Rewriting Pre-Training Data Boosts LLM Performance in Math and Code

Kazuki Fujii[1,2], Yukito Tajima[1], Sakae Mizuki[1,2], Hinari Shimada[1], Taihei Shiotani[1], Koshiro Saito[1], Masanari Ohi[1], Masaki Kawamura[1], Taishi Nakamura[1,2], Takumi Okamoto[1], Shigeki Ishida[1], Kakeru Hattori[1,2], Youmi Ma[1], Hiroya Takamura[2], Rio Yokota[2,3], and Naoaki Okazaki[1,2]

[1]Institute of Science Tokyo, Department of Computer Science
[2]National Institute of Advanced Industrial Science and Technology
[3]Institute of Science Tokyo, Institute of Integrated Research, Supercomputing Research Center

https://huggingface.co/datasets/tokyotech-llm/swallow-code
https://huggingface.co/datasets/tokyotech-llm/swallow-math

Abstract

The performance of large language models (LLMs) in program synthesis and mathematical reasoning is fundamentally limited by the quality of their pre-training corpora. We introduce two openly licensed datasets, released under the Llama 3.3 Community License, that significantly enhance LLM performance by systematically rewriting public data. SwallowCode (≈16.1 billion tokens) refines Python snippets from The-Stack-v2 through a novel four-stage pipeline: syntax validation, pylint-based style filtering, and a two-stage LLM rewriting process that enforces style conformity and transforms snippets into self-contained, algorithmically efficient examples. Unlike prior methods that rely on exclusionary filtering or limited transformations, our transform-and-retain approach upgrades low-quality code, maximizing data utility. SwallowMath (≈2.3 billion tokens) enhances Finemath-4+ by removing boilerplate, restoring context, and reformatting solutions into concise, step-by-step explanations. Within a fixed 50 billion token training budget, continual pre-training of Llama-3.1-8B with SwallowCode boosts pass@1 by **+17.0** on HumanEval and **+17.7** on HumanEval+ compared to Stack-Edu, surpassing the baseline model's code generation capabilities. Similarly, substituting SwallowMath yields **+12.4** accuracy on GSM8K and **+7.6** on MATH. Ablation studies confirm that each pipeline stage contributes incrementally, with rewriting delivering the largest gains. All datasets, prompts, and checkpoints are publicly available, enabling reproducible research and advancing LLM pre-training for specialized domains.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable zero-shot and few-shot capabilities across diverse tasks, yet their proficiency in mathematical reasoning and program synthesis remains constrained by the quality of pre-training corpora. Existing public datasets for specialized domains, such as The-Stack-v1 and v2 for code [1, 2] and Finemath-4+ for mathematics [3], rely primarily on rule-based extraction from web crawls (e.g., CommonCrawl) [4] or model-based scoring to filter low-quality samples. However, these approaches often retain noisy, redundant, or stylistically inconsistent data, limiting their effectiveness–particularly in the growing trend of multi-stage pretraining (or mid-training) aimed at enhancing mathematical reasoning and program synthesis (e.g., OLMo 2 [5], Nemotron-H [6], and Phi-4 [7]). For instance, as shown in Figure 1, continual pre-training of Llama-3.1-8B [8] on The-Stack-v1/v2 typically maintains baseline performance on benchmarks like HumanEval and HumanEval+ but struggles to achieve significant gains due to unaddressed data quality issues. Unlike prior methods that focus on filtering or preserving original samples, we

Preprint. Under review.

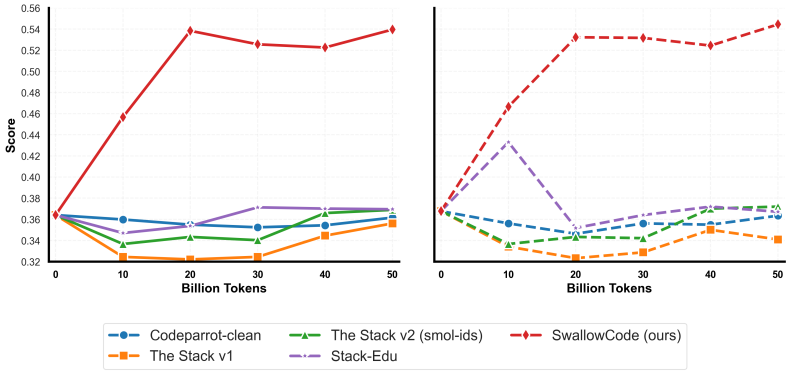

Figure 1: Comparison of Python-only datasets in 50 billion tokens continual pre-training setting. SwallowCode achieves the highest pass@1 on HumanEval(left) and HumanEval+(right) compared to CodeParrot-Clean, The-Stack-v1, The-Stack-v2-Smol, and Stack-Edu.

propose rewriting pre-training corpora to eliminate noise and redundancy, delivering high-quality, self-contained data that enables efficient model learning.

This paper introduces two openly licensed datasets under the Llama 3.3 Community License, designed to advance LLM capabilities in code generation and mathematical reasoning. SwallowCode (≈16.1 billion tokens) refines Python snippets from The-Stack-v2 through a novel four-stage pipeline: sequential syntax validation, pylint-based style filtering, and a two-stage LLM rewriting process that enforces style conformity and transforms snippets into algorithmically efficient, self-contained examples. By rewriting, rather than merely filtering, we eliminate noise and redundancy that persist in datasets like Stack-Edu, providing high-quality data that drives rapid accuracy improvements (Figure 1). SwallowMath (≈2.3 billion tokens) transforms Finemath-4+ by removing boilerplate, restoring missing context, and reformatting solutions into concise and step-by-step explanations. Although our experiments focus on Python for ease of automated evaluation, the pipeline is language-agnostic: any programming language with a parseable syntax and a linter can benefit from the same treatment, broadening its applicability to diverse coding domains.

Continual pre-training of Llama-3.1-8B [8] for 50 billion tokens on mixed dataset with SwallowCode and multilingual corpora improves pass@1 by **+17.0** on HumanEval and **+17.7** on HumanEval+ compared to an equivalent budget using Stack-Edu [3]. Similarly, substituting Finemath-4+ with SwallowMath yields **+12.4** accuracy on GSM8K and **+7.6** on MATH. To ensure the robustness of these gains, we conducted rigorous checks for test-set leakage, streaming the entire 16.1B-token SwallowCode corpus and finding no exact matches or high-similarity documents (Jaccard similarity ≥ 0.8) with HumanEval or HumanEval+ prompts. All datasets, prompts, and checkpoints are publicly released, enabling reproducible research. SwallowCode and SwallowMath provide a scalable framework for enhancing LLM pre-training, driving advancements in automated reasoning and software development.

2 Related work

2.1 Open code corpora

The Stack v1 [1] aggregates approximately 3TB of permissively licensed source code from public GitHub repositories. Although deduplication and license filtering are applied, boilerplate and autogenerated files are retained without language-specific refinement. The Stack v2 [2] extends this pipeline by sourcing data from the Software Heritage archive, relaxing licensing constraints, and introducing language-specific filters to enhance data quality. With a scale of approximately 900 billion tokens, it represents a significant increase in volume over v1. However, like its predecessor, v2 adopts a filtering-only approach, preserving files without semantic enhancements, which leaves inconsistent coding styles and fragmented scripts unaddressed.

2.2 Classifier-based filtering for code corpora

Recent work has shown that classifier-based filtering strategies, such as the FineWeb-Edu approach [9], can be effective for curating high-quality datasets [10]. These methods aim to improve model performance on code-related tasks by selecting semantically rich and well-documented samples from large-scale corpora. In this context, Stack-Edu [3] represents a significant effort to create a filtered variant of StarCoder2Data [2] prioritizing high-quality code. Stack-Edu begins by selecting the 15 most prevalent programming languages from StarCoder2Data, forming a subset of approximately 450 billion tokens. To assess code quality, Stack-Edu leverages Llama-3-70B-Instruct [8] to generate synthetic annotations for 500,000 code fragments, rating each on a 0–5 scale based on educational and structural quality. These annotations train language-specific classifiers based on the StarEncoder model [11], achieving F1 scores above 0.7 for most languages when applying a threshold of 3 for binary classification. The resulting dataset comprises 125 billion tokens and demonstrates improved model convergence and higher pass@1 scores on HumanEval compared to unfiltered corpora [3]. However, Stack-Edu's exclusionary filtering strategy discards low-scoring snippets rather than rewriting or augmenting them, leaving residual issues—such as missing context or inconsistent naming conventions in the retained data.

2.3 LLM-driven pre-training corpus rewriting

Efforts to enhance code datasets using LLMs have gained traction. Austin et al. [12] explored LLM capabilities for program synthesis, finding that human feedback significantly improves code accuracy, but their work focused on synthesis rather than dataset rewriting. Cosmopedia [13] demonstrated the potential of synthetic data generation, using Mixtral-8×7B-Instruct [14] to create high-quality text corpora, though it did not address code-specific challenges.

Jain et al. [15] applied LLM-driven code transformations to instruction-tuning exemplars, focusing on variable renaming, modularization, and comment addition. While effective for fine-tuning, their approach is limited in scope and context compared to our SwallowCode pipeline. Their transformations address only a subset of stylistic improvements, whereas our Style-Guided Code Rewriting (SGCR) pipeline comprehensively enforces ten criteria from the Google Python Style Guide, including descriptive variable naming, effective type annotations, modular function design, error handling, and readability-focused formatting (see Section 3.3.1). Furthermore, our Self-Contained Optimization Rewriting (SCOR) pipeline introduces semantic enhancements—ensuring self-containment by resolving dependencies, optimizing algorithms, and transforming trivial snippets into instructive examples—which are absent in Jain et al.'s work (see Section 3.3.2). Additionally, Jain et al. target instruction-tuning data, a smaller and more curated dataset, whereas SwallowCode systematically rewrites large-scale pre-training corpora, a more challenging task due to the diversity and volume of code samples. Our pipeline integrates rigorous preprocessing with syntax error and linter-based filtering to ensure high-quality inputs for rewriting.

SwallowCode's transform-and-retain paradigm upgrades low-quality code rather than discarding it, addressing limitations of existing datasets like The Stack v1 [1] and v2 [2], and Stack-Edu [3], which rely on filtering. By combining filtering, SGCR, and SCOR, SwallowCode produces a high-quality corpus that significantly improves performance on HumanEval and HumanEval+ benchmarks, advancing the state-of-the-art in code dataset curation.

2.4 Synthetic Data Generation for Code

Recent approaches, such as Magpie [16], leverage high-performing LLMs to generate synthetic instruction-tuning datasets from prompts and desired characteristics without relying on existing data. However, we opted against generating synthetic code data from scratch for SwallowCode due to two key limitations. First, previous work [17] demonstrates that low-diversity synthetic data restricts LLM performance. Second, achieving high diversity in synthetic code datasets requires diverse topics and keywords as seeds, as seen in Nemotron-4 340B's synthetic instruction data [18]. For code, defining such seeds (e.g., varied algorithmic paradigms or problem domains) remains an open challenge, as no established methodology ensures sufficient diversity across programming tasks. Instead, our approach leverages high-quality code from The-Stack-v2, filtered for syntactic and stylistic rigor (Section 3.2), and applies LLM-driven rewriting to enhance quality while preserving the inherent

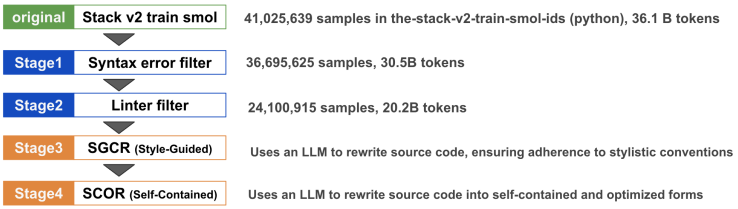

Figure 2: Four-stage pipeline for constructing SwallowCode: (1) syntax filtering to remove invalid Python code, (2) linter-based filtering using pylint to enforce coding standards, and (3–4) two-stage LLM rewriting with Style-Guided Code Rewriting (SGCR), which enforces consistent style and readability, and Self-Contained Optimization Rewriting (SCOR), which ensures self-containment and optimizes algorithms for efficiency.

diversity of real-world code. This transform-and-retain strategy maximizes data utility and avoids the risks of synthetic data homogeneity.

3 Construction of the code corpus

The development of SwallowCode is driven by an empirical and exploratory approach, informed by data ablation experiments evaluating each stage of the data processing pipeline, as illustrated in Figure 2. Specifically, we continually pre-trained the baseline model with the dataset only differing code text subset: before and after applying specific stage in the pipeline, and decide if we should adopt or discard the stage based on the evaluation results. In this section, we present the experimental results and describe the design choices shaping the pipeline. Please refer to Appendix H.1 for detailed results. To ensure full reproducibility, all code, configurations, and supporting materials are publicly available at `https://github.com/rioyokotalab/swallow-code-math`.

3.1 Experimental setup

To evaluate the impact of each design choice in our data processing pipeline, we conduct systematic data ablation studies. Each ablation trains a model that differs only in the target pre-training dataset, holding all other factors constant, including model architecture, parameter count, non-target data, total token budget, and hyperparameters. Specifically, we perform continual pre-training starting from Llama-3.1-8B, using a total of approximately 50 billion tokens, a maximum sequence length of 8,192, a global batch size of approximately 4 million tokens, and the Llama-3 tokenizer. The target dataset is processed with less than one epoch within each ablation. We evaluate model checkpoints approximately every 10 billion tokens using ten downstream benchmarks.

Continual pre-training is conducted using Megatron-LM (version core_r0.9.0) [19]. For evaluation, we utilize the BigCodeBench [20] and lm-evaluation-harness [21] on the following benchmarks: OpenBookQA [22], TriviaQA [23], HellaSwag [24], SQuAD 2.0 [25], XWINO [26], MMLU [27], GSM8K [28], BBH [29], HumanEval [30], and HumanEval+ [31]. The effectiveness of the code corpus is specifically assessed using HumanEval and HumanEval+. The pre-training data mixture consists of 84% multilingual text and 16% code. Detailed proportions and data source are provided in Appendix A.4.1, with comprehensive training hyperparameters reported in Appendix A.1. All ablation models, associated checkpoints, and supporting materials are publicly available in our Hugging Face repository to ensure full reproducibility.

3.2 Filtering

To construct the SwallowCode corpus, a high-quality Python code dataset, we implement a rigorous filtering pipeline starting with the-stack-v2-train-smol-ids [2] as the base dataset. The filtering process is critical to ensure that only syntactically correct and well-structured code is retained, enhancing downstream performance in code generation tasks. We focus exclusively on Python code to maintain consistency across ablation studies and enable fair comparisons with existing public corpora. Our pipeline employs two key filtering techniques, syntax error filtering and linter-based filtering, that significantly improve code quality. We also evaluated LLM-based scoring in ablation experiments but did not adopt it in the final pipeline due to its limited performance gains relative to computational

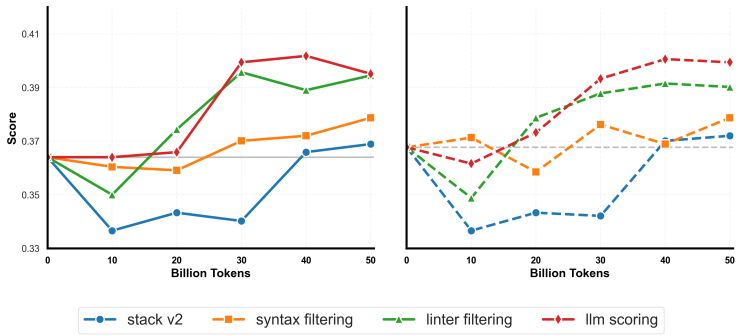

Figure 3: Performance comparison of filtering methods on HumanEval (left) and HumanEval+ (right) benchmarks. The original dataset (Stack v2) is compared against datasets processed with syntax error filtering, linter-based filtering, and LLM-based scoring (evaluated in ablation studies but not adopted). Syntax and linter-based filtering enhance code generation performance, while LLM-based scoring provides marginal gains, considering the computational cost.

cost. Figure 3 summarizes the performance of these methods on the HumanEval and HumanEval+ benchmarks, demonstrating the impact of our filtering strategy.

3.2.1 Syntax error filtering

Despite the heuristic filtering in the BigCode project, the-stack-v2-train-smol-ids [2] includes Python code samples with invalid syntax according to Python 3.10 specifications. To address this, we apply syntax error filtering by compiling each code sample using Python's built-in `compile()` function, discarding any samples that fail to compile. This process reduces the dataset from approximately 41 million to 37 million samples, a 9.7% reduction. As shown in Figure 3, removing syntactically invalid samples improves performance on HumanEval and HumanEval+ [1]. Consequently, we adopt syntax-error filtering as a standard step in all subsequent experiments.

3.2.2 Linter-based filtering

Beyond syntactic correctness, code quality depends on adherence to coding standards. Many samples in the initial dataset exhibit poor structure, generating numerous warnings when analyzed by static analysis tools. We employ Pylint, a widely-used Python linter, to enforce a quality threshold, excluding samples with scores below 7.0 on a 0–10 scale based on rule violations. Additionally, we penalize overly verbose comments using a custom heuristic scoring algorithm (detailed in Appendix B). This step reduces the dataset from 36.7 million to 24.1 million samples, a 34.3% reduction. Figure 3 illustrates the performance gains of more than 1 point in HumanEval and HumanEval + achieved by linter-based filtering. Consequently, we adopt linter-based filtering, with a threshold of 7.0, as a standard step in all subsequent experiments.

3.2.3 LLM-based scoring filtering

Recent approaches leverage large language models (LLMs) to generate synthetic annotations for training quality classifiers, which are then used to filter web-scale corpora by retaining high-quality samples (e.g., FineWeb [9], Stack-Edu [3]). Instead of training a separate classifier, we directly prompt Llama-3.3-70B-Instruct [8] to evaluate each Python code snippet on a scale of 0–10, based on ten criteria, including code readability, modularity, and adherence to naming conventions, derived from the Google Python Style Guide. The detailed scoring prompt and the distribution of the quality scores are provided in Appendix C.

We exclude samples scoring below 6, retaining only those deemed sufficiently high-quality, and use this filtered subset alongside multilingual data for continual pre-training in our ablation studies.

[1]The performance trajectory of The Stack v2 exhibits an initial decline followed by a recovery, consistent with forgetting and subsequent adaptation observed in prior continual pretraining studies [32]. This behavior is not particularly noteworthy.

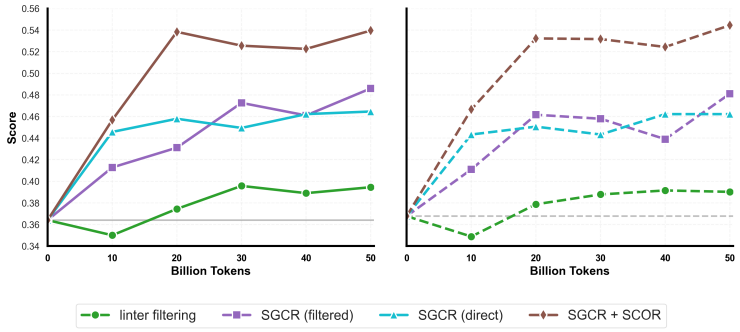

Figure 4: Performance comparison of LLM-driven rewriting steps on HumanEval (left) and HumanEval+ (right) benchmarks. The pre-rewriting (syntax-error and linter-based filtering) is compared against SGCR and SCOR. SGCR improves performance by over 9 points, while SCOR, applied after SGCR, further enhances scores by over 5 points, demonstrating the effectiveness of stylistic and semantic rewrites in the SwallowCode pipeline.

As shown in Figure 3, LLM-based filtering yields modest improvements (less than 1 point) over linter-based filtering on the HumanEval and HumanEval+ benchmarks.

Given these limited gains, we compare LLM-based scoring to our LLM-driven rewriting pipeline, which refines code snippets by enhancing clarity and correctness (Section 3.3). Comparing Figure 3 and Figure 4, although the rewriting pipeline requires 1.22 times the computational resources of LLM-based scoring (a 22% increase), it achieves significantly greater performance gains on HumanEval and HumanEval+ (detailed in Appendix F). Consequently, we do not incorporate LLM-based scoring in subsequent experiments, which favors the more effective and cost-efficient rewriting approach.

3.3 LLM-driven rewriting

Recent studies highlight the potential of large language models (LLMs) to transform training data, enhancing the performance of instruction-tuned models. Jain et al. [15] proposed three transformations—variable renaming, modularization, and plan annotation—for code cleaning, demonstrating their effectiveness for instruction tuning. However, their approach is limited to curated instruction-tuning data and a narrow set of stylistic improvements, lacking semantic optimizations or preprocessing integration.

To address these limitations, we developed the SwallowCode pipeline, which systematically rewrites large-scale pre-training corpora using two complementary LLM-driven processes: Style-Guided Code Rewriting (SGCR) and Self-Contained Optimization Rewriting (SCOR). SGCR revises Python code snippets based on the Google Python Style Guide [2], enforcing stylistic improvements like clear naming and modular design (Section 3.3.1). SCOR extends SGCR by ensuring self-containment and applying semantic optimizations, such as efficient algorithms and instructive examples (Section 3.3.2). To clarify, SwallowCode retains the code-only format of The-stack-v2-train-smol-ids [2], consisting solely of Python snippets without the text-code pair structure typical of instruction-tuning datasets. The rewriting pipeline, powered by Llama-3.3-70B-Instruct, enhances code quality through stylistic and semantic transformations without introducing instructional prompts or responses. Thus, performance gains on HumanEval and HumanEval+ (Section 3.4) stem from improved data quality, not from distillation of instruction tuning capabilities.

To illustrate the scope of these transformations, Table 1 compares the coverage of SGCR and SCOR against the approach of Jain et al. [15]. While SGCR addresses stylistic criteria like type hints, error handling, and docstrings, SCOR introduces semantic enhancements, including self-containment and optimization (algorithm and data structure), broadening the scope of the SwallowCode pipeline.

3.3.1 SGCR: Style-Guided Code Rewriting

SGCR enhances code readability by adding docstrings and type hints, unifying variable reassignment patterns, and standardizing function and class names in accordance with the Google Python Style

[2] https://google.github.io/styleguide/pyguide.html

Table 1: Comparison of code transformation coverage for Jain et al. [15], SGCR, and SCOR.

Criterion	Jain et al.	SGCR	SCOR
Variable Renaming	✓	✓	×
Modularization	✓	✓	×
Comments	✓	✓	×
Type Hint	×	✓	×
Error Handling	×	✓	×
Docstring	×	✓	×
Self-Contained	×	×	✓
Optimized	×	×	✓

Guide. Compared to the pre-rewriting (syntax-error and linter-based filtering), SGCR achieves improvements over 9 points on both HumanEval and HumanEval+ as shown Figure 4. We also evaluate SGCR applied directly to the raw `the-stack-v2-train-smol-ids` corpus versus SGCR applied after syntax-error and linter-based filtering. As illustrated in Figure 4, the SGCR pipeline with syntax and linter filtering outperforms direct SGCR by approximately 2 points on downstream code-generation benchmarks. Consequently, we adopt the pipeline that syntax and linter-based filtering prior to SGCR in all subsequent experiments.

Ablation studies reveal that SGCR significantly improves HumanEval and HumanEval+ scores but results in an approximate 10-point decrease on the MBPP benchmark [12] (detailed in Appendix G). Analysis indicates that MBPP's solutions often use non-standard function and class names, and SGCR's automated renaming introduces function name mismatches with MBPP's unit tests, leading to "undefined" errors during evaluation. The identified mismatches obscure the model's true code-generation capabilities, motivating us to exclude MBPP from our evaluation benchmarks across all experiments.

3.3.2 SCOR: Self-Contained Optimization Rewriting

Although SGCR ensures adherence to stylistic criteria, it does not modify the program semantics. Manual observation of models trained on SGCR-processed data reveals three recurring issues: (i) missing dependencies, where models attempt to import non-existent libraries or call undefined functions, causing runtime errors; (ii) inefficient algorithms, such as using naive recursion or quadratic-order algorithms for problems that linear-order or dynamic-programming solutions; and (iii) trivial snippets, such as code that merely prints constants or performs basic arithmetic, offering minimal training value.

To address these limitations while preserving SGCR's stylistic improvements, we introduce Self-Contained Optimization Rewriting (SCOR). Guided by a ten-rule prompt (detailed in Appendix D.2), SCOR rewrites each snippet to ensure self-containment by inlining or satisfying external dependencies, replaces inefficient algorithms with more computationally efficient alternatives, and transforms trivial code into meaningful, executable examples. As illustrated in Figure 4, SCOR improves HumanEval and HumanEval+ scores by over 5 points compared to SGCR. These results underscore the importance of semantic-level rewrites beyond stylistic enhancements, establishing SCOR as the final stage of the SwallowCode construction pipeline. We did not conduct an ablation experiment evaluating SCOR in isolation without SGCR. However, prompt validation experiments with Llama-3.3-70B-Instruct indicated that simultaneously applying SGCR and SCOR often reduced the quality of the rewritten code due to the challenges LLMs face in balancing multiple objectives. This led to the adoption of a two-stage SGCR and SCOR approach, with the isolated SCOR evaluation deferred to future work.

Compared to the 2-point performance gain achieved during the filtering stages (green line in Figure 4), the rewriting stages using SGCR and SCOR led to a total performance improvement of 14 points—9 points from SGCR and 5 points from SCOR. This highlights the significant potential for enhancing dataset curation by incorporating LLM-driven rewriting approach.

3.4 The Final SwallowCode dataset

Applying the complete pipeline, including syntax error filtering, Pylint-based filtering, Style-Guided Code Rewriting (SGCR), and Self-Contained Optimization Rewriting (SCOR), to

`the-stack-v2-train-smol-ids` produces the **SwallowCode** corpus, comprising **16.1 billion** tokens. All intermediate artifacts, including non-optimal variants, are publicly available to support future research efforts.

Comparison with existing corpora Figure 1 compares SwallowCode with several widely used open code datasets: CodeParrot-Clean (12.8 billion tokens)[3], The Stack v1 (98.2 billion tokens) [1], The Stack v2-Smol (36.1 billion tokens) [2], and Stack-Edu (17.9 billion tokens) [3]. For a fair comparison, we extract only the Python subsets of each corpus and, following the protocol outlined in Section 3.1, allocate 16% (8 billion) code tokens within a 50 billion-token mixed batch, ensuring each sample is processed no more than once. SwallowCode outperforms all comparable publicly available corpora on the HumanEval (pass@1) and HumanEval+ (pass@1) benchmarks, demonstrating the effectiveness of our pipeline design. Detailed results are provided in Appendix H.1.

Language generality While our experiments focus on Python to facilitate automated evaluation, the pipeline's components are language-agnostic, requiring only (i) static syntax checking and (ii) the availability of a linter or style tool. Thus, the methodology can be readily adapted to other programming languages with minimal modifications.

4 Construction of the math corpus

Section 3 demonstrated that LLM-driven rewriting significantly boosts coding performance. To evaluate the transferability of this approach, we apply a tailored rewriting pipeline to mathematical data. We select finemath-4+ [3], a high-quality, publicly available math corpus, as the starting point and process it through a rewriting pipeline. According to the evaluation in finemath [3], it outperforms other open mathematical datasets, including OpenWebMath [4], InfiMM-Math [33], and finemath-3+, in terms of the performance on benchmarks such as GSM8K and MATH. Given its reported superior performance and public availability, we adopt finemath-4+ as the foundation corpus for constructing our mathematical corpus to maximize the effectiveness of our rewriting approach.

4.1 Experimental setup

We adhere to the protocol outlined in Section 3.1, performing continual pre-training of Llama-3.1-8B for approximately 50 billion tokens, varying only the target math corpus. The evaluation benchmarks mirror those of Section 3.1, with HumanEval+ replaced by the MATH dataset [34], with GSM8K and MATH as the primary math-focused benchmarks. The pre-training mixture comprises 82.2% multilingual text, 13.0% code, and 4.79% math; detailed proportions and data sources are provided in Appendix A.4.2. The complete hyperparameters are listed in the Appendix A.1.

4.2 LLM-driven rewriting

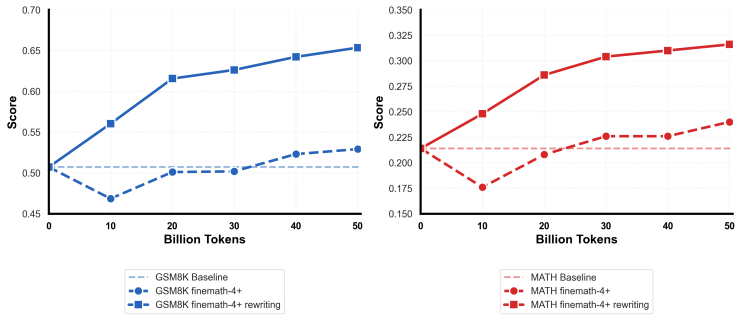

Figure 5: Performance gains from LLM-driven rewriting of finemath-4+. The rewritten corpus improves GSM8K(left) by 12.4 points and MATH(right) by 7.6 points.

The Finemath-4+ corpus [3] is a collection of documents in which snippets of mathematical text are embedded within passages that are otherwise unrelated to mathematics. In addition, the mathematical

[3] https://huggingface.co/datasets/codeparrot/codeparrot-clean

content ranges widely, from elementary arithmetic to advanced topics. This heterogeneity renders rule-based filtering challenging, as it struggles to distinguish relevant mathematical content from irrelevant artifacts. To address this, we design an LLM-driven rewriting pipeline using Llama-3.3-70B-Instruct, which not only cleans and refines the data but also enhances its quality for mathematical reasoning tasks.

The rewriting prompt instructs the model to: (1) remove residual web headers, footers, and privacy notices; (2) eliminate irrelevant metadata, such as question and answer timestamps; (3) restore missing context in incomplete questions or answers; (4) rewrite derivation steps to be concise yet comprehensive; and (5) provide clear, step-by-step solutions. Steps (1) and (2) are analogous to the syntax error and linter-based filtering applied to SwallowCode (Section 3), addressing inappropriate content that rule-based methods alone could not effectively filter. Steps (3) through (5) parallel the self-containment and style enhancements of the code rewriting pipeline, adapting them to the mathematical domain. The complete prompt is provided in Appendix E.

As shown in Figure 5, the rewritten corpus yields substantial improvements: 12.4 points on GSM8K and 7.6 points on MATH. These results demonstrate that LLM-driven rewriting, while tailored to the unique characteristics of mathematical data, successfully enhances the already high-quality finemath-4+ corpus. This confirms the generalizability of our rewriting approach beyond code, offering a robust method for improving open-domain datasets for mathematical reasoning.

5 Limitations

While SwallowCode and SwallowMath significantly improve code generation and mathematical reasoning performance of LLMs, several limitations should be noted. First, the rewriting pipelines may preserve biases present in the source data. For example, `the-stack-v2-train-smol-ids` may over-represent certain coding patterns (e.g., recursive algorithms over iterative ones), and Finemath-4+ may favor specific problem types (e.g., algebra over geometry). Additionally, as the rewriting process relies on Llama-3.3-70B-Instruct, the resulting datasets may reflect this model's preferences, such as favoring certain variable naming conventions or solution strategies, potentially limiting generalizability. Second, our evaluations are confined to continual pre-training with a 50 billion token budget, as detailed in Section 3.1, to ensure controlled and reproducible experiments within computational constraints. The impact of extending pre-training beyond this budget remains unexplored, and performance trends at larger scales may differ, particularly for tasks requiring extensive training data. Third, although the SwallowCode pipeline is designed to be language-agnostic, requiring only static syntax checking and linter tools, our experiments focus exclusively on Python to facilitate automated evaluation. Empirical validation for other languages (for example, Java, C++) was not feasible due to resource constraints, limiting evidence of the pipeline's broader applicability.

6 Conclusion

In this work, we present SwallowCode and SwallowMath, two pre-training datasets released under the Llama 3.3 Community License, designed to enhance large language model (LLM) capabilities in program synthesis and mathematical reasoning. SwallowCode, spanning approximately 16.1 billion tokens, refines Python code from The-Stack-v2 using a transform-and-retain pipeline that includes syntax validation, style-guided rewriting, and self-contained optimization. SwallowMath, with approximately 2.3 billion tokens, upgrades Finemath-4+ by reformatting mathematical content into concise, context-rich solutions. Unlike traditional approaches that discard low-quality data, our method systematically enhances it, maximizing utility and setting a new standard for dataset curation.

SwallowCode and SwallowMath achieve state-of-the-art performance among publicly available datasets for code and math reasoning, demonstrating the efficacy of our data refinement strategies. The language-agnostic design of our pipeline, which requires only parseable syntax and a linter, suggests its potential applicability to other programming languages. To foster reproducible research and drive advancements in LLM pretraining, we release SwallowCode, SwallowMath, all associated prompts, and model checkpoints. These resources establish a scalable framework for high-quality data curation, paving the way for enhanced automated reasoning and software development.

References

[1] Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. The stack: 3 tb of permissively licensed source code, 2022. URL https://arxiv.org/abs/2211.15533.

[2] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder 2 and the stack v2: The next generation, 2024. URL https://arxiv.org/abs/2402.19173.

[3] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. Smollm2: When smol goes big – data-centric training of a small language model, 2025. URL https://arxiv.org/abs/2502.02737.

[4] Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. Openwebmath: An open dataset of high-quality mathematical web text, 2023. URL https://arxiv.org/abs/2310.06786.

[5] Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 olmo 2 furious, 2025. URL https://arxiv.org/abs/2501.00656.

[6] NVIDIA, :, Aaron Blakeman, Aarti Basant, Abhinav Khattar, Adithya Renduchintala, Akhiad Bercovich, Aleksander Ficek, Alexis Bjorlin, Ali Taghibakhshi, Amala Sanjay Deshmukh, Ameya Sunil Mahabaleshwarkar, Andrew Tao, Anna Shors, Ashwath Aithal, Ashwin Poojary, Ayush Dattagupta, Balaram Buddharaju, Bobby Chen, Boris Ginsburg, Boxin Wang, Brandon Norick, Brian Butterfield, Bryan Catanzaro, Carlo del Mundo, Chengyu Dong, Christine Harvey, Christopher Parisien, Dan Su, Daniel Korzekwa, Danny Yin, Daria Gitman, David Mosallanezhad, Deepak Narayanan, Denys Fridman, Dima Rekesh, Ding Ma, Dmytro Pykhtar, Dong Ahn, Duncan Riach, Dusan Stosic, Eileen Long, Elad Segal, Ellie Evans, Eric Chung, Erick Galinkin, Evelina Bakhturina, Ewa Dobrowolska, Fei Jia, Fuxiao Liu, Gargi Prasad, Gerald Shen, Guilin Liu, Guo Chen, Haifeng Qian, Helen Ngo, Hongbin Liu, Hui Li, Igor Gitman, Ilia Karmanov, Ivan Moshkov, Izik Golan, Jan Kautz, Jane Polak Scowcroft, Jared Casper, Jarno Seppanen, Jason Lu, Jason Sewall, Jiaqi Zeng, Jiaxuan You, Jimmy Zhang, Jing Zhang, Jining Huang, Jinze Xue, Jocelyn Huang, Joey Conway, John Kamalu, Jon Barker, Jonathan Cohen, Joseph Jennings, Jupinder Parmar, Karan Sapra, Kari Briski, Kateryna Chumachenko, Katherine Luna, Keshav Santhanam, Kezhi Kong, Kirthi Sivamani, Krzysztof Pawelec, Kumar Anik, Kunlun Li, Lawrence McAfee, Leon Derczynski, Lindsey Pavao, Luis Vega, Lukas Voegtle, Maciej Bala, Maer Rodrigues de Melo, Makesh Narsimhan Sreedhar, Marcin Chochowski, Markus Kliegl, Marta Stepniewska-Dziubinska, Matthieu Le, Matvei Novikov, Mehrzad Samadi, Michael Andersch, Michael Evans, Miguel Martinez, Mike Chrzanowski, Mike Ranzinger, Mikolaj Blaz, Misha Smelyanskiy, Mohamed Fawzy, Mohammad Shoeybi, Mostofa Patwary,

Nayeon Lee, Nima Tajbakhsh, Ning Xu, Oleg Rybakov, Oleksii Kuchaiev, Olivier Delalleau, Osvald Nitski, Parth Chadha, Pasha Shamis, Paulius Micikevicius, Pavlo Molchanov, Peter Dykas, Philipp Fischer, Pierre-Yves Aquilanti, Piotr Bialecki, Prasoon Varshney, Pritam Gundecha, Przemek Tredak, Rabeeh Karimi, Rahul Kandu, Ran El-Yaniv, Raviraj Joshi, Roger Waleffe, Ruoxi Zhang, Sabrina Kavanaugh, Sahil Jain, Samuel Kriman, Sangkug Lym, Sanjeev Satheesh, Saurav Muralidharan, Sean Narenthiran, Selvaraj Anandaraj, Seonmyeong Bak, Sergey Kashirsky, Seungju Han, Shantanu Acharya, Shaona Ghosh, Sharath Turuvekere Sreenivas, Sharon Clay, Shelby Thomas, Shrimai Prabhumoye, Shubham Pachori, Shubham Toshniwal, Shyamala Prayaga, Siddhartha Jain, Sirshak Das, Slawek Kierat, Somshubra Majumdar, Song Han, Soumye Singhal, Sriharsha Niverty, Stefania Alborghetti, Suseella Panguluri, Swetha Bhendigeri, Syeda Nahida Akter, Szymon Migacz, Tal Shiri, Terry Kong, Timo Roman, Tomer Ronen, Trisha Saar, Tugrul Konuk, Tuomas Rintamaki, Tyler Poon, Ushnish De, Vahid Noroozi, Varun Singh, Vijay Korthikanti, Vitaly Kurin, Wasi Uddin Ahmad, Wei Du, Wei Ping, Wenliang Dai, Wonmin Byeon, Xiaowei Ren, Yao Xu, Yejin Choi, Yian Zhang, Ying Lin, Yoshi Suhara, Zhiding Yu, Zhiqi Li, Zhiyu Li, Zhongbo Zhu, Zhuolin Yang, and Zijia Chen. Nemotron-h: A family of accurate and efficient hybrid mamba-transformer models, 2025. URL https://arxiv.org/abs/2504.03624.

[7] Marah I Abdin, Jyoti Aneja, Harkirat S. Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J. Hewett, Mojan Javaheripi, Piero Kauffmann, James R. Lee, Yin Tat Lee, Yuanzhi Li, Weishung Liu, Caio C. T. Mendes, Anh Nguyen, Eric Price, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Xin Wang, Rachel Ward, Yue Wu, Dingli Yu, Cyril Zhang, and Yi Zhang. Phi-4 technical report, 2024. URL https://doi.org/10.48550/arXiv.2412.08905.

[8] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal

Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3

herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

[9] Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

[10] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Leandro von Werra, and Thomas Wolf. Smollm - blazingly fast and remarkably powerful, 2024.

[11] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder: may the source be with you!, 2023. URL https://arxiv.org/abs/2305.06161.

[12] 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.

[13] Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. Cosmopedia, February 2024. URL https://huggingface.co/datasets/HuggingFaceTB/cosmopedia. Software dataset.

[14] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

[15] Naman Jain, Tianjun Zhang, Wei-Lin Chiang, Joseph E. Gonzalez, Koushik Sen, and Ion Stoica. Llm-assisted code cleaning for training accurate code generators, 2023. URL https://arxiv.org/abs/2311.14904.

[16] Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing, 2024. URL https://arxiv.org/abs/2406.08464.

[17] Hao Chen, Abdul Waheed, Xiang Li, Yidong Wang, Jindong Wang, Bhiksha Raj, and Marah I. Abdin. On the diversity of synthetic data and its impact on training large language models, 2024. URL https://arxiv.org/abs/2410.15226.

[18] Nvidia, :, Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H. Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan Cohen, Sirshak Das, Ayush Dattagupta, Olivier Delalleau, Leon Derczynski, Yi Dong, Daniel Egert, Ellie Evans, Aleksander Ficek, Denys Fridman, Shaona Ghosh, Boris Ginsburg, Igor Gitman, Tomasz Grzegorzek, Robert Hero, Jining Huang, Vibhu Jawa, Joseph Jennings, Aastha Jhunjhunwala, John Kamalu, Sadaf Khan, Oleksii Kuchaiev, Patrick LeGresley, Hui Li, Jiwei Liu, Zihan Liu, Eileen Long, Ameya Sunil Mahabaleshwarkar, Somshubra Majumdar, James Maki, Miguel Martinez, Maer Rodrigues de Melo, Ivan Moshkov, Deepak Narayanan, Sean Narenthiran, Jesus Navarro, Phong Nguyen, Osvald Nitski, Vahid Noroozi, Guruprasad Nutheti, Christopher Parisien, Jupinder Parmar, Mostofa Patwary, Krzysztof Pawelec, Wei Ping, Shrimai Prabhumoye, Rajarshi Roy, Trisha Saar, Vasanth Rao Naik Sabavat, Sanjeev Satheesh, Jane Polak Scowcroft, Jason Sewall, Pavel Shamis, Gerald Shen, Mohammad Shoeybi, Dave Sizer, Misha Smelyanskiy,

Felipe Soares, Makesh Narsimhan Sreedhar, Dan Su, Sandeep Subramanian, Shengyang Sun, Shubham Toshniwal, Hao Wang, Zhilin Wang, Jiaxuan You, Jiaqi Zeng, Jimmy Zhang, Jing Zhang, Vivienne Zhang, Yian Zhang, and Chen Zhu. Nemotron-4 340b technical report, 2024. URL https://arxiv.org/abs/2406.11704.

[19] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020. URL https://arxiv.org/abs/1909.08053.

[20] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.

[21] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

[22] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

[23] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[24] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[25] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad, 2018.

[26] Alexey Tikhonov and Max Ryabinin. It's all in the heads: Using attention heads as a baseline for cross-lingual transfer in commonsense reasoning, 2021.

[27] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

[28] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.

[29] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

[30] 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, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

[31] 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, 2023. URL https://arxiv.org/abs/2305.01210.

[32] Kazuki Fujii, Taishi Nakamura, Mengsay Loem, Hiroki Iida, Masanari Ohi, Kakeru Hattori, Hirai Shota, Sakae Mizuki, Rio Yokota, and Naoaki Okazaki. Continual pre-training for cross-lingual llm adaptation: Enhancing japanese language capabilities, 2024. URL https://arxiv.org/abs/2404.17790.

[33] Xiaotian Han, Yiren Jian, Xuefeng Hu, Haogeng Liu, Yiqi Wang, Qihang Fan, Yuang Ai, Huaibo Huang, Ran He, Zhenheng Yang, and Quanzeng You. Infimm-webmath-40b: Advancing multimodal pre-training for enhanced mathematical reasoning, 2024. URL https://arxiv.org/abs/2409.12568.

[34] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[35] Naoaki Okazaki, Kakeru Hattori, Hirai Shota, Hiroki Iida, Masanari Ohi, Kazuki Fujii, Taishi Nakamura, Mengsay Loem, Rio Yokota, and Sakae Mizuki. Building a large japanese web corpus for large language models, 2024. URL https://arxiv.org/abs/2404.17733.

[36] Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. Datacomp-lm: In search of the next generation of training sets for language models, 2025. URL https://arxiv.org/abs/2406.11794.

[37] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm, 2021. URL https://arxiv.org/abs/2104.04473.

A Detailed setup for data ablation experiments

This section provides details on the training hyperparameters, training library versions, training environments, distributed training settings, and data mixture used in the dataset ablation experiments described in Sections 3 and 4.

A.1 Training hyperparameters

We performed continual pre-training from Llama-3.1-8B [4] using approximately 50 billion tokens. As shown in Table 2, the model architecture and tokenizer are identical to those of Llama-3.1-8B. The training hyperparameters are detailed in Table 3.

Table 2: Training model's architecture

Hyperparameter	Value
Architecture	Llama-3
Hidden size	4 096
FFN hidden size	14 336
Number of layers	32
Number of attention heads	32
Number of key/value heads	8
Sequence length	8 192
Normalization	RMSNorm
RMSNorm epsilon	1.0×10^{-5}
RoPE base	500000
Attention dropout	0.0
Hidden dropout	0.0
Tokenizer	Llama-3 tokenizer

Table 3: Training hyperparameters

Hyperparameter	Value
Adam beta1	0.9
Adam beta2	0.95
Adam epsilon	1.0×10^{-8}
Gradient clipping	1.0
Weight Decay	0.1
Learning rate (max)	2.5×10^{-5}
Learning rate (min)	2.5×10^{-6}
Warmup steps	1000
Warmup style	linear
Decay style	cosine

A.2 Training environment

We utilized the TSUBAME[5] supercomputer of the Institute of Science Tokyo, for training. We employed mixed precision (bfloat16) and used multiple NVIDIA H100 nodes for distributed parallel training. Each node is equipped with four NVIDIA H100 94GB GPUs, and the nodes are interconnected via InfiniBand NDR200.

We conducted continual pre-training with libraries shown in Table 4.

[4] https://huggingface.co/meta-llama/Llama-3.1-8B
[5] https://www.t4.cii.isct.ac.jp/docs/handbook.en/

Table 4: Training library versions

Component / Library	Version
Training library	Megatron-LM
mcore	0.9.0
CUDA Toolkit	12.4
cuDNN	9.1.0
NCCL	2.21.5
HPC-X	2.17.1
ninja	1.11.1
PyTorch	2.5.0
TransformerEngine	1.12

A.3 Distributed training settings

Training large language models on a single GPU is challenging due to both GPU memory constraints and the time required for training. In terms of GPU memory, even using the latest H100 80GB, it is difficult to train the 8B model used in this study. Moreover, even if the model parameters, gradients, and optimizer states could fit on a single GPU, training on a single GPU would require an unrealistic amount of time to complete. Therefore, in this study, we adopted distributed parallel training, combining data parallelism and model parallelism. We conducted all ablation experiments with the distributed setting shown in Table 5.

Table 5: Distributed training setting for ablation experiments

Hyperparameter	Value
Data Parallelism (DP)	32
Tensor Parallelism (TP)	2
Context Parallelism (CP)	1
Pipeline Parallelism (PP)	1
Micro batch size	2
Global batch size	512
Sequence Parallelism	true
Distributed optimizer	true
Tensor Parallelism Communication Overlap	true

A.4 Data mixture for ablation experiments

The SwallowProject[6], a collaborative effort between the Institute of Science Tokyo, Japan, and the National Institute of Advanced Industrial Science and Technology (AIST), aims to develop open-source large language models (LLMs) with strong capabilities in both Japanese and English. A key challenge in continual pre-training from high-performing models like Llama-3.1-8B is mitigating catastrophic forgetting, particularly in maintaining or improving mathematical reasoning and code generation performance. To address this, our ablation experiments were designed to enhance Llama-3.1-8B's performance on HumanEval, HumanEval+, GSM8K, and MATH while incorporating a multilingual dataset predominantly composed of Japanese and English text. This approach led to the development of SwallowCode and SwallowMath, as detailed in Sections 3 and 4. The data mixture reflects a high proportion of Japanese text, consistent with our project's focus on bilingual proficiency. However, as described in Sections 3.1 and 4.1, all ablation experiments maintain identical settings except for the target code or math dataset, ensuring a fully controlled experimental design.

This continual pre-training strategy aligns with established practices in high-quality LLM development, as seen in models like OLMo-2 [5] and Nemotron-H [6]. Specifically, adopting a staged pre-training approach, as observed in OLMo-2 and Nemotron-H, where later training phases leverage high-quality multilingual text alongside specialized math and code datasets to enhance LLM capabilities, ensures that our ablation experiments align with realistic LLM training scenarios.

[6] https://swallow-llm.github.io/index.en.html

A.4.1 Code ablation data mixture

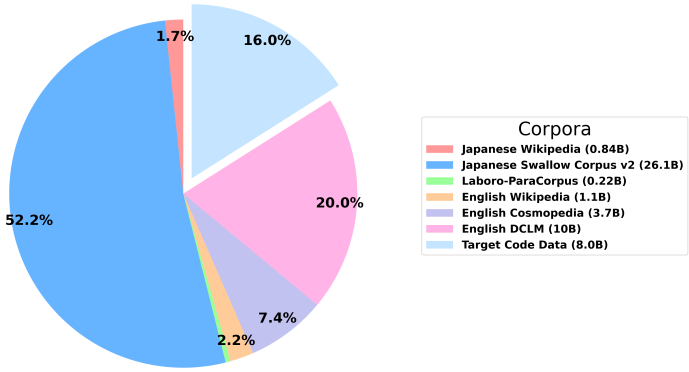

Figure 6: Data ratio for SwallowCode ablation experiments.

The training dataset for the code ablation experiments comprises approximately 50 billion tokens. The distribution of the dataset components is illustrated in Figure 6, with the following components and their respective token counts. Note that the **Target Code Data** varies depending on the specific ablation experiment conducted.

- **Japanese Wikipedia**[7]: 0.84 billion tokens
- **Japanese Swallow Corpus v2** [35]: 26.1 billion tokens
- **Laboro-ParaCorpus**[8]: 0.22 billion tokens
- **English Wikipedia**[9]: 1.1 billion tokens
- **English Cosmopedia** [13]: 3.7 billion tokens
- **English DCLM** [36]: 10.0 billion tokens
- **Target Code Data**: 8.0 billion tokens

A.4.2 Math ablation data mixture

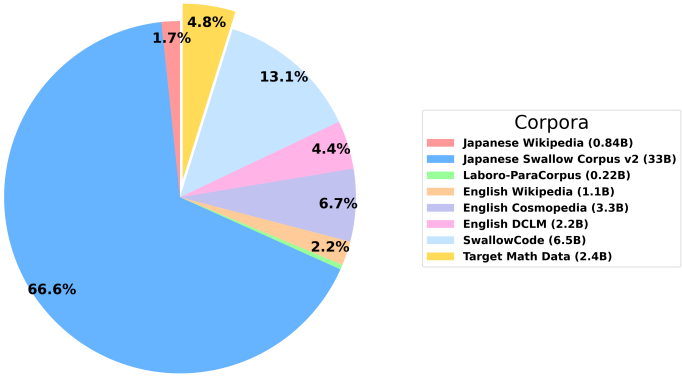

Figure 7: Data ratio for SwallowMath ablation experiments.

The training dataset for the math ablation experiments also consists of approximately 50 billion tokens. The composition of the dataset is shown in Figure 7, with the following components and their respective token counts. Note that the **Target Math Data** varies depending on the specific ablation experiment conducted.

[7] https://dumps.wikimedia.org/jawiki/
[8] https://github.com/laboroai/Laboro-ParaCorpus
[9] https://dumps.wikimedia.org/enwiki/

- **Japanese Wikipedia**[10]: 0.84 billion tokens
- **Japanese Swallow Corpus v2** [35]: 33.0 billion tokens
- **Laboro-ParaCorpus**[11]: 0.22 billion tokens
- **English Wikipedia**[12]: 1.1 billion tokens
- **English Cosmopedia** [13]: 3.3 billion tokens
- **English DCLM** [36]: 2.2 billion tokens
- **SwallowCode (Syntax, Pylint Filtered)**: 6.5 billion tokens
- **Target Math Data**: 2.4 billion tokens

B Code linting filtering

A threshold of 7.0 balances code quality and dataset size, while the comment penalty reduces overly verbose or non-functional scripts. As described in Section 3.2.2, the code for performing linter filtering is publicly available at https://github.com/rioyokotalab/swallow-code-math. The relevant code is also provided below.

When conducting linting with pylint, warnings and errors dependent on the linting environment, such as import errors, are excluded using the `--disable` option. Additionally, some files with Python extensions primarily contain comments with textual content and have minimal script functionality. To address this, as discussed in Section 3.2.2, we introduced a mechanism that imposes a heuristic penalty based on the proportion of comments to filter out such files.

```python
def check_comment_ratio(code: str):
    total_lines = 0
    comment_lines = 0

    try:
        tokens = tokenize.generate_tokens(StringIO(code).readline)
        for token_type, _, _, _, _ in tokens:
            total_lines += 1
            if token_type == tokenize.COMMENT:
                comment_lines += 1

    except tokenize.TokenError as e:
        print(f"Token error encountered: {str(e)}")
        return 0
    except IndentationError as e:
        print(f"indentation error encountered {str(e)}")
        return 0

    if total_lines == 0:
        return 0

    return comment_lines / total_lines


def apply_comment_penalty(score: float, comment_ratio: float) -> float:
    if comment_ratio == 1.0:
        return 0.0
    elif comment_ratio > 0:
        penalty_factor = 1 - comment_ratio
        score *= penalty_factor
    return score


```

[10] https://dumps.wikimedia.org/jawiki/
[11] https://github.com/laboroai/Laboro-ParaCorpus
[12] https://dumps.wikimedia.org/enwiki/

```python
def check_code_quality(code: str):
    with tempfile.NamedTemporaryFile(delete=False, suffix=".py") as temp_file:
        temp_file.write(code.encode())
        temp_file.flush()

        result = subprocess.run(
            ["pylint", "--persistent=n","--disable=E0401,C0114,C0301,C0103,C0116,C0411,R0903,W0511,C0412", temp_file.name],
            capture_output=True,
            text=True,
        )

    pylint_output = result.stdout
    score = None

    for line in pylint_output.split("\n"):
        if "Your code has been rated at" in line:
            score = float(line.split("/")[0].split()[-1])

    comment_ratio = check_comment_ratio(code)

    if score is not None:
        score = apply_comment_penalty(score, comment_ratio)

    return score, pylint_output
```

C Code LLM-based scoring

As described in Section 3.2.3, this section presents the prompt used for LLM-based scoring. The prompt was provided to Llama-3.3-70B-Instruct to evaluate code quality. The scoring criteria were developed with reference to the Google Python Style Guide [13]. The actual code pipeline is available at https://github.com/rioyokotalab/swallow-code-math. The dataset created through this process is available at https://huggingface.co/datasets/tokyotech-llm/swallow-code, and the checkpoints of the model trained on this dataset are available at https://huggingface.co/collections/tokyotech-llm/swallowcode-6811c84ff647568547d4e443.

Prompt Used for Code Quality Evaluation

```
You are a smart software engineer.  Please evaluate the following code on a
scale of 1 to 10 based on the following criteria:
1.  Are variable names descriptive and consistent with naming conventions?
2.  Are comments and docstrings appropriately written to explain the purpose
and functionality of the code?
3.  Are type annotations used effectively where applicable?
4.  Are functions appropriately modularized, with well-defined
responsibilities and clear separation of concerns?
5.  Are variables' lifetimes intentionally managed, avoiding frequent
reassignment or overly long scopes?
6.  Is error handling implemented appropriately where necessary?
7.  Is the code properly indented and follows standard formatting guidelines?
8.  Do comments provide context and rationale, rather than merely describing
what the code does?
9.  Are functions and classes designed with clear, single responsibilities?
10.  Is the code formatted in a way that enhances readability?
```

Figure 8 illustrates the distribution of the quality scores assigned.

[13] https://google.github.io/styleguide/pyguide.html

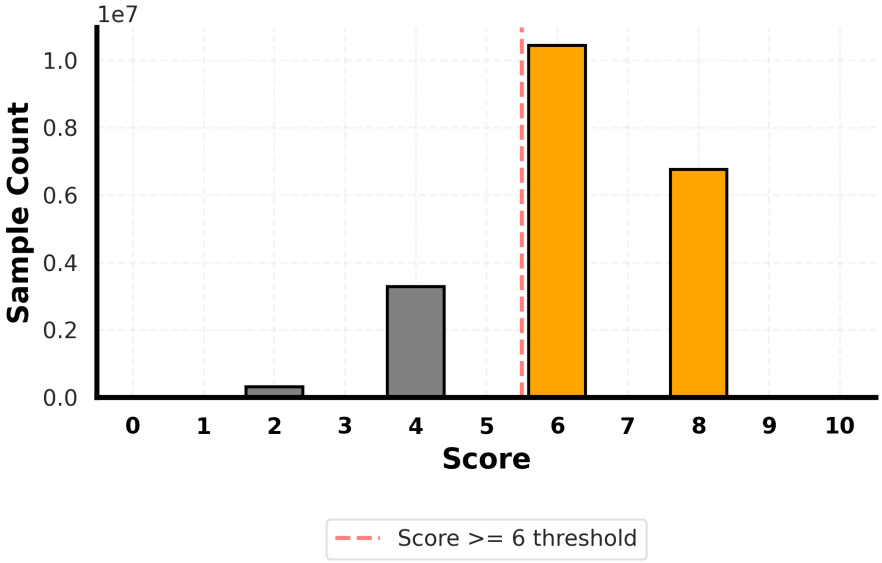

Figure 8: Distribution of quality scores assigned by Llama-3.3-70B-Instruct to the Python code corpus. The dashed line at 6 denotes the filtering threshold. Notably, only even scores appear, an emergent property of how Llama-3.3-70B-Instruct interpreted the scoring prompts.

D Code LLM rewriting

As described in Section 3.3, this section presents the prompts used for LLM-based rewriting, specifically for Style-Guided Code Rewriting (SGCR) and Self-Contained Optimization Rewriting (SCOR). Each prompt was provided to Llama-3.3-70B-Instruct to perform data rewriting.

D.1 Style-Guided Code Rewriting (SGCR)

The prompt used for SGCR is provided below. The dataset created through this rewriting process is available at https://huggingface.co/datasets/tokyotech-llm/swallow-code, and the checkpoints of the model trained on this dataset are available at https://huggingface.co/collections/tokyotech-llm/swallowcode-6811c84ff647568547d4e443.

Prompt Used for SGCR

```
You are a smart software engineer.  Please evaluate the following code on a
scale of 1 to 10 based on the following criteria:
1.  Are variable names descriptive and consistent with naming conventions?
2.  Are comments and docstrings appropriately written to explain the purpose
and functionality of the code?
3.  Are type annotations used effectively where applicable?
4.  Are functions appropriately modularized, with well-defined
responsibilities and clear separation of concerns?
5.  Are variables' lifetimes intentionally managed, avoiding frequent
reassignment or overly long scopes?
6.  Is error handling implemented appropriately where necessary?
7.  Is the code properly indented and follows standard formatting guidelines?
8.  Do comments provide context and rationale, rather than merely describing
what the code does?
9.  Are functions and classes designed with clear, single responsibilities?
10.  Is the code formatted in a way that enhances readability?

And provide suggestions for improvement based on the evaluation criteria.
You can also provide an improved version of the code in the following style:
```

```
### Evaluation: 7
### Suggestions: Provide specific, actionable suggestions to improve the
code based on the evaluation criteria.

### Improved Code: Provide a revised version of the code incorporating the
suggested improvements.
```python
def improved_function(arg1: int, arg2: str) -> str:
# Your improved code here
pass
```
```

D.2 Self-Contained Optimization Rewriting (SCOR).

The prompt used for SCOR is provided below. The dataset created through this rewriting process is available at https://huggingface.co/datasets/tokyotech-llm/swallow-code, and the checkpoints of the model trained on this dataset are available at https://huggingface.co/collections/tokyotech-llm/swallowcode-6811c84ff647568547d4e443.

Prompt Used for SCOR

```
You are a smart software engineer. Please change a given code into
self-contained and well-structured code following the below best practices
and pythonic way.
1. Use meaningful variable and function names.
2. Write a clear and concise docstring for the function.
3. Use type hints for the function signature.
4. Write a clear and concise comment for the code block.
5. Ensure the code is self-contained and does not depend on external
variables.
6. Ensure the code is well-structured and easy to read.
7. Ensure the code is free of errors and runs correctly.
8. Ensure the code is optimized and does not have redundant operations.
9. Ensure the algorithm and data structures are efficient and concise.

If given code is not self-contained or too simple, please change it to a
more educational and useful code.
```

E Math LLM rewriting

As described in Section 4.2, this section presents the prompt used for LLM rewriting in the construction of SwallowMath. The prompt consists of five components: (1) remove residual web headers, footers, and privacy notices; (2) delete extraneous metadata such as question and answer timestamps; (3) fill in missing context when either the question or answer is incomplete; (4) rewrite explanations to be concise yet information-dense; and (5) present a clear step-by-step solution. Steps (1)–(2) parallel our syntax-error and linter filtering for code, while steps (3)–(5) correspond to the self-containment and style rewrites used in SwallowCode.

Prompt Used for Math Rewriting

```
You are an intelligent math tutor. You are given the following math problem
and answer with some unnecessary parts. Please remove the unneeded parts of
the questions. For example, the date of the question submitted, the answer
date, the privacy policy, the footer, the header, etc., should be removed.
However, please keep the main question and answer.
If questions or answers lack some information or are not elaborate, please
make them more informative and easy to understand. If needed, please add
more detail about the step-by-step calculation process.
```

F Computational cost

This section quantifies the computational cost, measured in GPU hours (H100), required for the LLM scoring, LLM rewriting, and ablation experiments conducted in this study. These measurements are based on empirical data. By analyzing the computational cost, we aim to elucidate the relationship between the resource demands of the data cleaning pipeline and the resulting improvements in downstream task performance for each method. Furthermore, by providing an estimate of the computational resources needed to reproduce our results, we seek to facilitate future research.

F.1 Computational cost of LLM scoring

The data synthesis process utilized vLLM 0.7.2 and PyTorch 2.5.1. The dataset was synthesized with a global batch size of 2048 and tensor parallelism of 4. Data generation was performed using four H100 (94 GB) GPUs per job. With an input processing speed of approximately 2000 tokens/s and an output generation speed of approximately 3000 tokens/s, and given an average input dataset length of 836 tokens, an average output dataset length of 1271 tokens, and a total sample count of 20,826,548, we estimate that the dataset creation for the experiments described in Section 3.2.3 consumed a total of 19,477 H100 GPU hours. This estimate excludes vLLM initialization and safetensor loading.

F.2 Computational cost of LLM rewriting

The synthesis process for LLM rewriting also employed vLLM 0.7.2 and PyTorch 2.5.1, with a global batch size of 2048 and tensor parallelism of 4. Data generation was conducted using 4 H100 (94GB) GPUs per job. The input processing speed was approximately 2000 tokens/s, and the output generation speed was approximately 3000 tokens/s. Given an average input dataset length of 836 tokens, an average output dataset length of 1819 tokens, and a total sample count of 20,826,548, we estimate that the dataset creation for the experiments described in Section 3.3.1 consumed a total of 23,703 H100 GPU hours. This estimate excludes vLLM initialization and safetensor loading.

F.3 Computational cost of continual pre-training data ablation experiments

The ablation experiments described in Sections 3.1 and 4.1 were conducted using 64 H100 (94GB) GPUs for 24.7 hours per experiment. Thus, each experiment consumed 1,580 H100 GPU hours. With a total of 15 experiments (13 code ablation experiments and two math ablation experiments), the overall computational cost amounted to 23,700 H100 GPU hours. These experiments achieved a training throughput of 530 TFLOP/s/GPU and 590,000 tokens/s, as derived from the FLOP/s formula presented in the Megatron-LM paper [37]. This corresponds to approximately 53.5% of the peak BF16 Tensor Core performance of the H100.

G MBPP

As discussed in Section 3.3.1, the MBPP dataset [14] contains Python functions with naming conventions that deviate from standard Python style guidelines. For example, the following code snippet uses camelCase instead of the recommended snake_case:

```
def is_Power_Of_Two(x):
    return x and (not(x & (x - 1)))
def differ_At_One_Bit_Pos(a, b):
    return is_Power_Of_Two(a ^ b)
```

As described in Section 3.3.1, Style-Guided Code Rewriting (SGCR) rewrites code to conform to Python's naming conventions [15], specifically enforcing snake_case for function names. Consequently, when tasked with implementing functions that use non-standard naming (e.g., camelCase), an LLM trained on SGCR-processed data may rewrite function names to adhere to snake_case. This leads to mismatches during MBPP evaluation, where calling a function with its original non-standard name results in an "is not defined" error.

[14] https://github.com/google-research/google-research/blob/master/mbpp/mbpp.jsonl

[15] https://peps.python.org/pep-0008/#descriptive-naming-styles

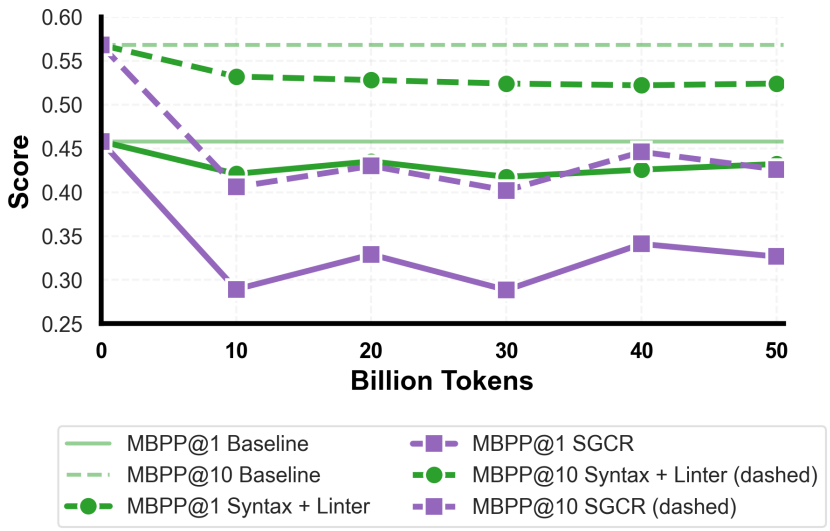

Figure 9: Comparison of MBPP@1 and MBPP@10 scores for models trained on linter-filtered data versus SGCR-rewritten data in a 50B-token continual pre-training ablation study. SGCR's enforcement of snake_case naming conventions leads to lower scores due to mismatches with MBPP's non-standard function names.

Figure 9 illustrates this issue: models trained on SGCR-processed data exhibit lower MBPP@1 and MBPP@10 scores compared to models trained on data processed only with syntax-error and linter-based filtering (Section 3.2.2). This performance drop stems from the naming mismatches described above, which obscure the model's true code generation capabilities. Based on this finding, we concluded that MBPP is not a suitable benchmark for evaluating LLM code generation in our experiments, as its evaluation framework penalizes adherence to standard Python naming conventions. Therefore, we excluded MBPP from the benchmarks used in this study.

H Evaluations

In this section, we present the evaluation results of models trained through ablation experiments on code and math datasets, assessed across ten benchmarks encompassing code and mathematical downstream tasks.

H.1 Code ablation experiments results

As described in Section 3.1, we evaluated models continually pre-trained from Llama-3.1-8B on ten English downstream tasks. In the following, we report the evaluation results for 13 code ablation experiments, with their relationships illustrated in Figure 10. Experiments exp1, exp8, exp9, and exp13 serve as baselines, utilizing only Python data extracted from existing open code corpora. The remaining experiments are conducted to construct the SwallowCode dataset. We evaluated performance using the following ten benchmarks: OpenBookQA [22], TriviaQA [23], HellaSwag [24], SQuAD 2.0 [25], XWinograd [26], MMLU [27], GSM8K [28], BBH [29], HumanEval [30], and HumanEval+ [31].

Table 6: Performance across benchmarks in the-stack-v2-train-smol-ids Python subset ablation.

Experiment 1: the-stack-v2-train-smol-ids Python subset										
Tokens (B)	**OpenBookQA**	**TriviaQA**	**HellaSwag**	**SQuAD2.0**	**XWINO**	**MMLU**	**GSM8K**	**BBH**	**HumanEval**	**HumanEval+**
10	0.3640	0.6659	0.5995	0.3354	0.9032	0.6294	0.4602	0.6019	0.3366	0.3366
20	0.3540	0.6567	0.6019	0.3360	0.9024	0.6238	0.4852	0.5898	0.3433	0.3433
30	0.3700	0.6588	0.6034	0.3377	0.9045	0.6263	0.5072	0.5939	0.3402	0.3421
40	0.3800	0.6618	0.6053	0.3380	0.9097	0.6341	0.5011	0.6016	0.3659	0.3701
50	0.3700	0.6679	0.6054	0.3350	0.9045	0.6340	0.5027	0.6091	0.3689	0.3720

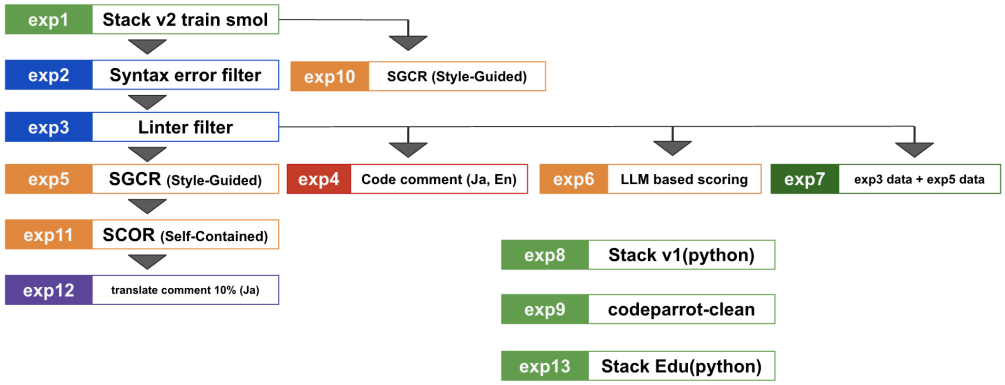

Figure 10: Relationships between code ablation experiments.

Table 7: Performance across benchmarks in syntax-error-free data ablation from Experiment 1.

Experiment 2: Syntax-error-free data from Experiment 1										
Tokens (B)	OpenBookQA	TriviaQA	HellaSwag	SQuAD2.0	XWINO	MMLU	GSM8K	BBH	HumanEval	HumanEval+
10	0.3560	0.6675	0.6015	0.3385	0.9062	0.6321	0.4784	0.5881	0.3604	0.3713
20	0.3520	0.6635	0.6026	0.3364	0.9049	0.6252	0.4784	0.5781	0.3591	0.3585
30	0.3560	0.6637	0.6012	0.3375	0.9080	0.6313	0.5019	0.5950	0.3701	0.3762
40	0.3580	0.6679	0.6046	0.3346	0.9062	0.6330	0.5019	0.5998	0.3720	0.3689
50	0.3660	0.6694	0.6055	0.3340	0.9084	0.6325	0.5155	0.6044	0.3787	0.3787

Table 8: Performance across benchmarks in syntax-error and Pylint-filtered (score ≥ 7) data ablation from Experiment 2.

Experiment 3: Syntax-error and Pylint-filtered (score ≥ 7) data from Experiment 2										
Tokens (B)	OpenBookQA	TriviaQA	HellaSwag	SQuAD2.0	XWINO	MMLU	GSM8K	BBH	HumanEval	HumanEval+
10	0.3560	0.6628	0.6010	0.3340	0.9071	0.6235	0.4564	0.6007	0.3500	0.3488
20	0.3500	0.6613	0.6015	0.3361	0.9054	0.6237	0.4860	0.5838	0.3744	0.3787
30	0.3620	0.6596	0.6008	0.3359	0.9080	0.6307	0.4867	0.5921	0.3957	0.3878
40	0.3720	0.6650	0.6030	0.3352	0.9058	0.6326	0.4822	0.5990	0.3890	0.3915
50	0.3740	0.6677	0.6054	0.3291	0.9019	0.6327	0.4996	0.6145	0.3945	0.3902

Table 9: Performance across benchmarks in comment-language-restricted (English and Japanese) data ablation from Experiment 3.

Experiment 4: Comment-language-restricted (English and Japanese) data from Experiment 3										
Tokens (B)	OpenBookQA	TriviaQA	HellaSwag	SQuAD2.0	XWINO	MMLU	GSM8K	BBH	HumanEval	HumanEval+
10	0.3640	0.6713	0.5988	0.3329	0.9054	0.6312	0.4708	0.5953	0.3549	0.3476
20	0.3520	0.6601	0.6011	0.3306	0.9067	0.6250	0.4898	0.5802	0.3689	0.3768
30	0.3680	0.6596	0.6047	0.3365	0.9118	0.6301	0.4989	0.5890	0.3768	0.3768
40	0.3660	0.6671	0.6049	0.3363	0.9071	0.6333	0.5155	0.6024	0.3756	0.3750
50	0.3700	0.6703	0.6061	0.3357	0.9101	0.6347	0.5133	0.6036	0.3841	0.3854

Table 10: Performance across benchmarks in SGCR-rewritten data ablation from Experiment 3.

Experiment 5: SGCR-rewritten data from Experiment 3										
Tokens (B)	OpenBookQA	TriviaQA	HellaSwag	SQuAD2.0	XWINO	MMLU	GSM8K	BBH	HumanEval	HumanEval+
10	0.3560	0.6689	0.5996	0.3295	0.9054	0.6256	0.4875	0.5991	0.4128	0.4110
20	0.3460	0.6610	0.6031	0.3352	0.9032	0.6262	0.4920	0.5801	0.4311	0.4616
30	0.3620	0.6637	0.6043	0.3378	0.9110	0.6269	0.5216	0.5984	0.4726	0.4579
40	0.3660	0.6645	0.6053	0.3372	0.9045	0.6328	0.4989	0.5945	0.4610	0.4390
50	0.3660	0.6667	0.6066	0.3325	0.9058	0.6352	0.5027	0.6065	0.4860	0.4811

Table 11: Performance across benchmarks in LLM-scored (score ≥ 6) data ablation from Experiment 3.

Experiment 6: LLM-scored (score ≥ 6) data from Experiment 3										
Tokens (B)	**OpenBookQA**	**TriviaQA**	**HellaSwag**	**SQuAD2.0**	**XWINO**	**MMLU**	**GSM8K**	**BBH**	**HumanEval**	**HumanEval+**
10	0.3640	0.6679	0.6002	0.3277	0.9041	0.6280	0.4701	0.5976	0.3640	0.3616
20	0.3540	0.6593	0.6010	0.3358	0.9045	0.6249	0.4822	0.5810	0.3659	0.3732
30	0.3660	0.6594	0.6021	0.3398	0.9071	0.6226	0.5140	0.5893	0.3994	0.3933
40	0.3700	0.6636	0.6021	0.3370	0.9080	0.6300	0.5027	0.6019	0.4018	0.4006
50	0.3640	0.6684	0.6046	0.3353	0.9084	0.6324	0.5011	0.6090	0.3951	0.3994

Table 12: Performance across benchmarks in mixed (1:1) data ablation from Experiments 3 and 5.

Experiment 7: Mixed (1:1) data from Experiments 3 and 5										
Tokens (B)	**OpenBookQA**	**TriviaQA**	**HellaSwag**	**SQuAD2.0**	**XWINO**	**MMLU**	**GSM8K**	**BBH**	**HumanEval**	**HumanEval+**
10	0.3620	0.6660	0.5994	0.3293	0.9032	0.6242	0.4738	0.6156	0.3616	0.3598
20	0.3460	0.6585	0.6018	0.3297	0.9024	0.6293	0.4845	0.5809	0.3823	0.3890
30	0.3680	0.6611	0.6022	0.3384	0.9062	0.6241	0.5110	0.6045	0.3848	0.3951
40	0.3640	0.6666	0.6028	0.3327	0.9088	0.6323	0.5072	0.6056	0.4018	0.4018
50	0.3680	0.6695	0.6052	0.3320	0.9097	0.6300	0.5027	0.6051	0.4116	0.3988

Table 13: Performance across benchmarks in the-stack-v1 Python subset ablation.

Experiment 8: the-stack-v1 Python subset										
Tokens (B)	**OpenBookQA**	**TriviaQA**	**HellaSwag**	**SQuAD2.0**	**XWINO**	**MMLU**	**GSM8K**	**BBH**	**HumanEval**	**HumanEval+**
10	0.3660	0.6646	0.6033	0.3310	0.9028	0.6219	0.4784	0.5955	0.3244	0.3341
20	0.3500	0.6595	0.6018	0.3233	0.9037	0.6246	0.4701	0.5898	0.3220	0.3232
30	0.3640	0.6575	0.6014	0.3279	0.9071	0.6226	0.5057	0.5878	0.3244	0.3287
40	0.3680	0.6638	0.6029	0.3265	0.9067	0.6320	0.5004	0.5984	0.3445	0.3500
50	0.3620	0.6650	0.6053	0.3212	0.9084	0.6273	0.5080	0.5998	0.3561	0.3409

Table 14: Performance across benchmarks in codepartto-clean Python subset ablation.

Experiment 9: codepartto-clean Python subset										
Tokens (B)	**OpenBookQA**	**TriviaQA**	**HellaSwag**	**SQuAD2.0**	**XWINO**	**MMLU**	**GSM8K**	**BBH**	**HumanEval**	**HumanEval+**
10	0.3540	0.6651	0.6006	0.3221	0.9062	0.6295	0.4708	0.5875	0.3598	0.3561
20	0.3560	0.6556	0.6013	0.3358	0.9067	0.6289	0.4731	0.5870	0.3549	0.3463
30	0.3680	0.6570	0.6045	0.3390	0.9071	0.6290	0.4890	0.5976	0.3524	0.3561
40	0.3720	0.6613	0.6048	0.3352	0.9075	0.6300	0.4958	0.6108	0.3543	0.3549
50	0.3600	0.6638	0.6055	0.3321	0.9097	0.6273	0.5072	0.6139	0.3616	0.3634

Table 15: Performance across benchmarks in SGCR-rewritten data ablation from Experiment 1.

Experiment 10: SGCR-rewritten data from Experiment 1										
Tokens (B)	**OpenBookQA**	**TriviaQA**	**HellaSwag**	**SQuAD2.0**	**XWINO**	**MMLU**	**GSM8K**	**BBH**	**HumanEval**	**HumanEval+**
10	0.3640	0.6667	0.5996	0.3325	0.9032	0.6164	0.4845	0.5959	0.4457	0.4433
20	0.3460	0.6592	0.6032	0.3324	0.9067	0.6231	0.4890	0.5655	0.4579	0.4506
30	0.3660	0.6585	0.6029	0.3379	0.9101	0.6176	0.5064	0.5855	0.4494	0.4433
40	0.3600	0.6650	0.6024	0.3339	0.9067	0.6284	0.5148	0.5967	0.4622	0.4622
50	0.3600	0.6687	0.6047	0.3337	0.9084	0.6317	0.5057	0.6041	0.4646	0.4622

Table 16: Performance across benchmarks in SCOR-rewritten data ablation from Experiment 5.

Experiment 11: SCOR-rewritten data from Experiment 5										
Tokens (B)	**OpenBookQA**	**TriviaQA**	**HellaSwag**	**SQuAD2.0**	**XWINO**	**MMLU**	**GSM8K**	**BBH**	**HumanEval**	**HumanEval+**
10	0.3580	0.6680	0.6006	0.3317	0.9067	0.6237	0.4655	0.6108	0.4567	0.4665
20	0.3500	0.6564	0.6026	0.3349	0.9084	0.6241	0.4981	0.5718	0.5384	0.5323
30	0.3620	0.6640	0.6023	0.3385	0.9054	0.6253	0.5095	0.5928	0.5256	0.5317
40	0.3640	0.6705	0.6041	0.3401	0.9088	0.6317	0.5095	0.5982	0.5226	0.5244
50	0.3700	0.6685	0.6055	0.3359	0.9114	0.6322	0.5110	0.6062	0.5396	0.5445

Table 17: Performance across benchmarks in mixed (90% Experiment 11, 10% Japanese-translated comments) data ablation.

Experiment 12: Mixed (90% Experiment 11, 10% Japanese-translated comments)										
Tokens (B)	**OpenBookQA**	**TriviaQA**	**HellaSwag**	**SQuAD2.0**	**XWINO**	**MMLU**	**GSM8K**	**BBH**	**HumanEval**	**HumanEval+**
10	0.3640	0.6625	0.6020	0.3341	0.9054	0.6221	0.4738	0.5697	0.4799	0.4768
20	0.3500	0.6551	0.6021	0.3361	0.9058	0.6266	0.4943	0.5776	0.5165	0.5079
30	0.3640	0.6595	0.6034	0.3410	0.9080	0.6250	0.5011	0.6008	0.5110	0.5091
40	0.3640	0.6640	0.6022	0.3361	0.9054	0.6330	0.4898	0.6008	0.5299	0.5268
50	0.3600	0.6655	0.6057	0.3340	0.9080	0.6315	0.5072	0.6057	0.5329	0.5293

Table 18: Performance across benchmarks in Stack Edu Python subset ablation.

Experiment 13: Stack Edu Python subset										
Tokens (B)	**OpenBookQA**	**TriviaQA**	**HellaSwag**	**SQuAD2.0**	**XWINO**	**MMLU**	**GSM8K**	**BBH**	**HumanEval**	**HumanEval+**
10	0.3640	0.6696	0.5986	0.3358	0.9037	0.6246	0.4761	0.6004	0.3470	0.3433
20	0.3520	0.6632	0.6021	0.3364	0.9067	0.6233	0.4898	0.5942	0.3537	0.3518
30	0.3660	0.6600	0.6024	0.3439	0.9097	0.6251	0.4989	0.5916	0.3713	0.3640
40	0.3700	0.6650	0.6033	0.3402	0.9067	0.6325	0.4958	0.6084	0.3701	0.3720
50	0.3740	0.6665	0.6061	0.3368	0.9062	0.6350	0.5087	0.6173	0.3695	0.3671

H.2 Math ablation experiments results

As described in Section 4.1, we evaluated models continually pre-trained from Llama-3.1-8B on ten English downstream tasks. Below, we present the evaluation results for two math ablation experiments. Specifically, we adopted the following ten evaluation benchmarks: OpenBookQA [22], TriviaQA [23], HellaSwag [24], SQuAD 2.0 [25], XWinograd [26], MMLU [27], GSM8K [28], BBH [29], HumanEval [30], and MATH [27].

Table 19: Performance across benchmarks in finemath-4+ ablation.

Experiment 1: finemath-4+										
Tokens (B)	**OpenBookQA**	**TriviaQA**	**HellaSwag**	**SQuAD2.0**	**XWINO**	**MMLU**	**HumanEval**	**GSM8K**	**BBH**	**MATH**
10	0.3700	0.6626	0.5990	0.3350	0.8985	0.6243	0.3439	0.4685	0.6057	0.1760
20	0.3720	0.6536	0.5963	0.3510	0.9032	0.6261	0.3622	0.5011	0.5896	0.2080
30	0.3700	0.6574	0.5999	0.3506	0.8998	0.6253	0.3561	0.5019	0.5971	0.2260
40	0.3720	0.6577	0.6024	0.3499	0.9049	0.6312	0.3701	0.5231	0.6054	0.2260
50	0.3740	0.6608	0.6001	0.3550	0.9058	0.6329	0.3561	0.5292	0.6166	0.2400

Table 20: Performance across benchmarks in finemath-4+ rewritten with Llama-3.3-70B-Instruct ablation.

Experiment 2: finemath-4+ rewriten(Llama-3.3-70B-Instruct)										
Tokens (B)	**OpenBookQA**	**TriviaQA**	**HellaSwag**	**SQuAD2.0**	**XWINO**	**MMLU**	**HumanEval**	**GSM8K**	**BBH**	**MATH**
10	0.3720	0.6643	0.5970	0.3443	0.9015	0.6343	0.3439	0.5603	0.5535	0.2480
20	0.3800	0.6580	0.5946	0.3428	0.8994	0.6293	0.3762	0.6156	0.5669	0.2860
30	0.3660	0.6618	0.5964	0.3470	0.9011	0.6298	0.3530	0.6262	0.6383	0.3040
40	0.3700	0.6610	0.5973	0.3535	0.9088	0.6358	0.3738	0.6422	0.6237	0.3100
50	0.3800	0.6637	0.5972	0.3537	0.9045	0.6337	0.3683	0.6535	0.6414	0.3160

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly outline the contributions, including the creation of SwallowCode and SwallowMath datasets, their performance improvements on benchmarks (e.g., +17.0 on HumanEval, +12.4 on GSM8K), and the novel rewriting pipeline. The scope is explicitly defined as enhancing LLM performance in Python code generation and mathematical reasoning.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 5 explicitly discusses limitations, including potential biases in source data (e.g., over-representation of certain coding patterns), the focus on Python for code evaluation, the 50 billion token training budget, and the reliance on Llama-3.3-70B-Instruct for rewriting, which may introduce model-specific preferences.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA] .

   Justification: The paper focuses on empirical contributions—dataset curation and LLM performance evaluation—without presenting theoretical results or proofs. The methodology relies on data processing pipelines and ablation experiments (Sections 3, 4).

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides comprehensive details for reproducing experiments, including training hyperparameters, data mixtures, compute environment, and distributed training settings (Sections 3.1, 4.1, Appendix A). Datasets, prompts, and model checkpoints are released at `https://huggingface.co/tokyotech-llm`, and code is available at `https://github.com/rioyokotalab/swallow-code-math`. Computational costs are quantified in Appendix F.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: SwallowCode and SwallowMath datasets, prompts, model checkpoints, and code are publicly available at https://huggingface.co/tokyotech-llm and https://github.com/rioyokotalab/swallow-code-math. Instructions for reproducing experiments are provided in Sections 3, 4, and Appendix A, including data processing pipelines and training configurations.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: The paper details the experimental setup, including model architecture (Llama-3.1-8B), training hyperparameters (e.g., learning rate, batch size), data mixtures, and evaluation benchmarks (Sections 3.1, 4.1, Appendix A.1). Data splits and evaluation result details are referenced in Appendix H.1.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No] .

   Justification: The choice of single-run metrics is justified by the high computational cost of multiple runs (Appendix F) and reinforced by evaluating five checkpoints per experiment at every 10B tokens, ensuring robust performance assessment across training stages. Validation across multiple benchmarks (HumanEval, HumanEval+, GSM8K, MATH) further supports result reliability (Appendix H.1).

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper specifies the compute resources used, including the TSUBAME supercomputer with NVIDIA H100 GPUs (94GB), distributed training settings (e.g., 64 GPUs, tensor parallelism), and execution times (e.g., 24.7 hours per experiment). Total compute costs are quantified in Appendix F (e.g., 23,700 H100 GPU hours for ablations). See Appendix A.2, A.3.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: The research complies with the NeurIPS Code of Ethics, using permissively licensed datasets (e.g., The-Stack-v2, Finemath-4+) and releasing new assets under the Llama 3.3 Community License. No human subjects, crowdsourcing, or unethical data practices were involved (Sections 3, 4, 1).

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [No]

Justification: The paper focuses on foundational dataset curation for improving LLM performance in code generation and mathematical reasoning, without direct societal applications or immediate risks of misuse. Potential positive impacts, such as advancing automated reasoning and software development, are implied in Section 6, but specific negative impacts (e.g., misuse for malicious code generation) are not discussed, as the work is not tied to deployment or specific applications.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: The datasets (SwallowCode, SwallowMath) and models released do not pose high risks for misuse, as they are curated for Python code and mathematical reasoning data, derived from permissively licensed sources, and released under the Llama 3.3 Community License. No sensitive (e.g., unsafe images) is included, and the paper's focus is on improving pre-training data quality (Sections 3, 4, 1).

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Existing datasets (e.g., The-Stack-v2, Finemath-4+) are credited with citations and their permissive licenses noted (Sections 3, 4, 2). The paper respects the terms of use by processing and releasing derived datasets (SwallowCode, SwallowMath) under compatible licenses (Llama 3.3 Community License).

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: SwallowCode and SwallowMath are thoroughly documented in the paper (Sections 3, 4, 5; Appendix D, E) and in their respective READMEs on HuggingFace (`https://huggingface.co/datasets/tokyotech-llm/swallow-code`, `https://huggingface.co/datasets/tokyotech-llm/swallow-math`), which detail dataset contents and construction pipelines. The code is also documented at `https://github.com/rioyokotalab/swallow-code-math`.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.

- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The research does not involve crowdsourcing or human subjects, focusing solely on computational experiments with publicly available, permissively licensed datasets (Sections 3, 4).

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper involves no human subjects or crowdsourcing, as it focuses on dataset curation and computational experiments (Sections 3, 4).

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The paper describes the use of Llama-3.3-70B-Instruct as a core component of the data rewriting pipelines (SGCR and SCOR for SwallowCode, and math rewriting for SwallowMath) in Sections 3.3, 4.2. The LLM's role in transforming pre-training data is central to the methodology, and its usage, including prompts, is detailed in Appendix D, E.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.

- Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.