numerous studies have investigated optimal strategies for data transformations and the development of novel methods. For instance, AutoAugment (Cubuk et al., 2019a) applies reinforcement learning to identify effective augmentation policies, while Meta Learning (Bilalli et al., 2018) approach uses a predictive meta-model to suggest data transformations for a specific classification algorithm. As generative artificial intelligence becomes increasingly prevalent, recent studies have begun to explore code generation using large language models (LLMs), leveraging their generative capabilities to propose and assess complex, data-aware solutions.

The NNGPT framework (Kochnev et al., 2025a,b) previously established a methodology for synthesizing neural network architectures using LLMs. Extending this work, we automate the generation and evaluation of data augmentation functions within the NNGPT ecosystem. This study also addresses the limited diversity of data augmentations in the LEMUR dataset, which comprises a broad range of high-capacity and edge-optimized neural network models (Goodarzi et al., 2025; Uzun et al., 2026; Din et al., 2025) and serves as the knowledge base for the NNGPT. Motivated by recent advances in LLM applications across multiple domains (Gado et al., 2025; Rupani et al., 2025; Khalid et al., 2025) and prior NNGPT experiments (Jesani et al., 2025; Vysyaraju et al., 2025; Mittal et al., 2025; Khalid et al., 2026), we curate a diverse set of PyTorch data transformation functions and systematically evaluate their effects, producing performance-annotated metadata that links each code snippet to its impact on model training. This metadata is used to fine-tune the language model, enhancing its understanding of data variability and performance effects. We further implement a system that generates PyTorch data transformation functions and iteratively refines the generator through supervised fine-tuning.

2 Related Works

The automated generation of data augmentations (Yang et al., 2023) has evolved significantly. The process began with AutoAugment (Cubuk et al., 2019a), which uses reinforcement learning to search a fixed list of operations. This approach defines a discrete search space of 14 to 16 standard image processing methods, such as Rotate, ShearX, and ShearY. In this approach, generation refers to finding an optimal policy composed of sub-policies that specify two sequential transformation operations, the probability of applying each operation, and the magnitude of each operation. This method transforms the problem into a large-scale discrete search challenge, with AutoAugment's search space containing approximately 10^{32} possible policies.

The high computational cost of automated search in AutoAugment led to approaches that simplified the generation process. RandAugment (Cubuk et al., 2019b) demonstrated that a complex search algorithm is unnecessary. Instead, it automated generation by reducing the search space to two interpretable hyperparameters: N (the number of transformations to apply) and M (a single, global magnitude for all transformations). RandAugment randomly samples N transformations from a predefined list and applies them with magnitude M. This generation method matched the performance of AutoAugment, indicating that the diversity of the transformation space is more critical than the complexity of the generation algorithm.

A key limitation of both AutoAugment and RandAugment is that they generate a policy that is applied to every image. However, a policy that is good for one image may be harmful to another (Aboudeshish et al., 2025). This led to the development of automated generation frameworks that are instance-specific and creates a unique augmentation policy for each individual image (Minh et al., 2021). For each image, a Deep Q-Network (DQN) iteratively generates a policy by selecting an action from a list of transformations or a "Stop" action. This process generates a unique, optimal chain of transformations for every sample in the dataset. This is a far more granular method of "generating a large number of transforms," as it generates one policy per instance rather than one policy per dataset.

Later research aimed to generate more effective and diverse transforms by expanding the space of possible transformations (Mumuni and Mumuni, 2025b). This expansion of the generation space occurred in two ways: through learned transformations and generative models. This approach enables the model to learn the transformation function itself. For example, Spatial Transformer Networks (STNs) (Mumuni and Mumuni, 2025b) can be integrated into a model to learn optimal affine transformations directly from the data.

The latest paradigm in automated generation utilizes Large Language Models (LLMs) as the primary generation engine (Mumuni and Mumuni, 2025a, Ding et al., 2024). Recent approaches to adapting LLMs for specific tasks have shifted toward Instruction Tuning and Supervised Fine-Tuning (SFT) (Parthasarathy et al., 2024, Chung et al., 2024). Ouyang et al. (2022) showed that fine-tuning models on human-written instructions aligns them more closely with user intent than simply increasing model size. This method is also effective in specialized domains, such as Python programming (Bai et al., 2022). In code generation, where models must address complex and functional requirements beyond basic text completion, such alignment is crucial. Chen et al. (2021) further confirmed this by demonstrating that optimizing general-purpose LLMs on code corpora enhances performance on functional correctness benchmarks.

Recent research in LLM adaptation also highlights the significance of data quality and training strategies over sheer data quantity. Longpre et al. (2023) identified that task balancing and enriching training data, such as by inverting input-output pairs, are essential for improving generalization. Their results indicate that combining zero-shot, few-shot, and Chain-of-Thought (CoT) data during fine-tuning leads to better performance across evaluation settings.

The structure of input prompts and the training data have a significant impact on the quality of generated outputs, even though SFT updates model weights. A systematic survey by Sahoo et al. (2025) categorizes advanced prompting techniques, including Chain-of-Thought (CoT) and decomposed prompting, which are essential for guiding models through multi-step logical tasks such as complex data augmentation. Building on these findings, Kojima et al. (2023) demonstrated that LLMs function as effective "zero-shot reasoners" and can perform complex task by simply appending the prompt "Let's think step by step." This approach

enables the creation of reasoning-dense training data without the need for manual annotation.

In the field of code generation, AceCoder (Li et al., 2023b) addresses the challenge of requirement understanding through a guided code generation mechanism. This approach prompts the model to produce intermediate outputs, such as test cases or clarifications, prior to generating the final code. It instructs the model on "what to write" before determining "how to write it." Furthermore, LAIL (LLM-Aware In-Context Learning) (Li et al., 2023a) was introduced to ensure that high-quality examples are utilized during training or inference. This method filters training data based on the model's preferences, rather than relying on heuristic text similarity metrics, by employing a teacher LLM to estimate the likelihood that a given example will facilitate ground truth generation. These methods highlight a shift toward employing LLMs as active participants in code construction and quality assurance workflows, rather than just as final predictors.

3 Methodology

3.1 LLM based code generation

We used the Olympic Coder 7B (Hugging Face, 2025) model, an open-source AI model developed by Hugging Face. It is specifically designed to address complex olympiad-level programming problems and is fine-tuned on the CodeForces-CoTs (Penedo et al., 2025) dataset. We carried out code generation through various prompting approaches (Schulhoff et al., 2025, Sahoo et al., 2025) including Zero-shot prompting, Role prompting, Constraint prompting, and Chain-of-thought prompting methods. However, we noticed issues such as identical transform functions and syntax errors in the generated output.

3.2 Brute Force Approach

After initial approaches yielded limited improvements, we adopted a manual method. First, we designed a system to automatically generate image transformation functions using the PyTorch (Paszke et al., 2019) torchvision (maintainers and contributors, 2016) package. The available transforms were organized into a dictionary. Given two parameters, the total number of files and the number of augmentations per file, we generated a number of transformation scripts. Each file contained one, two, or three selected transforms from the dictionary, in addition to fixed transforms: resize$(64, 64)$, to tensor, and normalization. The generator permuted and cycled through various transform combinations to generate the files, assigning random parameters to each selected transform function. In total, 6,000 transform files were generated and evaluated, 2,000 for each case of using one, two, or three variable transforms.

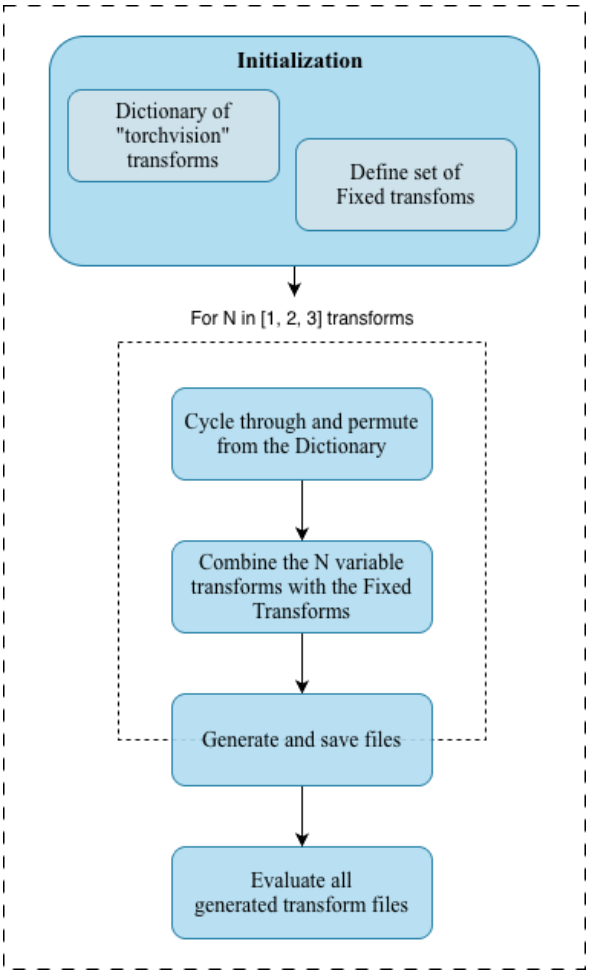

Figure 1: Brute-force data transformation generation and evaluation pipeline for constructing an LLM fine-tuning dataset. Image transformation functions are automatically generated, evaluated under a fixed training configuration, and stored with their corresponding accuracy, yielding a performance-labeled dataset used in subsequent LLM fine-tuning.

4 Fine Tuning LLM

We employed an iterative instruction fine tuning approach that alternates between generating data transformations, evaluating their performance, and using the resulting metadata to refine the language model. To enable efficient adaptation without the computational overhead of full-parameter

fine-tuning, we employed Low-Rank Adaptation (LoRA) (Hu et al., 2021) configured as shown in Listing 1 along with other hyperparameters. We utilized the set of generated image transforms and their evaluations obtained from the brute-force technique.

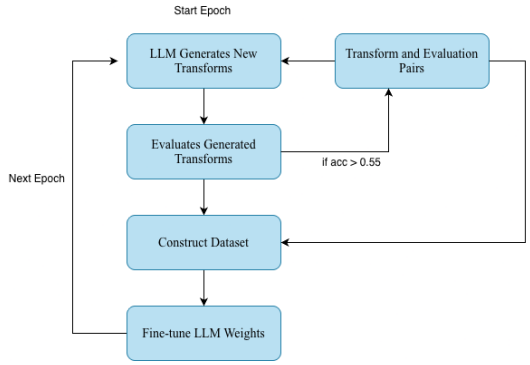

Figure 2: The iterative performance-guided fine-tuning loop. In each cycle, the LLM synthesizes candidate data transformations which are empirically validated via downstream model training. High-fidelity metadata from these trials is used to update model parameters via LoRA, inducing a semantic alignment between generative output and empirical performance cues.

$$\text{LLM}_{\theta_t} \xrightarrow{\text{Generate}} \{T_1, \ldots, T_n\} \xrightarrow{\text{Evaluate}} \{(T_i, \text{Acc}_i)\} \xrightarrow{\text{Filter}} \mathcal{D}^{(t)} \xrightarrow{\text{Fine-Tune}} \text{LLM}_{\theta_{t+1}}$$

First, a prompt is constructed using a few-shot strategy from a given prompt template (Listing 3) by randomly selecting seed transforms(T_i) from the training data$\mathcal{D}^{(t)}$. The LLM(LLM_{θ_t}) then generated several transforms that utilize common patterns from the references to optimize for specific task in each iteration(t). Each generated transform was evaluated using the same hyperparameters as the brute-force method. Generated data is filtered to identify better examples, specifically looking for instances where the generated transform improved upon the baseline accuracy. A dataset is constructed by iterating through each transform ("A") and searching for another transform as an 'add-on' transform ("B") with a higher accuracy to generate training pairings in which B outperforms A. This collection of "B better than A" pairs is formatted into instruction-tuning pairs using another prompt template (Listing 2) that serves as the fine-tuning dataset.

```
hyperparameters = {
    # LoRA Adapter Configuration

    # Rank of update matrices
    "r": 32,
    # Scaling factor
    "lora_alpha": 32,
    "lora_dropout": 0.05,
    "bias": "none",
    # Adapters applied to all attention
       projections
    "target_modules": [
        "q_proj", "k_proj",
        "v_proj", "o_proj"
    ],

    # Optimization Strategy
    "optimizer": "paged_adamw_8bit",
    "learning_rate": 1.5e-4,
    "lr_scheduler_type": "cosine",
    "warmup_ratio": 0.05,
    # Epochs per fine-tuning iteration
    "num_train_epochs": 3,

    # Batch Size
    "per_device_train_batch_size": 1,
    "gradient_accumulation_steps": 8,
    "effective_batch_size": 8,

    # Generation & Sampling
    # Controls diversity
    "temperature": 0.8,
    # Nucleus sampling
    "top_p": 0.9,
    "top_k": 70,
    "max_new_tokens": 16 * 1024
}
```

Listing 1: Hyperparameter configuration for LoRA fine-tuning and generation.

```
"prompt" : [
    "You are an expert image
       transformation optimizer.",
    "Baseline transform code (Accuracy
       : {accuracy}):",
    "<tr>{transform_code}</tr>",
    "Generate an improved Python
       transform function ('transform
       ') that achieves a higher
       accuracy with 1 epoch, batch
       64, lr 0.01, and momentum 0.9
       for 'cifar-10' dataset and
       task: 'img-classification' ",
    "Your response MUST contain
       exactly one set of the XML
       tags <tr>...</tr>. DO NOT
       include any leading or
       trailing text, markdown fences
        (```), comments, or any other
        XML tags like <path> or <text
       >."
  ],
  "output" : [
    "<tr>{addon_transform_code}</tr>"
  ]
```

Listing 2: Prompt used for fine-tuning

```
"prompt": [
      "You are an expert image transformation generator.",
      "Your task is to generate new image transformation code.",
      "Use common patterns and ideas from the following two reference transforms:",
      "Reference 1 (Accuracy: {accuracy}):",
      "<tr>{transform_code}</tr>",
      "Reference 2 (Accuracy: {addon_accuracy}):",
      "<tr>{addon_transform_code}</tr>",
      "Provide a new, high-performance transform for 'cifar-10' (task: 'img-classification') for training with 1 epoch, batch 64, lr 0.01, and momentum 0.9",
      "Respond with:",
      "1. A <tr> XML tag containing the complete Python transform code (function name 'transform').",
      "The code must be wrapped strictly in <tr> and </tr> tags."
    ]
```

Listing 3: Prompt used for generation

5 Experiments and Results

All data transformation functions were evaluated for the image classification task using a ResNet (He et al., 2016) model on the CIFAR-10 (Krizhevsky et al., 2009) dataset. The model was trained for 1 epoch with a batch size of 64, a learning rate of 0.01, a momentum of 0.9, and a dropout rate of 0.2, due to resource and time constraints. All experiments, including the brute-force search and iterative fine-tuning loops, were conducted on a local workstation with a single NVIDIA GeForce RTX 4090 GPU (24 GB VRAM) which demonstrates the accessibility of our method for researchers with limited computational resources.

The Constraint method achieved the highest performance among LLM-based generation techniques. In this approach, the LLM was directed to modify a specified transform. Of the LLM generated transforms, approx. 22% were syntactically correct. The wide confidence interval of $[0.0644, 0.1436]$ and the mean accuracy of 0.1040, as shown in Table 1, indicate that LLMs without fine-tuning may not be optimal for generating transforms. The highest accuracy achieved was 0.5728, using the RandomResizedCrop, ColorJitter, RandomHorizontalFlip, GaussianBlur, ToTensor, and Normalize transforms.

Configuration	Mean Accuracy	Best Accuracy	95% Confidence Interval
LLM generated(without fine-tuning)	0.1040	0.5728	[0.0518, 0.1563]
1 transform selected	0.5256	**0.6124**	[0.5234, 0.5279]
2 transforms selected	0.4832	0.6071	[0.4801, 0.4863]
3 transforms selected	0.4401	0.5983	[0.4363, 0.4439]

Table 1: Performance metrics for the brute-force generation pipeline. Baseline results for the non-fine-tuned LLM are compared against systematic permutations of multiple torchvision transforms. These results provided the performance-labeled metadata required for subsequent iterative supervised fine-tuning.

The brute-force approach generated 6,000 transforms. This method achieved a maximum accuracy of 0.6124 using the RandomPosterize, Resize, ToTensor, and Normalize transforms. Data transformations with a single selected transform generally outperformed those utilizing two or three transforms, as indicated in Figure 3 and Table 1, which show an increased confidence interval with a higher number of selected transforms. Several data transformations that outperformed the best-performing transform in the LEMUR dataset, when trained with identical hyperparameters and the CIFAR-10 dataset using ResNet, were incorporated into the LEMUR dataset.

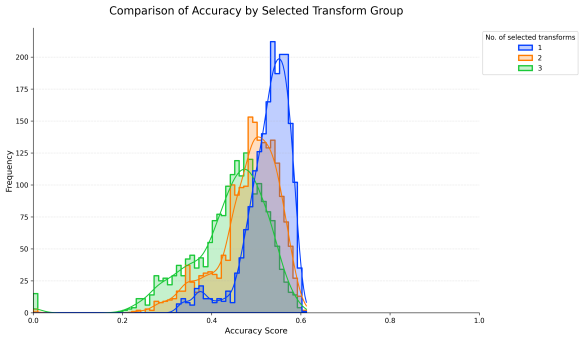

Figure 3: Accuracy distribution of data transformation functions grouped by the number of selected transforms. Single-transform configurations exhibit higher mean accuracy and lower variance compared to compositions of multiple transforms.

The fine-tuning process began with a curated dataset comprising 2,361 pairs of transforms and evaluations obtained through a brute-force approach. Data redundancy was reduced by removing duplicate transform files. Additionally, 1,180 augmented samples were added. Each sample had the input Resize parameter explicitly set to 256. After this initial configuration, the dataset was dynamically expanded by incorporating any new LLM-generated transform with an accuracy greater than 0.55 into the training set for subsequent iterations. The performance of the generated data transform

functions was tracked over 28 fine-tuning epochs (A0 to A27), with 10 transforms generated per epoch. Figure 4 shows the LLM's performance during the fine-tuning loop which includes the mean accuracy of all valid transformations and the maximum accuracy achieved by the best single transformation per epoch.

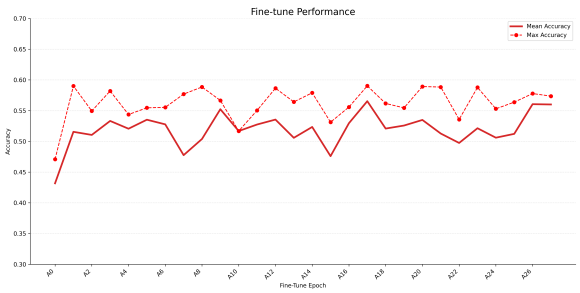

Figure 4: Mean and maximum accuracy of generated transformations across fine-tuning epochs. The mean accuracy exhibits a steadily upward trajectory (r = 0.34) over successive epochs, indicating improved overall generation quality, while the maximum accuracy remains relatively stable, suggesting consistent rediscovery of high-performing transformations.

The mean accuracy increased from 0.4317 at epoch A0 to 0.56 at epoch A27. This improvement demonstrates that the fine-tuning process effectively aligned the LLM weights with high-performing transform code. The model generated candidates that led to better convergence after exposure to more positive examples. Maximum accuracy did not show a clear monotonic increase but remained stable. It indicates the model reliably rediscovered or slightly improved the best-known solutions. The gap between mean and maximum accuracy narrowed in final epochs, showing reduced variance. The model generated more consistently effective transformations and fewer low-quality outliers as shown in Figure 5. Qualitative code analysis confirmed that the model learned the inductive bias from the augmented dataset. The generator produced transforms with various resolution parameters, such as 224, 256, and 32. This result shows that the model developed a semantic relation between parameters like Resize(256) and high-accuracy rewards, rather than memorizing syntax.

5.1 Comparison against LEMUR Baseline

To contextualize the effectiveness of automated transform generation, we compared our findings to the best-performing transformations in the LEMUR

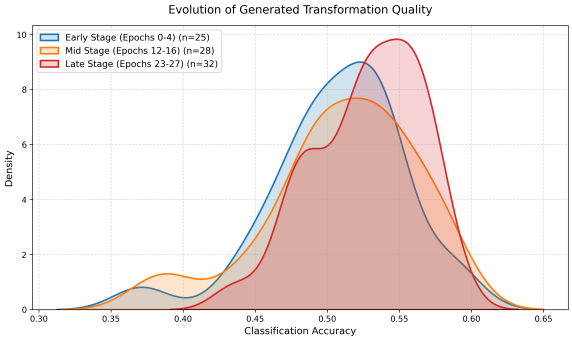

Figure 5: Evolution of Generated Transformation Quality. Kernel Density Estimation (KDE) of validation accuracy across fine-tuning stages. The shift from Early (blue) to Late (red) epochs indicates the model effectively minimizes the generation of low-performing code and converges on a high-performance semantic region

Methods	Peak Accuracy
LEMUR dataset	0.6533
LLM (without fine-tuning)	0.6329
Brute Force Approach	**0.6634**
Fine Tuned LLM	0.6339

Table 2: Comparison of peak classification accuracy across different augmentation generation strategies. All methods are evaluated under identical training configurations, enabling a direct comparison between predefined LEMUR transforms, brute-force generation, and LLM-based approaches.

dataset, as well as our initial Brute-Force approach. All baselines were evaluated using the same experimental constraints: a ResNet architecture trained on CIFAR-10 for a single epoch with a batch size of four, a learning rate of approximately 0.0102, and a momentum of approximately 0.8826.

Table 2 summarizes the highest accuracy using the predefined transforms in the LEMUR dataset, the best result from the top 150 brute force-generated transforms, and the maximum accuracy attained in the final epoch of iterative fine-tuning. The experimental results show that the Brute Force approaches slightly outperform the standard augmentation strategies found in the LEMUR dataset.

5.2 Efficiency of Fine-Tuning vs. Brute Force

Although high-performing transforms were successfully found using the brute-force search, 6000 candidates had to be generated and evaluated in order to identify them. In contrast, the Fine-tuned LLM showed better sample efficiency. Despite generating only 10 candidates per epoch (280 candidates in total), the model consistently produced transformations with accuracies over the 0.55 threshold. The fine-tuning process effec-

tively "distilled" the knowledge from the brute-force dataset into the model weights. The LLM did not need to explore thousands of random possibilities. Instead, it quickly converged on the optimal strategy. The refined model could provide competitive augmentations at a significantly higher rate than the random brute-force search.

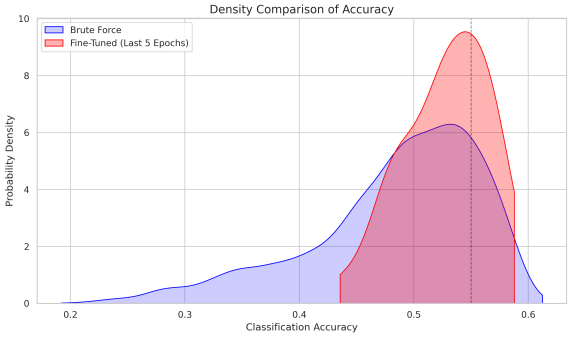

Figure 6: Impact of performance-aware SFT on transformation efficacy. Comparison of classification accuracies achieved by transforms generated at different stages of the iterative loop. The narrowing of the variance and the upward shift in the median accuracy provide empirical evidence that the LLM is successfully internalizing the semantic performance cues from the metadata repository.

6 Ablation Study

Effect of Redundancy in Dataset

We used a dataset of 6,000 pairs from the brute force approach, along with transforms generated from previous fine-tuning iterations to understand how data volume and redundancy affect the model's generative stability and convergence. Transforms files containing errors, such as invalid syntax, were also included in the later fine-tuning iterations and assigned an accuracy of 0.0. There was also significant redundancy, with multiple files containing identical transformation logic differing only by random seed values.

As shown in Figure 7, the curated dataset showed a positive trajectory, consistently outperforming fine-tuning with the Unfiltered dataset and achieving mean accuracies exceeding 0.56. Although the large volume of the dataset enabled the model to generate valid Python syntax, the duplicate files hindered the optimization. The model likely "memorized" frequent file patterns rather than learning to distinguish the specific semantic features that contribute to higher accuracy.

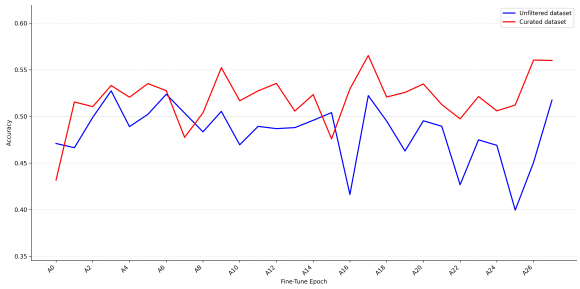

Figure 7: Impact of dataset composition on fine-tuning performance. Mean validation accuracy across fine-tuning epochs for curated and unfiltered datasets. The curated dataset yields more stable convergence and higher accuracy, while redundancy and noisy samples in the unfiltered dataset hinder semantic learning.

Effect of Prompt Engineering

To evaluate how the model responds to different instruction formats, we compared two approaches. Firstly, the Direct approach (Listing 3) used a simple prompt focused on code generation. Then Structured CoT (Chain-of-Thought) with Constraints (Listing 4) used a verbose prompt requiring analysis before code, along with explicit negative constraints.

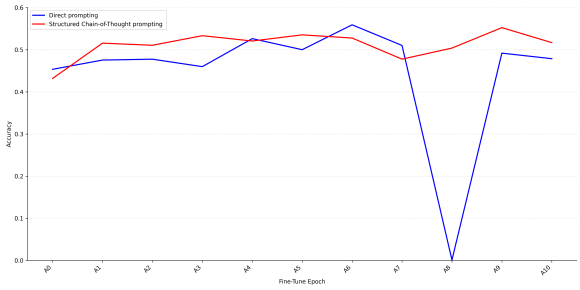

Figure 8: Comparative performance of Direct vs. Structured Chain-of-Thought prompting. The results indicate that Structured CoT is prone to optimization instability, evidenced by the total performance drop at epoch 8. The Direct approach remains more stable and effective, successfully internalizing semantic cues without the noise introduced by verbose reasoning requirements.

As shown in Figure 8, the Direct approach demonstrated better stability and convergence than the Structured approach. Although the mean accuracy peaked at epoch A6, it dropped below 0.48 by epoch A10. This drop suggests that the extra tokens needed for analysis introduced noise. The model struggled to balance generating clear reasoning with valid Python syntax.

The Direct approach shows a consistent positive trend comparatively, peaking above 0.55 at

epoch A10. Removing unnecessary instructions allowed the fine-tuning signal to focus on the target code. Moreover, removing the "Negative Constraints" and reasoning requirements concentrated the model's attention solely on the target output (the transform code). The inclusion of "Negative Constraints" may have mistakenly directed the model's focus towards the artifacts (SVG, HTML) that it was intended to avoid, or diluted the context window with unnecessary instructions.

```
"prompt": [
  "You are an expert image transformation generator.",
  "Your task is to synthesize a high-performance augmentation strategy with common patterns and ideas of two reference transforms.",
  "### Reference 1 (Acc: {accuracy})",
  "<tr>{transform_code}</tr>",
  "### Reference 2 (Acc: {addon_accuracy})",
  "<tr>{addon_transform_code}</tr>",
  "### Task",
  "Create a new 'transform' function for CIFAR-10 that combines the effective parts of both references.",
  "Target Settings: 1 epoch, batch 64, lr 0.01.",
  "### Instructions",
  "1. Briefly analyze why Ref 1 and 2 worked, and propose a strategy.",
  "2. <tr>: Write the executable Python code.",
  "### Negative Constraints",
  "- DO NOT output SVG, <path>, <g>, or HTML tags.",
  "- DO NOT output markdown fences (```).",
  "Respond strictly in this format:",
  "<tr>... code ...</tr>"
]
```

Listing 4: Structured CoT with Constraints Prompt used for generation

7 Conclusion

This work presents a performance-aware, closed-loop framework for the autonomous synthesis of data transformations, demonstrating that large language models can be effectively grounded in empirical training outcomes. By constructing a novel repository of over 6,000 empirically evaluated PyTorch augmentation functions and fine-tuning via Low-Rank Adaptation, we induce task-level reasoning in the generator without relying on explicit symbolic rewards, reinforcement learning, or handcrafted objectives.

Our experiments yield several insights relevant to the design of future automated code synthesis and learning systems. First, empirical alignment through iterative fine-tuning shifts the generative distribution from random code synthesis toward informed, task-aligned design, achieving up to a $600\times$ reduction in evaluated candidates compared to brute-force discovery while improving mean accuracy from 0.43 to 0.56 and preserving competitive peak performance. Second, qualitative and quantitative analyses show that the model internalizes semantic performance cues—such as the benefits of resolution scaling (e.g., `Resize(256)`)—rather than memorizing transformation syntax, indicating meaningful generalization beyond surface-level patterns. Third, ablation studies reveal a critical trade-off between prompt complexity and optimization stability: while direct prompting supports reliable improvement, structured Chain-of-Thought prompting introduces syntactic instability that leads to catastrophic performance degradation, underscoring the fragility of complex reasoning formats in performance-critical code-generation tasks.

Taken together, these results demonstrate that grounding LLMs in non-textual, empirical feedback loops provides a robust alternative to symbolic or reward-based alignment for complex downstream objectives. By addressing the limited diversity and supervision of existing augmentation datasets, this work lays a scalable foundation for autonomous machine learning pipelines. Future work will focus on improving syntactic robustness at larger model scales and extending this grounded reasoning framework to multimodal data, broader architectural families, and more heterogeneous optimization tasks.

8 Limitations

Generalization

We limited our experimental setup, which includes both the generative fine-tuning loop and evaluation, to the ResNet architecture and a single dataset. Furthermore, the augmentations generated are implicitly specialized for this specific configuration. We assume that the best augmentation strategy depends on the interactions between model architecture and data distribution rather than a single transformation function that works well in every situation. Since we did not extend the fine-tuning process to alternative architectures or datasets, we cannot quantify the LLM's ability to dynamically adapt its generation to discover the "best fit" solutions. Fur-

thermore, the constraint to a single-epoch training limits our assessment of the long-term convergence stability of the neural network.

Syntactic Instability

In contrast to the consistent increase in mean accuracy, we observed instability in the number of valid transformations generated. There were several missing imports, indentation errors, and forbidden XML tags in the transformation code. This suggests an imbalance between syntactic consistency and semantic learning. Although fine-tuning effectively prioritized the logic of augmentation to maximize the specified metric (accuracy), it sometimes compromised the structural constraints necessary for execution. This trade-off resulted in syntactical errors when the model attempted to generate complex code structures.

Exploration Saturation

Lastly, the Max Accuracy plateaued at about 0.60 early, suggesting that the optimization process likely reached a local optimum. This suggests that the model struggled to find better strategies in later epochs, even though it improved at replicating known good patterns. It also suggests a gap in the model's ability for exploration, implying that the self-improvement loop might favor safe, gradual improvements over drastically different and better approaches in the absence of additional mechanisms that encourage diversity.

9 Ethics Statement

Our research involves the training and evaluation of Large Language Models (LLMs) and neural networks, which incurs a significant computational cost. Specifically, the construction of our fine-tuning dataset required a baseline evaluation of more than 6,000 transformation files. We acknowledge the energy consumption associated with this initial data collection. However, the primary motivation of this work is to reduce such costs in future workflows. We used the CIFAR-10 and the LEMUR datasets for training and evaluation. We further used the Olympic Coder 7B model, an open-source large language model available via Hugging Face, for our code generation tasks. These resources are publicly available, standard benchmarks, and tools within the research community. We used AI tools to assist with language editing and proofreading. However, the research methods, data analysis, and all intellectual contributions are entirely our own.

References

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

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, and 12 others. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. *Preprint*, arXiv:2204.05862.

Besim Bilalli, Alberto Abelló, Tomàs Aluja-Banet, and Robert Wrembel. 2018. Intelligent assistance for data pre-processing. *Computer Standards Interfaces*, 57:101–109.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tai, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, and 16 others. 2024. Scaling instruction-finetuned language models. *J. Mach. Learn. Res.*, 25(1).

Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. 2019a. Autoaugment: Learning augmentation policies from data. *Preprint*, arXiv:1805.09501.

Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. 2019b. Randaugment: Practical automated data augmentation with a reduced search space. *Preprint*, arXiv:1909.13719.

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*.

Bosheng Ding, Chengwei Qin, Ruochen Zhao, Tianze Luo, Xinze Li, Guizhen Chen, Wenhan Xia, Junjie Hu, Anh Tuan Luu, and Shafiq Joty. 2024. Data augmentation using large language models: Data perspectives, learning paradigms and challenges. *Preprint*, arXiv:2403.02990.

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*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

Hugging Face. 2025. OlympicCoder-7B.

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*.

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*.

Waleed Khalid, Dmitry Ignatov, and Radu Timofte. 2026. From memorization to creativity: Llm as a designer of novel neural-architectures. *arXiv preprint*.

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:2511.20333*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large language models are zero-shot reasoners. *Preprint*, arXiv:2205.11916.

Alex Krizhevsky, Geoffrey Hinton, and 1 others. 2009. Learning multiple layers of features from tiny images.

Jia Li, Ge Li, Chongyang Tao, Jia Li, Huangzhao Zhang, Fang Liu, and Zhi Jin. 2023a. Large language model-aware in-context learning for code generation. *Preprint*, arXiv:2310.09748.

Jia Li, Yunfei Zhao, Yongmin Li, Ge Li, and Zhi Jin. 2023b. Acecoder: Utilizing existing code to enhance code generation. *Preprint*, arXiv:2303.17780.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. 2023. The flan collection: Designing data and methods for effective instruction tuning. *Preprint*, arXiv:2301.13688.

TorchVision maintainers and contributors. 2016. Torchvision: Pytorch's computer vision library. `https://github.com/pytorch/vision`.

Tran Ngoc Minh, Mathieu Sinn, Hoang Thanh Lam, and Martin Wistuba. 2021. Automated image data preprocessing with deep reinforcement learning. *Preprint*, arXiv:1806.05886.

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*.

Alhassan Mumuni and Fuseini Mumuni. 2025a. Automated data processing and feature engineering for deep learning and big data applications: A survey. *Journal of Information and Intelligence*, 3(2):113–153.

Alhassan Mumuni and Fuseini Mumuni. 2025b. Data augmentation with automated machine learning: approaches and performance comparison with classical data augmentation methods. *Knowledge and Information Systems*, 67(5):4035–4085.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *Preprint*, arXiv:2203.02155.

Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Khan, and Arsalan Shahid. 2024. The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities. *Preprint*, arXiv:2408.13296.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, and 2 others. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Guilherme Penedo, Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Edward Beeching, Agustín Piqueres Lajarín, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. 2025. Codeforces cots. https://huggingface.co/datasets/open-r1/codeforces-cots.

Bhavya Rupani, Dmitry Ignatov, and Radu Timofte. 2025. Exploring the collaboration between vision models and llms for enhanced image classification. *Preprints*.

Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2025. A systematic survey of prompt engineering in large language models: Techniques and applications. *Preprint*, arXiv:2402.07927.

Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara, Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, and 12 others. 2025. The prompt report: A systematic survey of prompt engineering techniques. *Preprint*, arXiv:2406.06608.

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*.

Z. Yang, R. O. Sinnott, J. Bailey, and 1 others. 2023. A survey of automated data augmentation algorithms for deep learning-based image classification tasks. *Knowledge and Information Systems*, 65:2805–2861.