Increasing LLM Coding Capabilities through Diverse Synthetic Coding Tasks

Amal Abed[1*] **Ivan Lukic**[1*] **Jörg K.H. Franke**[1,2,3,4] **Frank Hutter**[1,2,5]

[1]University of Freiburg [2]ELLIS Institute Tübingen
[3]Open-Sci Collective [4]LAION [5]Prior Labs

[*]Equal contribution

Abstract

Large language models (LLMs) have shown impressive promise in code generation, yet their progress remains limited by the shortage of large-scale datasets that are both diverse and well-aligned with human reasoning. Most existing resources pair problems with solutions, but omit the intermediate thought process that guides coding. To close this gap, we present a scalable synthetic data generation pipeline that produces nearly 800k *instruction–reasoning–code–test* quadruplets. Each sample combines a task, a step-by-step reasoning trace, a working solution, and executable tests, enabling models to learn not just the *what* but also the *how* of problem solving. Our pipeline combines four key components: curated contest problems, web-mined content filtered by relevance classifiers, data expansion guided by reasoning patterns, and multi-stage execution-based validation. A genetic mutation algorithm further increases task diversity while maintaining consistency between reasoning traces and code implementations. Our key finding is that fine-tuning LLMs on this dataset yields consistent improvements on coding benchmarks. Beyond raw accuracy, reasoning-aware data can substitute for model scaling, generalize across architectures, and outperform leading open-source alternatives under identical sample budgets. Our work establishes reasoning-centered synthetic data generation as an efficient approach for advancing coding capabilities in LLMs. We publish our dataset and generation pipeline to facilitate further research. [1]

1 Introduction

Large language models (LLMs) have shown strong progress in code generation Hui et al. [2024], Jiang et al. [2025], yet their limitations become clear on tasks requiring systematic reasoning and generalization. While benchmarks such as HumanEval [Chen et al., 2021] and MBPP [Austin et al., 2021] highlight the potential of scaling, they also expose a persistent bottleneck: the lack of training data that jointly captures *diversity, reasoning, and functional correctness* at scale. Without such resources, models often succeed on familiar problem types but fail to adapt to new challenges or to explain how solutions are derived.

Most available datasets provide instruction–code pairs or final solutions, but omit the intermediate reasoning that connects problem understanding to executable code. This omission matters because reasoning traces offer a training signal that can improve both reliability and interpretability. Human-annotated resources with reasoning exist but are prohibitively expensive to scale, leaving a gap for

[1] GitHub Repository 🤗 Hugging Face Dataset

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: The 4th Deep Learning for Code Workshop (DL4C).

building models that can both *generate correct programs* and *reveal the logic behind them*. Synthetic datasets have begun to address this, yet many focus narrowly on correctness or rely on costly closed models, limiting both scalability and openness.

To address this gap, we present a reproducible pipeline for constructing synthetic *instruction–reasoning–code–test* datasets designed to advance LLM coding capabilities. Starting from curated seed problems, we expand coverage through corpus filtering, generate structured quadruplets with mid-sized open models, and enforce quality via execution-grounded validation. To further broaden problem coverage, we introduce a genetic instruction mutation algorithm that creates novel task variants without sacrificing correctness.

The resulting dataset captures not only solutions but also the logical traces behind them, offering a scalable resource for training and evaluation. Controlled comparisons against leading open-source datasets show that our formulation consistently delivers stronger transfer performance under identical budgets, underscoring the importance of reasoning-augmented and diversity-driven data generation.

2 Related Work

Recent efforts have focused on scaling dataset construction for code generation. *Self-Code-Align* [Wei et al., 2024a] filters raw functions into structured coding concepts for controlled task creation, while *Magicoder* [Wei et al., 2024b] and *InverseCoder* [Wu et al., 2024] leverage open-source code to produce instruction–code pairs or invert the mapping to generate instructions from code. Other approaches expand task complexity through evolution, as in *WizardCoder* [Luo et al., 2024] and *EpiCoder* [Wang et al., 2025], or through biologically inspired mutation and crossover, as in *Genetic Instruct* [Majumdar et al., 2025].

While these pipelines improve scale and diversity, they rarely capture intermediate reasoning or systematically enforce correctness. Our work extends this line by generating **instruction–reasoning–code–test quadruplets**, validated through execution and guided by classifier-based filtering, yielding datasets that couple semantic diversity with functional reliability.

3 Methodology

We propose an end-to-end methodology for constructing large-scale, reasoning-augmented datasets for code generation. The pipeline is designed to transform heterogeneous programming material into standardized, validated, and diverse problem sets that directly support model training and evaluation. The process begins with broad task collection and expansion, balancing curated contest-style problems with large-scale mining of real-world programming sources. These raw tasks are then systematically structured into instruction–reasoning–solution–test quadruplets, ensuring that each sample exposes both problem statements and associated reasoning traces. A rigorous execution-based validation stage enforces functional correctness, discarding faulty generations while preserving diversity through multi-candidate refinement. To scale beyond limited seed material, we integrate a genetic-inspiration framework that iteratively evolves tasks through crossover and mutation, yielding novel but coherent programming challenges. Finally, a multi-stage deduplication process safeguards against redundancy, while fine-tuning and benchmark evaluations quantify efficiency, generalization, and dataset impact. By unifying curation, structuring, validation, expansion, and deduplication into a single cohesive workflow, our methodology balances scale, reliability, and reasoning fidelity in dataset construction.

3.1 Dataset Curation and Expansion

Our pipeline begins with a seed collection of curated programming tasks in the [LeetCode] style—*title, description, constraints, and examples*—which are particularly useful because they support automatic test generation. However, the publicly available set of only $\sim$2.3k such problems is insufficient for training large language models. To broaden coverage, we incorporated tasks from competitive programming platforms such as [Codeforces] and [AtCoder], combining existing HuggingFace datasets with custom scraping pipelines. These sources introduce greater diversity in problem structure and difficulty, though they often lack reference implementations. Consequently, reasoning traces, candidate code, and executable tests were systematically generated in later stages of our pipeline.

To scale further and move beyond the limited design space of curated contest tasks, we employed a classifier-guided mining approach inspired by Shao et al. [2024]. Specifically, we trained a FastText [Bojanowski et al., 2017] classifier on the curated problems and applied it to the 3B-document DCLM-Baseline corpus [Li et al., 2024], a high-quality subset of Common Crawl[CommonCrawl]. By enforcing a strict 90% relevance threshold, we extracted $\sim$4M candidate documents, striking a balance between high recall of coding-related material and effective filtering of irrelevant or noisy content. This step allowed us to capture a much broader distribution of real-world programming challenges, ranging from algorithmic puzzles to applied coding snippets, thereby providing a richer substrate for the subsequent reasoning-augmented generation stages.The resulting mix of curated and mined material forms the substrate for structured transformation with LLMs.

3.2 Structuring into Instruction–Reasoning–Solution–Test Quadruplets

From the pool of curated and mined documents, we employed **Qwen2.5-Coder-7B-Instruct** [Hui et al., 2024] (Apache 2.0 MIT License) to convert raw programming content into standardized *instruction–reasoning–solution–test* quadruplets. This stage is critical: raw problems from contest archives or web mining are often unstructured, ambiguous, or lack consistent interfaces. The LLM first reformulates each problem into a clear, self-contained instruction, then generates step-by-step reasoning traces that connect problem statements to code implementations, followed by **three candidate solution–test pairs**. This standardized format ensures that downstream models are exposed not only to final answers but also to the intermediate reasoning strategies that support generalization.

We selected Qwen2.5-Coder-7B-Instruct for its balance of efficiency and capability. Its moderate size makes it practical for large-scale generation while still providing strong reasoning and coding abilities. In addition, it is widely available and well-documented, which supports reproducibility and ease of integration into our pipeline. While larger variants such as Qwen2.5-Coder-32B [Hui et al., 2024] can offer higher quality, their computational cost is prohibitive at scale. Importantly, our pipeline remains model-agnostic: different LLMs can be substituted to trade efficiency for further quality improvements without altering the overall process.

3.3 Execution-Based Validation and Refinement

Building on prior execution-validated dataset pipelines such as MAmmoTH2 Yue et al. [2024] and Self-Code-Align Wei et al. [2024a], we incorporated a rigorous multi-candidate refinement stage to ensure that generated samples were both reliable and functionally correct. For each candidate instruction, the LLM was tasked with producing a reasoning trace alongside three alternative *solution–test* pairs. These solutions were executed inside isolated Python containers with strict limits on runtime, memory, and external calls, thereby preventing unsafe or non-terminating code. This multi-candidate approach substantially reduced the risk of discarding otherwise valid problems due to a single poor generation.

The validation process selected the first solution that passed all corresponding test cases, ensuring consistency between reasoning, implementation, and execution. Samples for which no candidate passed were discarded, preventing propagation of faulty code. Beyond correctness, this stage acted as a powerful filter against hallucinated reasoning traces or malformed test cases, refining the dataset into a coherent, executable form.

With this validation in place, the resulting dataset provided a reliable foundation for subsequent scaling and augmentation, which we address next with the Genetic-Instruct framework.

3.4 Evolutionary Expansion with Genetic-Instruct

To scale beyond the initial seed set, we adopted a *Genetic-Instruct* framework [Majumdar et al., 2025], which iteratively evolves new tasks from existing ones (Figure 1). This design is inspired by genetic algorithms: instead of generating problems entirely from scratch, the system reuses validated instructions as a population and applies controlled transformations to increase diversity while preserving internal consistency. Each cycle operates on a pool of high-quality instructions and produces a new generation of candidate tasks that inherit structure and reasoning patterns from their predecessors.

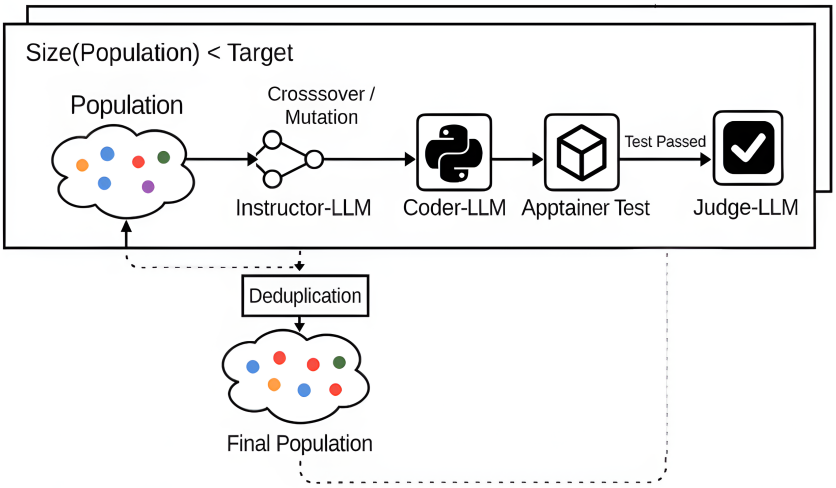

Figure 1: Process Flow for Genetic Instruct

At the core of each cycle, the *Instructor-LLM* generates not only a new instruction but also its accompanying reasoning trace. Two complementary operators guide this process:

- **Crossover:** The LLM is given five seed tasks as few-shot examples and instructed to synthesize exactly one new instruction by merging elements from at least two of them. The resulting hybrid task inherits aspects such as constraints, objectives, or reasoning strategies from multiple parents. Crucially, the Instructor-LLM also produces a coherent reasoning trace that integrates these elements, ensuring the derived problem is both novel and logically solvable.
- **Mutation:** The LLM perturbs an individual task through prompt-driven transformations. Mutations include tightening or adding constraints, increasing reasoning depth, or expanding the problem scope. As with crossover, each mutated task is paired with a fresh reasoning trace that remains consistent with the modified instruction.

The generated *instruction–reasoning* pairs then proceed through the rest of the pipeline. The *Coder-LLM* produces three candidate *solution–test* pairs for each task, guided by the reasoning trace. These are parsed into scripts and executed in secure Apptainer containers under strict resource limits, ensuring isolation, safety, and reproducibility. The first candidate that passes all tests is selected as the final implementation; if none succeed, the instruction is discarded.

Finally, the *Judge-LLM* enforces structural, semantic, and functional quality. In addition to verifying formatting and clarity, it checks that the generated code indeed solves the instruction in alignment with the reasoning trace. Only tasks that pass this judgment are retained. To sustain both quality and diversity, the instruction pool is periodically refreshed: every 200k accepted tasks, the validated set is mixed with the seed pool before entering the next cycle. This iterative feedback loop prevents collapse into repetitive formulations and steadily expands the dataset into a broader space of programming challenges while maintaining correctness and reasoning fidelity.

3.5 Deduplication, Diversity Preservation, and Decontamination

A critical challenge in large-scale dataset construction is avoiding redundancy, since repeated or near-identical problems can bias training and inflate benchmark performance. To mitigate this, we applied a multi-stage deduplication pipeline. First, all instructions were embedded using **MiniLM-L6-v2**[all MiniLM-L6-v2], a lightweight but effective sentence-transformer model. We then performed approximate nearest-neighbor search with **FAISS** [Douze et al., 2024] to efficiently identify candidate duplicates across the hundreds of thousands of structured problems in our dataset. Pairs with cosine similarity above 0.90 were flagged for further inspection. Because surface-level similarity does not always imply true duplication (e.g., two sorting problems with different constraints), flagged pairs were passed to a locally hosted **Gemma-3-27B-IT** [Team et al., 2025] model for verification. This

LLM-based verifier judged whether two instructions were functionally identical, even if phrased differently or containing minor variations. Confirmed duplicates were merged using a *union–find* clustering procedure, which groups all linked pairs into equivalence classes and retains only one representative per class. This approach ensured that the final dataset preserved diversity in problem formulation while eliminating redundancy both in surface phrasing and in underlying functionality.

In addition to deduplication, we also conducted a thorough data leakage check against common code evaluation benchmarks, namely **HumanEval** and **MBPP**. To this end, we computed and compared hashes of benchmark problems with those in our dataset. The comparison revealed **zero overlap**, confirming that our dataset does not contain leaked benchmark problems and is suitable for reliable downstream fine-tuning and evaluation.

3.6 Fine-Tuning Setup

To rigorously measure the impact of our dataset, we fine-tuned **Phi-2**, a 2.7B-parameter transformer developed by Microsoft [Gunasekar et al., 2023] (MIT License). Phi-2 was selected because it strikes a favorable balance between scale and capability: despite its relatively modest size compared to recent large language models, it demonstrates strong reasoning and code generation ability. This makes it a cost-effective and informative testbed, allowing us to isolate the contributions of our dataset without confounding factors introduced by extremely large model architectures.

Fine-tuning was carried out using the QLoRA [Dettmers et al., 2023] framework, which enables efficient adaptation of large models by applying low-rank updates to a subset of parameters while keeping the majority of the model frozen. Specifically, we set the rank to $r = 16$, the scaling factor to $\alpha = 16$, and targeted the modules `[q_proj, v_proj, k_proj, dense]`. Training was conducted for 10 epochs, providing sufficient exposure to the dataset for the model to adapt to the reasoning-augmented patterns while remaining computationally feasible.

3.6.1 Evaluation

We assessed the fine-tuned models on two widely used benchmarks: *HumanEval*, which emphasizes algorithmic reasoning and problem-solving, and *MBPP*, which contains shorter programming tasks with more direct mappings from instructions to solutions. To ensure consistency and rigor in evaluation, we adopted the `EvalPlus` framework [Liu et al., 2023], which extends the original benchmark test suites with additional cases and more robust correctness checks. This offers a stronger measure of generalization beyond the limited canonical test sets.

4 Results

4.1 HumanEval and MBPP Benchmarks

The base `phi-2 2.7B` achieved 45.7% (Base) and 40.9% (Extra) on HumanEval, and 62.7% / 51.6% on MBPP. LeetCode [greengerong, 2023] fine-tuning offered only marginal improvements, underscoring the limitations of small, curated datasets. In contrast, our synthetic data consistently boosted performance, with gains that scaled with dataset size. At 25k synthetic samples, pass rates reached 56.1% / 51.8% on HumanEval and 65.6% / 55.3% on MBPP—representing nearly +10 absolute points over baseline on HumanEval.

Overall, these experiments confirm that synthetic, reasoning-augmented data offers a scalable and effective path toward improving coding performance across diverse benchmarks.

4.2 Efficiency Gains

Synthetic, reasoning-augmented fine-tuning proved to be a more efficient alternative to scaling model size. The fine-tuned `phi-2 2.7B` achieved competitive, and in some cases superior, performance compared to substantially larger models such as `CodeLlama-70B` [Roziere et al., 2023], `Llama3-8B-instruct` [Dubey et al., 2024], and `DeepSeek-Coder-33B-base` [Guo et al., 2024] (Figure 2). This demonstrates that targeted synthetic data can significantly narrow the performance gap between small and large models, offering a more compute-efficient path to progress.

Table 1: Pass rates (%) on HumanEval and MBPP for `phi-2 2.7B`

Model	**HumanEval**		**MBPP**	
	Base Test (%)	Extra Tests (%)	Base Test (%)	Extra Tests (%)
Base Model (Phi-2 2.7B)	45.7	40.9	62.7	51.6
Fine-tuned on LeetCode dataset	47.6	42.1	63.0	51.6
Fine-tuned on 5k synthetic samples	54.3	49.4	64.3	54.5
Fine-tuned on 10k synthetic samples	54.9	50.6	65.6	55.3
Fine-tuned on 25k synthetic samples	**56.1**	**51.8**	**65.6**	**55.3**

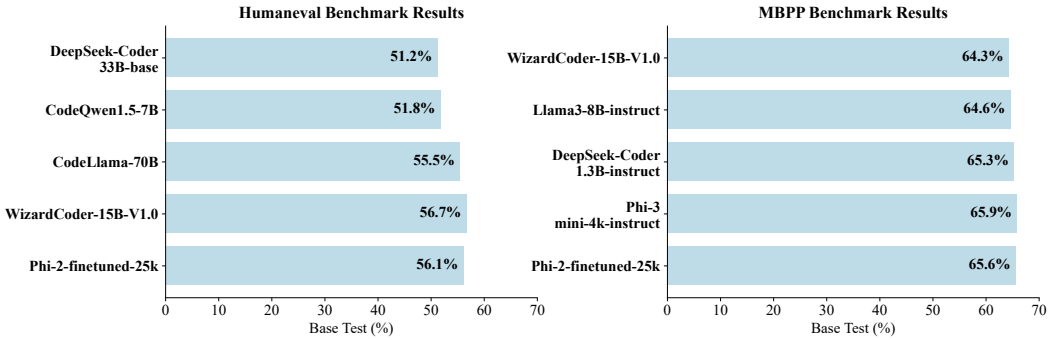

Figure 2: Pass rates on HumanEval and MBPP benchmarks for selected models.

The implications extend beyond raw benchmark numbers. Scaling to tens of billions of parameters requires specialized hardware, large-scale distributed training, and high inference costs, all of which limit accessibility. In contrast, our approach shows that carefully curated, reasoning-focused data allows models an order of magnitude smaller to deliver comparable gains. This not only reduces training and deployment costs, but also broadens access to competitive code generation systems for researchers and practitioners with modest resources.

In this context, efficiency means achieving strong performance without the prohibitive expenses of brute-force scaling. Synthetic, reasoning-augmented fine-tuning therefore emerges as a practical and sustainable alternative, advancing the capabilities of code generation models while keeping training and inference requirements manageable.

5 Experiments

Our experiments test whether targeted, reasoning-augmented fine-tuning can improve code generation without simply scaling model size. We evaluate along two complementary axes: (1) **generalization**, by assessing whether improvements transfer across architectures such as `CodeGemma-2B`; and (2) **dataset quality**, by contrasting homogeneous versus diverse subsets of our data and benchmarking against other recent open-source resources. Together, these studies isolate the contributions of scale, architecture, and dataset design.

To ensure fair comparisons, we standardized fine-tuning using QLoRA. A one-epoch configuration search was conducted over eight variants, varying rank ($r \in \{16, 32\}$), scaling ($\alpha = 2r$), target modules (`[q_proj, v_proj]` or `[q_proj, v_proj, k_proj]`), and whether to retain the final classification head (`save_head` $\in \{$`True`, `False`$\}$). The best-performing configuration on downstream benchmarks was then applied consistently across all models and datasets.

Compute resources. All fine-tuning experiments were run on a single NVIDIA A100 GPU (80GB) using QLoRA. Each configuration search run completed within $\sim$1–2 hours, while fine-tuning for 10 epochs with the selected hyperparameters required up to $\sim$12 hours for the largest dataset (25k examples). We report results from single-GPU runs to ensure reproducibility and accessibility, avoiding reliance on large-scale distributed infrastructure.

Table 2: Pass rates (%) on HumanEval and MBPP for `CodeGemma-2B`

Model	**HumanEval**		**MBPP**	
	Base Test (%)	Extra Tests (%)	Base Test (%)	Extra Tests (%)
Base Model (CodeGemma-2B)	23.2	17.7	55.6	45.8
Fine-tuned on LeetCode	31.1	25.0	57.7	48.4
Fine-tuned on 5k synthetic	33.5	27.4	60.6	50.0
Fine-tuned on 10k synthetic	36.6	31.1	**62.4**	**52.6**
Fine-tuned on 25k synthetic	**37.8**	**32.3**	**62.4**	51.6

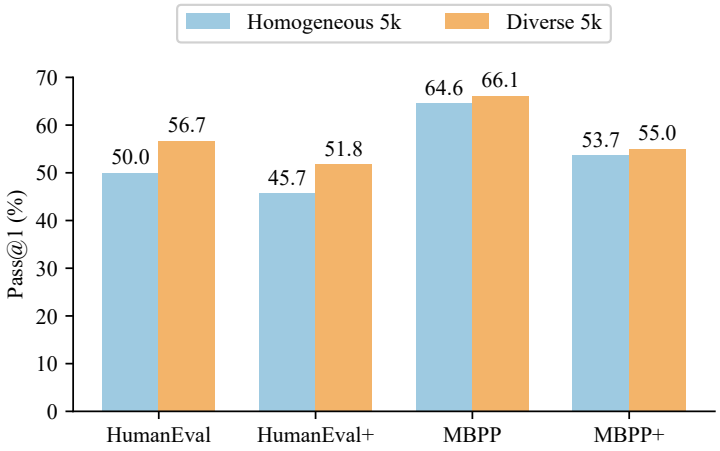

Figure 3: Pass@1 scores on HumanEval and MBPP for homogeneous vs. diverse subsets. Diversity yields stronger reasoning alignment.

5.1 Cross-Model Generalization

To evaluate whether these gains are architecture-specific, we applied the same training subsets to `CodeGemma-2B` [Team et al., 2024](Gemma License), a model already specialized for coding. For fair comparison, we re-fine-tuned `phi-2` under identical conditions (same subsets, training setup, and evaluation protocol) to those used for `CodeGemma`. As shown in Table 2, CodeGemma benefited strongly from our dataset: fine-tuning on 25k samples yielded a +14.6 point increase on HumanEval base tests (23.2% $\rightarrow$ 37.8%) and a +6.8 point gain on MBPP base tests (55.6% $\rightarrow$ 62.4%), with comparable improvements on extra tests. Interestingly, the scaling behavior was consistent with that of `phi-2`, suggesting that our dataset delivers distinct benefits even for models already optimized for code.

Taken together, these findings highlight two important aspects: (1) reasoning-augmented synthetic data enables efficient specialization of smaller models, reducing reliance on costly scaling, and (2) the benefits generalize across architectures and pretraining regimes. Both points strengthen the case for dataset-driven approaches as a practical alternative to simply increasing model size. Having established that our data scales efficiently across models, we next ask whether *the structure of the dataset itself*—in particular, its diversity versus homogeneity—affects downstream performance.

5.2 Diversity vs. Homogeneity

To disentangle the effects of scale and coverage, we compared matched 5k subsets under two conditions: *homogeneous* and *diverse*. The homogeneous subsets were sampled from our dataset *before the deduplication step*, resulting in highly similar, single-domain style problems. In contrast, the diverse subsets were drawn from the final deduplicated dataset, covering a broad range of domains and reasoning types.

Table 3: Evaluation results (%) across HellaSwag, WinoGrande, and MMLU for base, 5k-fine-tuned, and 25k-fine-tuned models

Dataset / Metric	Base Model	5K Fine-tuned	25K Fine-tuned
HellaSwag (Acc)	55.88	55.22	55.50
HellaSwag (Acc_Norm)	73.76	73.00	73.32
WinoGrande (Acc)	75.93	76.09	76.01
MMLU (Avg Acc)	54.37	53.94	54.14

Despite being identical in size, the diverse subsets consistently outperformed the homogeneous ones. On HumanEval, diverse fine-tuning reached 56.7% compared to 50.0% for homogeneous, and 51.8% versus 45.7% on HumanEval+. On MBPP, the effect was smaller but still present: 66.1% vs. 64.6%, and 55.0% vs. 53.7% on MBPP+. These results highlight that redundancy and narrow domain focus reduce the effective information content of training data, while exposure to a variety of problem formulations improves downstream generalization.

Taken together, these findings show that at small and medium training budgets, *diversity is more valuable than raw sample count*. Even relatively small but heterogeneous datasets can rival or exceed larger, less varied ones, underscoring the importance of deduplication and domain coverage in building effective resources for code generation. While these results highlight benefits within code-focused benchmarks, it remains essential to confirm that domain-specific fine-tuning preserves broader reasoning ability. We turn to this in the next section.

5.3 General Reasoning Benchmarks

A common concern with domain-specific fine-tuning is that it might reduce a model's broader reasoning ability. To verify this, we evaluated our models on three standard benchmarks outside programming: HellaSwag (commonsense inference) [Zellers et al., 2019], WinoGrande (coreference reasoning) [Sakaguchi et al., 2021], and MMLU (multi-task knowledge)[Hendrycks et al., 2020]. Across all three, accuracy remained essentially unchanged, with variations well within normal fluctuation (see Table 3).

These results indicate that reasoning-aware code fine-tuning preserves general capabilities. In other words, models can gain substantial improvements in code generation without sacrificing the versatility needed for non-coding tasks, making this approach suitable for building specialized yet broadly capable systems. Finally, to contextualize our approach against existing resources, we compare our dataset directly with other recent open-source alternatives.

5.4 Comparison with Other Datasets

To put our dataset in context, we benchmarked it against two recent open-source resources: *EpiCoder-func-380k* [Wang et al., 2025](MIT License) and *Self-OSS-Instruct-SC2-Exec-Filter-50k* [Wei et al., 2024a]. For fairness, we fine-tuned on matched 5k subsets from each source under identical training settings. Across both HumanEval and MBPP, models trained on our data consistently achieved higher pass rates, even though all experiments used the same sample budget (Figure 4).

The comparisons also highlight where our approach makes the biggest difference. Improvements were most pronounced on HumanEval, which contains more complex programming tasks that often require multi-step reasoning. This suggests that our dataset's emphasis on reasoning-oriented problem formulations, genetic instruction variation, and rigorous validation is particularly valuable for benchmarks that go beyond surface-level coding skills. While MBPP also showed gains, they were smaller, consistent with its focus on simpler problems. Together, these results indicate that careful dataset design can yield benefits beyond scale alone, especially when the downstream tasks demand structured reasoning.

6 Conclusion

This work demonstrates that large-scale, reasoning-augmented synthetic datasets can play a decisive role in advancing the coding capabilities of large language models. By combining FastText-

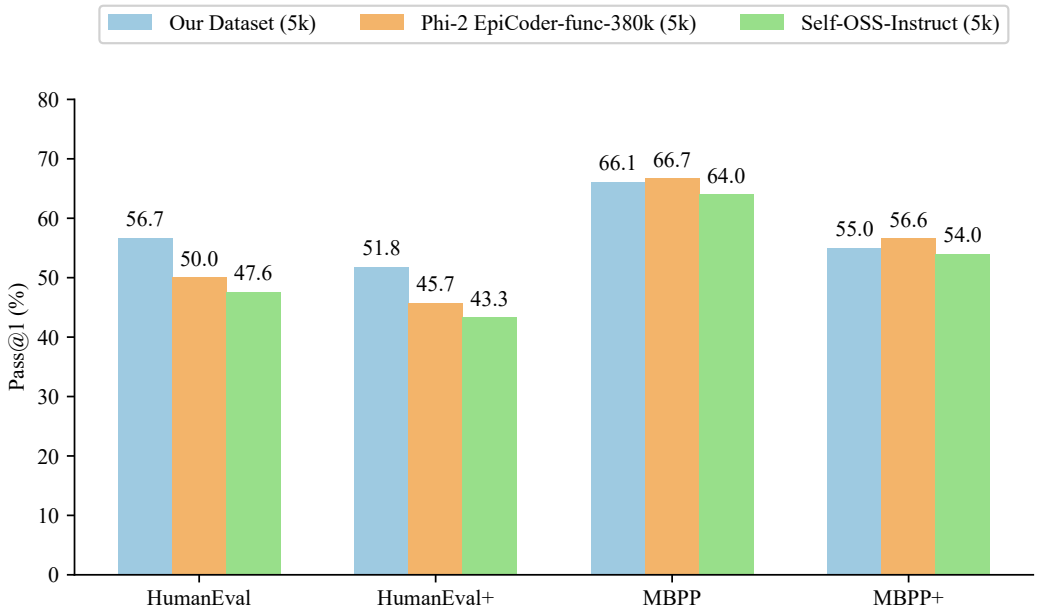

Figure 4: Comparison of pass@1 scores on HumanEval and MBPP for models fine-tuned on 5k samples from our dataset versus EpiCoder-func-380k and Self-OSS-Instruct-SC2-Exec-Filter-50k.

based corpus filtering, Qwen2.5-Coder generation, multi-stage validation, and our *Genetic Instruct* algorithm for systematic instruction variation, we produced nearly **800k high-quality instruction–reasoning–code–test quadruplets**. Unlike existing resources, our dataset captures not just solutions, but also the intermediate reasoning processes that enable models to generalize to harder problems. The resulting data proved to be diverse, interpretable, and reliable—qualities that our experiments identified as critical drivers of performance.

Extensive evaluations highlight the strength of this approach. Fine-tuning on our synthetic subsets consistently surpassed strong baselines such as LeetCode, and even allowed smaller models to rival or outperform substantially larger ones. These results underline that the key to progress is not raw dataset size, but the integration of reasoning and diversity. Further, cross-model experiments confirmed that the benefits generalize across architectures, while evaluations on general reasoning tasks showed no degradation—demonstrating the robustness of our pipeline.

A current limitation of our work is that the pipeline supports only Python. Extending it to other programming languages will require adapting the execution-based validation framework, which is essential to ensure functional correctness in more diverse coding environments.

Taken together, our findings establish synthetic, diversity-driven data generation as a powerful and scalable foundation for building more capable and efficient code-focused LLMs. Looking forward, extending this pipeline to multilingual programming languages (e.g., Java, C++, JavaScript) and deploying it at pretraining scale offers a clear pathway toward models that can reason, generalize, and adapt across programming paradigms. In a landscape where real-world datasets are limited, our results highlight that it is the *breadth and structure of data*—not raw scale—that unlocks the next generation of robust, reasoning-capable models for code.

Acknowledgements

This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 417962828. We acknowledge funding by the European Union (via ERC Consolidator Grant DeepLearning 2.0, grant no. 101045765). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Funded by the European Union

We gratefully acknowledge the Gauss Centre for Supercomputing e.V. for funding this work by providing computing time through the John von Neumann Institute for Computing (NIC) on the supercomputer JUWELS Booster at Jülich Supercomputing Centre (JSC).

References

all MiniLM-L6-v2. all-minilm-l6-v2. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2. Accessed: 2025-10-15.

AtCoder. Atcoder. https://atcoder.jp/. Accessed: 2025-10-15.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732. arXiv preprint arXiv:2108.07732.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Codeforces. Codeforces. https://codeforces.com/. Accessed: 2025-10-15.

CommonCrawl. Commoncrawl. https://commoncrawl.org/. Accessed: 2025-10-15.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 10088–10115. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1feb87871436031bdc0f2beaa62a049b-Paper-Conference.pdf.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407, 2024.

greengerong. Leetcode dataset, 2023. URL https://huggingface.co/datasets/greengerong/leetcode. Accessed: 2025-08-21.

Suriya Gunasekar, Xuezhi Chen, Tris Dube, Yichong Ge, Shrimai Ma, Yi Tay, and Barret Zoph. Phi-2: A small language model with a big knowledge of natural language. https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/, 2023.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Xiaodong Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *ArXiv*, abs/2009.03300, 2020. URL `https://api.semanticscholar.org/CorpusID:221516475`.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-coder technical report, 2024.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. *ACM Trans. Softw. Eng. Methodol.*, July 2025. ISSN 1049-331X. doi: 10.1145/3747588. Just Accepted.

LeetCode. Leetcode. `https://leetcode.com/`. Accessed: 2025-10-15.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, Hritik Bansal, Etash Guha, Sedrick Scott Keh, Kushal Arora, et al. Datacomp-lm: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 37:14200–14282, 2024.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and LINGMING ZHANG. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=1qvx610Cu7`.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=UnUwSIgK5W`.

Somshubra Majumdar, Vahid Noroozi, Mehrzad Samadi, Sean Narenthiran, Aleksander Ficek, Wasi Uddin Ahmad, Jocelyn Huang, Jagadeesh Balam, and Boris Ginsburg. Genetic instruct: Scaling up synthetic generation of coding instructions for large language models. In Georg Rehm and Yunyao Li, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 208–221, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-288-6. doi: 10.18653/v1/2025.acl-industry.16. URL `https://aclanthology.org/2025.acl-industry.16/`.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: an adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106, August 2021. ISSN 0001-0782. doi: 10.1145/3474381. URL `https://doi.org/10.1145/3474381`.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

CodeGemma Team, Heri Zhao, Jeffrey Hui, Joshua Howland, Nam Nguyen, Siqi Zuo, Andrea Hu, Christopher A Choquette-Choo, Jingyue Shen, Joe Kelley, et al. Codegemma: Open code models based on gemma. *arXiv preprint arXiv:2406.11409*, 2024.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Yaoxiang Wang, Haoling Li, Xin Zhang, Jie Wu, Xiao Liu, Wenxiang Hu, Zhongxin Guo, Yangyu Huang, Ying Xin, Yujiu Yang, Jinsong Su, Qi Chen, and Scarlett Li. Epicoder: Encompassing diversity and complexity in code generation. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=RAxe7nF4Oz.

Yuxiang Wei, Federico Cassano, Jiawei Liu, Yifeng Ding, Naman Jain, Zachary Mueller, Harm de Vries, Leandro Von Werra, Arjun Guha, and LINGMING ZHANG. Selfcodealign: Self-alignment for code generation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a. URL https://openreview.net/forum?id=xXRnUU7xTL.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with OSS-instruct. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 52632–52657. PMLR, 21–27 Jul 2024b. URL https://proceedings.mlr.press/v235/wei24h.html.

Yutong Wu, Di Huang, Wenxuan Shi, Wei Wang, Lingzhe Gao, Shihao Liu, Ziyuan Nan, Kaizhao Yuan, Rui Zhang, Xishan Zhang, Zidong Du, Qi Guo, Yewen Pu, Dawei Yin, Xing Hu, and Yunji Chen. Inversecoder: Unleashing the power of instruction-tuned code llms with inverse-instruct, 2024. URL https://arxiv.org/abs/2407.05700.

Xiang Yue, Tianyu Zheng, Ge Zhang, and Wenhu Chen. Mammoth2: Scaling instructions from the web. *Advances in Neural Information Processing Systems*, 37:90629–90660, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Annual Meeting of the Association for Computational Linguistics*, 2019. URL https://api.semanticscholar.org/CorpusID:159041722.