Turning the Tide: Repository-based Code Reflection

Wei Zhang[1], Jian Yang[1]*, Jiaxi Yang[2], Ya Wang, Zhoujun Li[1], Zeyu Cui, Binyuan Hui, Junyang Lin

[1]CCSE, Beihang University

[2]Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences

zwpride@buaa.edu.cn

https://LiveRepoReflection.github.io

Abstract

Code large language models (LLMs) enhance programming by understanding and generating code across languages, offering intelligent feedback, bug detection, and code updates through reflection, improving development efficiency and accessibility. While benchmarks (e.g. HumanEval/LiveCodeBench) evaluate code generation and real-world relevance, previous works ignore the scenario of modifying code in repositories. Considering challenges remaining in improving reflection capabilities and avoiding data contamination in dynamic benchmarks, we introduce LiveRepoReflection, a challenging benchmark for evaluating code understanding and generation in multi-file repository contexts, featuring 1,888 rigorously filtered test cases across 6 programming languages to ensure diversity, correctness, and high difficulty. Further, we create RepoReflection-Instruct, a large-scale, quality-filtered instruction-tuning dataset derived from diverse sources, used to train RepoReflectionCoder through a two-turn dialogue process involving code generation and error-driven repair. The leaderboard evaluates over 40 LLMs to reflect the model performance of repository-based code reflection.

1 Introduction

Code large language models (LLMs) (Hui et al., 2024; Yang et al., 2024a; Touvron et al., 2023; Anthropic, 2025; OpenAI, 2023) represent a significant advancement in comprehending and producing code across numerous programming languages. Fueled by extensive training on massive code repositories, LLMs empower developers by offering intelligent feedback, identifying potential bugs, and updating code snippets from human instructions. Code reflection refers to the ability of LLMs to examine and modify their previous responses. Using reflection, LLMs can streamline the development process, boost efficiency, and make programming more accessible to a wider number of developers.

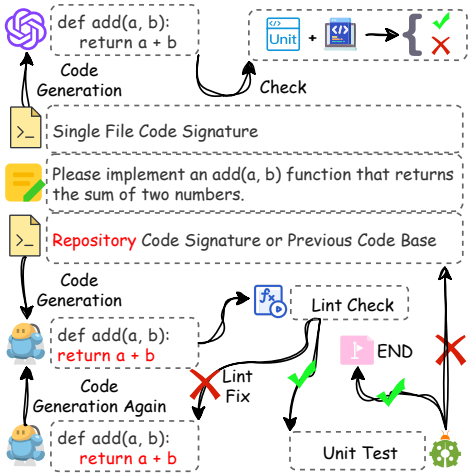

Figure 1: Comparison between general code generation tasks and LiveRepoReflection tasks.

Most previous works primarily focus on generating and evaluating functionally correct code from human instructions illustrated in Figure 1, such as HumanEval/MBPP (Chen et al., 2021; Austin et al., 2021) and MultiPL-E (Cassano et al., 2023). More recent LLMs (e.g. Claude3.7 (Anthropic, 2025) and Qwen3 (Yang et al., 2024a)) and code benchmarks (e.g. LiveCodeBench (Jain et al., 2024), McEval (Chai et al., 2024), and BigCodeBench (Zhuo et al., 2024)) aim to increase the difficulty, complexity, and real-world relevance by introducing more libraries, algorithms, and programming languages. The recently proposed benchmark Aider-polyglot (Gauthier, 2024a) covering 6 languages measures the ability of LLMs to apply code changes to source files without human intervention, reflecting a more realistic development workflow. However, the Aider benchmark is limited by a repository of programming exercises, where the repositories have been included in the pre-training data. There is still a lack of knowledge on how to improve the repository-based code reflection capability of LLMs and build dynamic benchmarks to measure the actual LLM capabilities to avoid data hacking.

*Corresponding author

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 7148–7164
November 4-9, 2025

To explore the LLM capability of repository-based code reflection, we propose an automatic creation pipeline to dynamically update the large-scale instruction corpus and the evaluation benchmark to avoid data hacking. This paper introduces LiveRepoReflection, a challenging benchmark for evaluating code understanding and generation in repository-style multi-file contexts, drawing inspiration from the Aider benchmarks and Exercism problems. To ensure data consistency and facilitate realistic evaluations, LiveRepoReflection comprises coding problems organized with a defined repository structure, including problem definitions, reference answers, code signatures, unit tests, and environment support files. Unlike the potentially redundant data from Exercism, LiveRepoReflection employs an optimized and streamlined file structure. Furthermore, the paper details the construction of RepoReflection-Instruct, a large-scale instruction tuning dataset derived from diverse sources and filtered for quality, which is used to train RepoReflectionCoder, a code-focused large language model. The data generation process for RepoReflection-Instruct involves a sophisticated multi-turn dialogue simulation encompassing code generation, error-driven repair, and style standardization, aiming to enhance the model's ability to handle complex coding tasks and iterative interactions.

The contributions are summarized as follows:

- We propose a high-quality, high-difficulty benchmark for evaluating code reflection of LLMs, featuring 1,888 rigorously filtered test cases across 6 programming languages. The benchmark emphasizes diversity, correctness, and challenge by retaining only cases that stump strong LLMs and undergo human annotation.

- Starting with 500k examples, a strict rejection sampling process ensured high quality by filtering repositories based on criteria such as having unit-test files, reference answer files, aligned code signatures and answers, compatible environment configurations, and standardized file names. The resulting high-quality dataset is used to fine-tune base LLMs to obtain RepoReflectionCoder, enhancing their ability to understand and reason about multi-file codebases with clear dependencies and reproducible environments.

- Our systematic evaluation of over 40+ LLMs on LiveRepoReflection led to the creation of a dynamic leaderboard to track model performance, with extensive experiments demonstrating that LiveRepoReflection effectively measures the alignment between model-generated responses and human preferences.

2 Automatic and Dynamic Pipeline for Repository-based Code Reflection

Repository Code Data File Structure Following Aider Code Edit Benchmark (Gauthier, 2024b) and Aider Polyglot Benchmark (Gauthier, 2024a), we collect 702 coding problems from Exercism[1] for C++, Go, Java, JavaScript, Python, and Rust, available at different repositories[2]. Then we remove the 225 coding problems in the Aider Polyglot Benchmark (Gauthier, 2024a) tests and some erroneous data, and we retain about 473 coding problems to keep the file structure of newly generated code problems the same as these code problems. However, this code data may contain redundancy, as the purpose of these coding problems is to help people complete courses and become proficient in programming languages. Illustrated in Figure 7 in Appendix A, we design a standardized repository code data file structure and streamline the file structure to the minimum size while ensuring normal evaluation compared to them. These data are used to guide the LLMs to generate the repository code file structure format of the newly generated coding problems but do not participate in guiding the content of the new coding problems illustrated in Figure 2.

Seed Code Data To build LiveRepoReflection with high quality and diversity, we collect six programming languages (Python, Java, Go, Rust, C++, JavaScript) code from multiple public sources, like GitHub, Hugging Face, and Reddit.

Multiple Turn Dialogue Data Generation We generate the program topics, definitions, unit tests, then reference answers sequencely and continue each step after the previous dialogue for consistency. To balance diversity and difficulty, a "creative" LLM randomly drawn from a mixed LLM stream produces topics and problem definitions, while multiple "reasoning" LLM, also randomly selected, generate unit tests and reference solutions.

[1] https://exercism.org/
[2] https://github.com/exercism

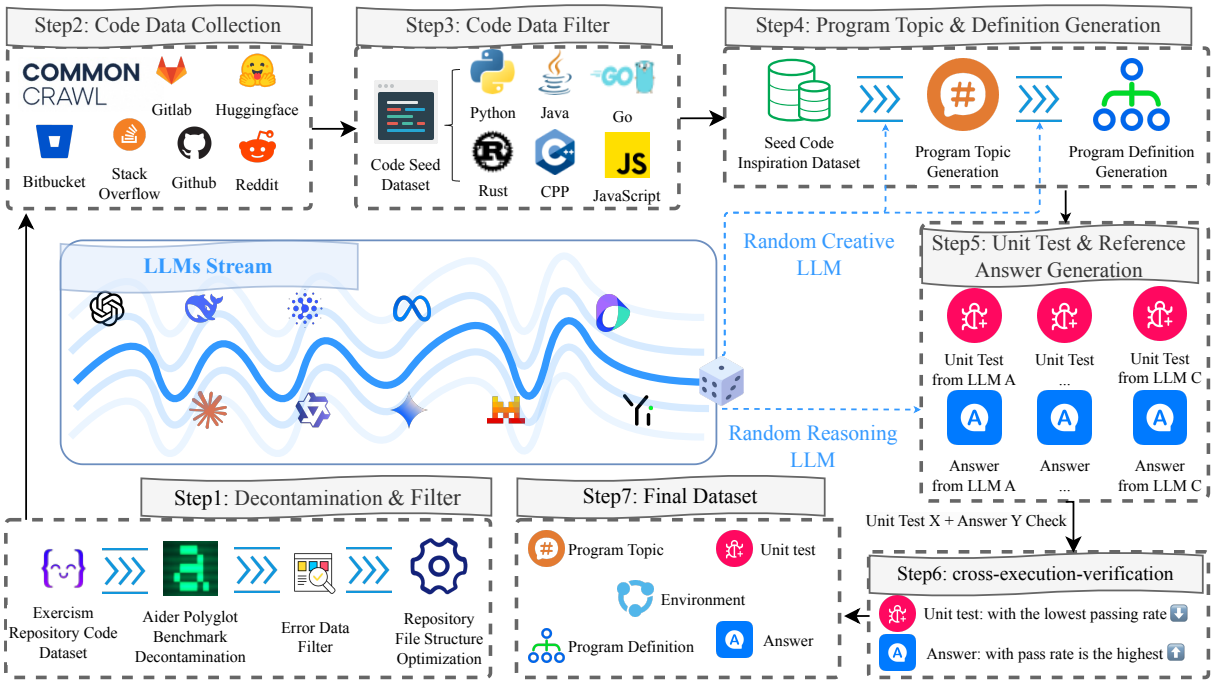

Figure 2: Overview of Construction Pipeline. We use this pipeline to generate polyglot repository code data of LiveRepoReflection and instruction corpus of RepoReflection-Instruct following our designed repository code file structure. 1) Pull Exercism repos, dedupe against other benchmarks, clean out broken samples, and optimize the repository file structure. 2) Collect code snippets from multiple public source like GitHub, Hugging Face and Stack Overflow. 3) Filter data by programming languages. 4) Use a randomized LLM stream to pick a "creative" LLM for generating program topics and definitions from the seed data. 5) Use several "reasoning" LLMs to produce unit tests and reference solutions. 6) Cross execute for verifying every unit test-solution pair, drop anomalies, then retain the test with the lowest pass rate and the solution with the highest and manual review. 7) Package everything into the final repository structure.

Cross-execution Verification We generate one program topic and definition but multiple unit-tests and multiple reference-answers for one coding program. unit-test and reference-answer pairs are sandbox cross-executed, and abnormal samples are dropped. For each program, we keep the unit test with the lowest pass rate and the reference answer with the highest; any 0%–pass cases undergo manual inspection. Each curated coding program are then organized into a repository.

3 LiveRepoReflection Benchmark

Selection of Executable Program Using our automated pipeline, we generate 100K coding program repository cases. We run them in a sandbox, including environment setup, compilation and testing, discarding any case that all LLMs can passe unit tests for too easy or exceeded 180s running time for too slow. Finally, we minimized same topic overlap per language to maximize diversity and retained 10K high-difficulty, high-correctness cases.

Selection of Difficult Problems For high correctness and difficulty of LiveRepoReflection, we generate the code signature based on the reference answer to organize coding program cases into a test repository following the file structure in Figure 7. Then 10 selected mainstream strong reasoning LLMs will write the answer and each LLM has one chance to modify its answer if its initial answer not pass the unit test for some error. For each code program case, we collect the results of 10 LLMs and the result can be "success", "failure-success" and "failure-failure".

If all 10 LLMs get "success", we think the code program case is easy and will discard it. If all 10 LLMs get "failure-success" or several model get "failure-failure", but more than half of the LLMs can still complete the task, we think this part of these code program cases has a certain degree of difficulty but cannot test all LLMs. Therefore, we will keep some data to test weaker LLMs. If more than half of the LLMs only get "failure-failure" but there are still LLMs that can complete it, we think these code program cases have a high degree of difficulty and are more suitable for evaluating most

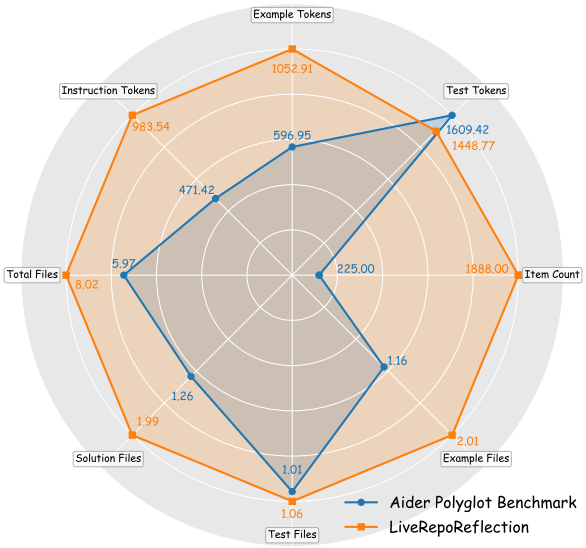

Figure 3: Comparison of dataset scales and structures between Aider Polyglot Benchmark (blue) and LiveRepoReflection (orange). We display eight metrics: (1) problem count; (2) average tokens in test suites; (3) average example-context tokens; (4) average instruction tokens; and (5–8) average files per repository (total, solution, test, example). These metrics highlight that LiveRepoReflection substantially exceeds Aider Polyglot Benchmark in problem coverage, contextual richness, and real-world, multi-file layout complexity.

LLMs. If there are some code program cases that all 10 LLMs get "failure-failure", we will keep them and mark them with special marks to focus on checking in the subsequent stages. The data that passes the inspection will be considered the most difficult part of the entire evaluation data. Finally, we discard nearly 8k data and only keep 2300 code program cases with high quality, high difficulty, high diversity.

Human Annotation To ensure the quality of LiveRepoReflection, we employ 8 graduate students and provide them with a complete code running sandbox environment and ensure the integrity of the data. Each person will complete the annotation of nearly 300 code program cases with the assistance of LLMs. The inspection tasks for each code program case are to check the rationality of the code program case, check the code environment configuration file, check the code file structure, check the reference answer and unit test. In the end, we retained 1,888 test code program cases as the final version of the test data.

Dataset Statistics Figure 3 presents a head-to-head comparison of LiveRepoReflection (orange) against the Aider Polyglot Benchmark (blue) over

full-file code generation: the entire file, including changes, is wrapped in a markdown code block.

Here is the updated copy of your file demo.py:

````
demo.py
```python
def main():
    print("goodbye")
```
````

patch-based incremental edits: specify a file name and the ORIGINAL and UPDATED code blocks.

Here are the changes you requested to demo.py:

````
demo.py
```python
<<<<<<< ORIGINAL
    print("hello")
=======
    print("goodbye")
>>>>>>> UPDATED
```
````

Figure 4: Two evaluation edit format of LiveRepoReflection.

eight key dimensions. First, LiveRepoReflection comprises 1,888 problems—more than 8× the 225 in Aider Polyglot Benchmark, enabling far broader coverage. In terms of test-suite length, our benchmarks average 1,448.8 tokens versus 1,609.4 in Aider Polyglot Benchmark, reflecting more concise, targeted checks. Example contexts in LiveRepoReflection average 1,052.9 tokens compared to 596.9 (≈1.8×), and instructional contexts average 983.5 tokens versus 471.4 (≈2.1×), underscoring richer problem descriptions. Structurally, each LiveRepoReflection repository contains an average of 8.02 files (vs. 5.97), including 2.00 solution files (vs. 1.26), 1.06 test files (vs. 1.01), and 2.01 example files (vs. 1.16). This expand, multi-file layout more faithfully mirrors real-world codebases and challenges models to navigate complex project structures. Overall, LiveRepoReflection delivers substantially greater scale, depth, and structural complexity, making it a more rigorous and realistic testbed for modern code-generation systems. And we provide language distribution comparison in Table 3 in Appendix B.

4 Training of RepoReflectionCoder

Source Data Creation and Quality-Based Reject Sampling We collect approximately 500,000 code examples (including 400,000 newly automatically and dynamically generated ones) to construct the RepoReflection-Instruct corpus. To ensure data quality, strict rejection sampling was applied to all examples, and only repositories meeting all five of the following criteria were retained as a high-quality fine-tuning dataset: 1) At least one unit test file; 2) At least one reference answer file; 3) The

Model	Size	Python				Java				Cpp				Rust				Go				Javascript				All			
		P1	P2	WF	FW	P1	P2	WF	FW	P1	P2	WF	FW	P1	P2	WF	FW	P1	P2	WF	FW	P1	P2	WF	FW	P1	P2	WF	FW
Closed-Source LLMs																													
Claude-3-5-Haiku-20241022	🔒	11.5	33.7	97.8	65.9	12.8	22.4	96.4	42.9	4.2	13.3	89.5	68.4	1.4	5.7	**100.0**	75.0	2.7	16.9	99.5	83.8	1.7	50.0	95.0	96.6	7.8	24.4	97.5	68.1
Claude-3-5-Sonnet-20240620	🔒	12.1	34.6	98.3	65.1	12.0	23.6	97.6	49.2	3.5	8.4	84.6	58.3	1.4	6.6	99.5	78.6	3.7	16.6	98.3	77.6	1.7	45.0	98.3	96.2	8.1	24.5	97.3	67.0
Claude-3-5-Sonnet-20241022	🔒	14.4	45.7	98.2	68.5	10.8	30.4	99.2	64.5	4.2	14.7	99.3	71.4	2.8	7.5	99.5	62.5	5.7	23.6	95.5	75.8	1.7	53.3	78.3	96.8	9.6	32.6	97.4	70.6
Claude-3-7-Sonnet-20250219	🔒	17.6	51.1	99.9	65.6	**14.0**	**33.2**	**100.0**	57.8	6.3	23.1	98.6	72.7	3.3	6.6	**100.0**	50.0	6.5	25.8	**100.0**	75.0	1.7	80.0	98.3	97.9	11.8	37.1	99.8	68.3
Claude-3-7-Sonnet-20250219-Thinking	🔒	14.1	40.7	99.9	65.3	12.8	25.6	99.2	50.0	4.9	20.3	99.3	75.9	3.3	8.0	**100.0**	58.8	6.0	18.4	99.8	67.6	1.7	43.3	**100.0**	96.0	9.9	28.8	99.7	65.6
Doubao-1-5-Thinking-pro-m-250415	🔒	10.1	33.5	99.0	69.8	2.4	16.4	99.6	85.4	4.2	14.7	97.2	71.4	0.5	3.3	**100.0**	85.7	1.5	6.7	99.5	77.8	16.7	75.0	**100.0**	77.7	5.9	22.0	99.2	73.1
Gemini-2.0-Flash	🔒	5.6	26.6	99.0	78.9	5.6	16.0	96.8	65.0	2.8	8.4	97.2	66.7	1.9	3.3	**100.0**	42.9	0.5	2.7	98.8	81.8	0.0	23.3	96.7	**100**	3.7	16.0	98.6	76.8
Gemini-2.5-Flash-preview-04-17	🔒	1.7	19.1	59.5	91.1	1.6	10.8	82.8	85.2	2.8	16.1	57.3	82.6	2.4	5.2	97.6	54.5	2.7	8.2	80.1	66.7	0.0	10.0	76.7	**100**	2.0	13.6	71.7	85.2
Gemini-2.5-pro-preview-05-06	🔒	0.5	13.5	**100.0**	**96.4**	0.8	6.0	**100.0**	86.7	0.7	4.9	**100.0**	**85.7**	0.0	3.8	**100.0**	100.0	1.5	3.2	99.5	53.8	0.0	5.0	**100.0**	**100**	0.7	8.3	99.9	**91.7**
GPT-4o-mini-2024-07-18	🔒	7.6	22.9	98.3	67.0	2.8	14.0	95.6	80.0	2.8	7.7	85.3	63.6	1.9	2.8	98.6	33.3	0.2	4.7	96.8	**94.7**	41.7	75.0	**100.0**	44.4	5.5	16.1	96.7	66.1
GPT-4o-2024-11-20	🔒	9.1	30.4	99.6	69.9	8.8	21.6	99.6	59.3	3.5	14.7	**100.0**	76.2	2.8	6.6	**100.0**	57.1	2.0	11.2	**100.0**	82.2	46.7	68.3	**100.0**	31.6	7.6	22.5	99.8	66.0
GPT-4.1-2025-04-14	🔒	10.6	45.7	**100.0**	76.8	3.6	27.6	**100.0**	87.0	4.9	26.6	**100.0**	81.6	3.3	9.4	**100.0**	65.0	2.2	14.9	99.8	85.0	28.3	36.7	**100.0**	22.9	7.2	30.9	99.9	76.7
GPT-4.1-mini-2025-04-14	🔒	10.9	38.8	99.9	72.0	4.0	17.6	**100.0**	77.3	7.7	21.7	**100.0**	64.5	4.2	7.1	**100.0**	40.0	4.7	19.1	99.8	75.3	50.0	**85.0**	**100.0**	41.2	8.9	28.4	99.9	68.7
GPT-4.1-nano-2025-04-14	🔒	3.2	12.8	75.7	75.2	0.8	5.2	47.2	84.6	2.1	11.2	53.8	81.2	0.5	0.9	92.9	50.0	0.2	4.0	73.4	93.8	10.0	56.7	65.0	0.8	2.1	9.9	71.4	79.0
GPT-4.5-preview-2025-02-27	🔒	11.1	40.1	**100.0**	72.3	3.2	23.6	**100.0**	86.4	7.0	19.6	**100.0**	64.3	3.8	8.5	**100.0**	55.6	3.2	14.4	**100.0**	77.6	21.7	33.3	**100.0**	34.8	7.6	27.1	**100.0**	72.1
Grok-3	🔒	12.2	40.6	**100.0**	70.0	4.8	22.8	**100.0**	78.9	5.6	15.4	99.3	63.6	3.3	8.0	**100.0**	58.8	4.5	20.1	**100.0**	77.8	0.0	60.0	**100.0**	**100**	7.7	28.9	99.9	73.4
Grok-3-fast	🔒	14.3	42.4	99.9	66.4	4.8	25.2	**100.0**	81.0	5.6	16.1	99.3	65.2	4.7	9.4	**100.0**	50.0	5.7	20.1	99.8	71.6	0.0	50.0	**100.0**	**100**	9.0	29.9	99.8	69.9
Grok-3-mini	🔒	9.9	38.0	96.7	74.0	3.6	22.4	99.6	83.9	9.1	21.7	98.6	58.1	3.3	8.0	**100.0**	58.8	1.5	11.9	99.5	87.5	1.7	28.3	**100.0**	94.0	6.2	25.5	98.3	75.7
Grok-3-mini-fast	🔒	8.9	37.2	96.3	76.1	1.2	24.0	**100.0**	95.0	5.6	19.6	**100.0**	71.4	1.9	7.5	**100.0**	75.0	3.2	14.9	**100.0**	78.3	0.0	23.3	**100.0**	**100**	5.3	25.6	98.4	79.1
o1-mini-2024-09-12	🔒	12.2	41.2	98.4	70.4	2.8	18.0	92.0	84.4	5.6	23.1	82.5	75.8	2.4	4.7	97.6	50.0	4.2	16.9	96.5	75.0	**55.0**	**85.0**	95.0	35.3	9.0	28.9	95.8	68.8
o3-mini-2025-01-31	🔒	18.8	50.9	**100.0**	63.1	4.8	31.6	98.8	84.8	**10.5**	32.9	**100.0**	68.1	4.7	8.5	**100.0**	44.4	4.2	18.9	**100.0**	77.6	26.7	75.0	**100.0**	64.4	11.9	36.1	99.8	67.2
o4-mini-2025-04-16	🔒	**20.4**	**54.9**	99.8	62.9	4.4	**33.2**	**100.0**	86.7	**10.5**	**41.3**	93.0	74.6	**6.6**	**10.8**	**100.0**	39.1	**7.7**	**29.8**	99.3	74.2	45.0	50.0	98.3	10.0	**14.0**	**40.5**	99.2	65.4
Open-Source LLMs																													
Qwen3-0.6B-Instruct Think	0.6B	0.6	1.1	83.2	44.4	0.8	0.8	72.4	0.0	0.0	0.0	66.4	0.0	0.0	0.0	94.8	0.0	0.0	0.0	68.0	0.0	3.3	5.0	68.3	34.0	0.5	0.7	78.1	35.7
Qwen3-0.6B-Instruct Chat	0.6B	0.4	0.5	90.7	25.0	0.4	0.4	72.4	0.0	0.0	0.0	93.7	0.0	0.0	0.0	97.2	0.0	0.0	0.0	93.8	0.0	0.0	0.0	58.3	0.0	0.2	0.3	88.9	20.0
Qwen3-1.7B-Instruct Chat	1.7B	1.2	2.1	94.6	41.2	1.6	1.6	92.0	0.0	0.0	0.0	**100.0**	0.0	0.0	0.0	98.6	0.0	0.0	0.5	99.0	100.0	8.3	21.7	91.7	61.8	1.0	1.9	96.0	47.2
Qwen3-1.7B-Instruct Think	1.7B	2.8	6.8	79.6	58.9	1.2	1.2	50.4	0.0	0.0	0.0	61.5	0.0	0.0	0.0	78.8	0.0	0.5	0.5	62.3	0.0	0.0	6.7	38.3	**100**	1.5	3.4	69.3	56.9
Qwen3-4B-Instruct Chat	4B	4.0	10.9	98.4	62.9	1.2	2.4	**100.0**	50.0	2.1	4.9	**100.0**	57.1	0.0	0.5	**100.0**	**100.0**	0.7	2.5	**100.0**	70.0	5.0	21.7	100.0	90.8	2.4	6.7	99.3	64.3
Qwen3-4B-Instruct Think	4B	3.7	17.8	92.2	79.5	0.8	4.0	72.4	80.0	2.1	7.0	67.8	70.0	0.0	0.5	95.3	**100.0**	1.5	5.0	86.6	70.0	0.0	21.7	78.3	**100**	2.2	10.6	86.4	79.5
Qwen3-8B-Instruct Think	8B	4.4	22.9	92.3	80.9	3.6	8.8	80.0	59.1	1.4	10.5	73.4	**86.7**	0.0	1.9	98.1	**100.0**	1.7	7.9	84.4	78.1	5.0	16.7	81.7	77.0	3.0	14.4	87.9	79.0
Qwen3-8B-Instruct Chat	8B	4.3	15.6	99.9	72.7	9.6	19.2	99.6	50.0	1.4	4.2	**100.0**	66.7	0.0	0.5	**100.0**	**100.0**	0.5	4.0	99.3	**87.5**	20.0	43.3	98.3	53.8	4.0	11.9	99.7	66.7
OpenCoder-8B-Instruct	8B	0.2	0.2	**100.0**	0.0	0.8	0.8	**100.0**	0.0	0.0	0.0	**100.0**	0.0	0.0	0.0	**100.0**	0.0	0.0	0.0	**100.0**	0.0	0.0	0.0	100.0	0.0	0.2	0.2	**100.0**	0.0
Seed-Coder-8B-Instruct	8B	4.4	13.3	91.6	67.0	2.0	11.6	47.2	82.8	2.1	3.5	55.9	40.0	2.4	3.8	98.1	37.5	1.0	3.2	82.1	69.2	1.7	13.3	93.3	87.2	2.9	9.1	81.8	68.6
Qwen3-14B-Instruct Chat	14B	4.8	20.0	99.4	76.2	4.4	15.6	99.2	71.8	3.5	6.3	98.6	44.4	0.5	2.8	99.5	83.3	1.2	8.4	99.0	85.3	1.7	23.3	96.7	92.7	3.3	14.1	99.2	76.7
Qwen3-14B-Instruct Think	14B	4.8	26.0	87.7	81.7	2.0	16.4	71.6	87.8	1.4	10.5	69.2	**86.7**	1.4	3.3	92.9	57.1	3.5	9.2	85.6	62.2	1.7	11.7	71.7	85.5	3.4	16.9	83.8	80.0
Qwen3-30B-A3B-Instruct Think	3/30B	6.3	29.3	91.5	78.3	1.6	15.2	76.8	89.5	2.8	11.9	62.2	76.5	2.4	4.2	95.3	44.4	2.7	12.9	84.1	78.8	13.3	35.0	73.3	62.0	4.4	20.0	85.6	77.7
Qwen3-30B-A3B-Instruct Chat	3/30B	7.7	23.0	98.8	66.7	0.8	13.6	**100.0**	**94.1**	3.5	7.7	99.3	54.5	0.9	0.9	99.5	0.0	1.7	6.9	**100.0**	75.0	8.3	40.0	100.0	79.3	4.4	15.3	99.4	70.8
Qwen2.5-Coder-32B-Instruct	32B	6.6	22.7	98.4	71.0	3.2	10.8	93.6	70.4	2.1	6.3	98.6	66.7	0.9	4.7	98.6	80.0	1.2	6.5	98.5	80.8	1.7	38.3	91.7	95.6	3.9	14.9	97.6	74.0
Qwen3-32B-Instruct Think	32B	5.7	32.0	86.1	82.1	6.8	17.6	70.4	61.4	3.5	18.2	51.7	80.8	1.4	4.7	97.2	70.0	2.7	10.7	78.7	74.4	15.0	38.3	63.3	69.4	4.9	21.6	80.3	77.5
Qwen3-32B-Instruct Chat	32B	7.7	27.9	95.1	72.5	6.4	19.6	89.2	67.3	5.6	14.7	83.2	61.9	1.4	2.4	98.6	40.0	1.7	8.9	93.3	80.6	**28.3**	60.0	85.0	52.8	6.0	19.9	93.1	69.7
QwQ-32B	32B	2.1	21.8	92.6	**90.5**	1.6	8.4	94.4	81.0	3.5	13.3	91.6	73.7	1.4	3.3	98.1	57.1	1.0	5.2	94.8	81.0	0.0	8.3	76.7	**100**	1.7	13.3	93.3	**86.9**
DeepSeek-V3	37/671B	11.1	40.1	99.3	72.3	11.2	29.6	99.6	62.2	6.3	15.4	**100.0**	59.1	3.3	7.1	**100.0**	53.3	2.7	18.4	98.5	85.1	0.0	18.3	96.7	**100**	7.7	27.8	99.2	72.2
DeepSeek-R1	37/671B	9.6	37.6	99.9	74.4	6.4	26.0	99.6	75.4	7.0	**22.4**	98.6	68.8	**3.8**	**9.4**	**100.0**	60.0	**4.5**	17.6	99.3	74.6	0.0	46.7	100.0	**100**	6.9	27.8	99.6	75.0
DeepSeek-V3-250324	37/671B	12.7	39.9	99.8	68.2	**16.0**	**30.8**	**100.0**	48.1	4.9	12.6	**100.0**	61.1	2.4	7.5	**100.0**	68.8	3.7	**19.1**	99.5	80.5	0.0	15.0	100.0	**100**	**9.1**	27.8	99.8	67.4
DeepSeek-R1-Distill-Llama-8B	8B	1.0	2.8	76.2	65.2	0.4	0.8	38.0	50.0	0.0	2.8	42.7	100.0	0.0	0.0	85.8	0.0	0.0	0.2	53.3	100.0	5.0	6.7	25.0	25.0	0.6	1.8	63.2	64.7
DeepSeek-R1-Distill-Qwen-14B	14B	3.9	13.0	94.0	70.1	0.4	1.6	81.6	75.0	0.7	2.8	94.4	75.0	0.0	0.0	97.2	0.0	0.5	1.0	81.4	50.0	0.0	5.0	48.3	100.0	1.9	6.5	88.6	70.5
Llama3.1-8B-Instruct	8B	1.3	4.4	87.2	69.4	1.2	7.6	72.0	84.2	0.7	0.7	63.6	0.0	0.0	0.0	92.9	0.0	0.5	0.5	83.6	0.0	6.7	15.0	70.0	55.6	1.1	3.5	82.7	68.7
RepoReflectionCoder	32B	**13.5**	**40.4**	99.8	66.5	8.4	22.0	99.6	61.8	5.6	15.4	**100.0**	63.6	1.4	3.3	**100.0**	57.1	**4.5**	17.4	**100.0**	74.3	15.0	**78.3**	**100.0**	80.8	9.0	**28.2**	99.8	68.0

Table 1: Main Results for 'full-file code generation'.

number of code signature files matches the number of reference answer files; 4) Environment configuration files correspond with the declared programming language; 5) File naming and extensions are standardized and free of anomalies.

Quality Scoring Mechanism Following reject sampling, we employ LLM as a judge and the following sophisticated scoring checklist to quantitatively assess each code program. The composite score $S(p)$ for a code program p is formulated as a weighted linear combination of five key metrics: $S(p) = \sum_{i \in \{1,2,3,4,5\}} w_i \, S_i$, where $w = (0.3, 0.2, 0.2, 0.15, 0.15)$. The five key points are listed as: (1) $S_1 = S_{exec}$: Executability score derived from unit-test pass rate and the absence of compilation or interpretation errors, reflecting functional correctness. (2) $S_2 = S_{nov}$: Novelty score measuring code similarity against existing repositories and semantic diversity in problem descriptions. (3) $S_3 = S_{dif}$: Difficulty score inferred from low test pass rates and the extent of boundary and edge-case coverage, to select challenging tasks. (4) $S_4 = S_{style}$: Code style score based on static analysis tools (e.g., Black, ESLint) that evaluate adherence to naming conventions, code comments, and formatting. (5) $S_5 = S_{ppl}$: Inverse perplexity on reference answers, indicating code that is more challenging for the model to predict and thus potentially more informative.

LiveRepoReflection Decontamination We split the top 10,000 code problems ranked by the scoring function into character 5-gram fragments, use the MinHash algorithm to generate compact signatures, and then utilize an LSH index to efficiently filter candidate texts exhibiting Jaccard similarity greater than 0.8 with the test set. Finally, we perform exact matching verification to ensure the effective and accurate removal of duplicate or highly similar data, thereby maintaining the purity and high quality of the dataset. A total of 8,702 high-quality training code program cases, strictly decontaminated with the original 1,888 test sets in LiveRepoReflection.

Multi-turn Interaction for RepoReflection-Instruct Generation To achieve diverse multi-turn interaction styles, we simulated 840,839 multi-turn coding dialogues using four top models (claude-3-7-sonnet-20250219, qwen3-235b-a22b-instruct, o1-mini, o4-mini), each run four times in two formats (full-file generation and patch-based edits). Training examples are filtered by task importance as follows: 1) Direct generation (40%): produce a complete solution from a prompt. 2) Error-driven repair (40%): iteratively fix code based on compiler/runtime errors. 3) Style standardization (10%): refactor code to meet linting/style guidelines. 4) Dialogue summarization (10%): condense multi-turn interactions into concise overviews.

Model	Size	Python				Java				Cpp				Rust				Go				Javascript				All			
		P1	P2	WF	FW	P1	P2	WF	FW	P1	P2	WF	FW	P1	P2	WF	FW	P1	P2	WF	FW	P1	P2	WF	FW	P1	P2	WF	FW
Closed-Source LLMs																													
Claude-3-5-Haiku-20241022	🔒	12.0	36.6	98.9	67.3	12.4	25.2	95.6	50.8	2.1	9.9	92.3	78.6	2.4	5.2	99.5	54.5	3.2	16.4	99.5	80.3	1.7	41.7	**100.0**	95.9	8.0	25.4	98.2	68.5
Claude-3-5-Sonnet-20240620	🔒	11.2	35.4	98.8	68.3	11.6	22.4	94.8	48.2	2.8	10.5	95.1	73.3	1.4	5.7	98.6	75.0	4.0	16.1	98.3	75.4	3.3	45.0	**100.0**	92.7	7.7	24.6	97.9	68.6
Claude-3-5-Sonnet-20241022	🔒	13.8	41.5	**99.4**	66.8	14.4	28.8	**99.6**	50.0	2.8	14.0	89.5	80.0	3.3	5.2	**100.0**	36.4	**6.7**	21.6	**99.8**	69.0	1.7	31.7	**100.0**	94.6	10.0	29.1	**98.8**	65.8
Claude-3-7-Sonnet-20250219	🔒	17.4	51.2	97.9	66.0	**14.8**	29.2	95.6	49.3	5.6	22.4	88.1	75.0	3.8	6.6	98.1	42.9	5.7	21.3	98.0	73.3	0.0	70.0	93.3	**100.0**	11.6	35.3	96.8	67.2
Claude-3-7-Sonnet-20250219-Thinking	🔒	14.1	45.2	98.9	68.7	13.2	29.2	**99.6**	54.8	7.0	18.2	95.1	61.5	2.4	8.0	99.1	70.6	4.7	18.9	99.0	75.0	1.7	56.7	98.3	97.0	9.7	31.6	98.7	69.2
Doubao-1-5-Thinking-pro-m-250415	🔒	9.9	30.9	93.4	68.0	2.0	14.4	91.2	86.1	5.3	8.4	58.0	36.4	0.9	1.9	75.9	50.0	0.7	7.2	93.8	89.7	0.0	0.0	90.0	0.0	5.2	17.8	88.6	70.6
Gemini-2.0-Flash	🔒	6.3	23.7	76.7	73.2	4.8	8.0	80.0	40.0	2.1	6.3	74.8	66.7	2.4	3.8	88.2	37.5	0.2	3.2	78.2	92.3	0.0	10.0	81.7	**100.0**	3.9	13.2	78.8	70.8
Gemini-2.5-Flash-preview-04-17	🔒	4.6	22.6	75.1	79.5	3.2	10.8	72.8	70.4	2.8	12.6	79.7	77.8	0.9	5.2	89.2	81.8	2.0	9.4	86.8	78.9	1.7	15.0	83.3	88.7	3.2	15.3	79.5	78.8
Gemini-2.5-pro-preview-05-06	🔒	1.7	18.2	91.6	**90.6**	2.4	11.2	87.6	78.6	2.1	11.9	93.7	82.4	0.9	3.8	90.6	75.0	1.7	6.0	96.0	70.8	0.0	8.3	85.0	**100.0**	1.7	12.2	91.8	**86.1**
GPT-4o-2024-11-20	🔒	10.4	30.5	92.8	66.0	3.2	15.6	77.2	79.5	4.2	11.9	63.6	64.7	1.4	4.7	95.3	70.0	0.7	9.9	85.4	**92.5**	40.0	76.7	80.0	47.8	6.8	21.3	86.8	67.9
GPT-4o-mini-2024-07-18	🔒	7.6	19.8	88.2	61.7	0.4	7.6	80.0	**94.7**	0.7	2.8	58.7	75.0	0.0	0.5	92.5	**100.0**	0.5	2.7	89.6	81.8	26.7	43.3	90.0	38.3	4.3	11.8	85.7	63.2
GPT-4.1-nano-2025-04-14	🔒	0.7	2.8	83.8	73.9	1.6	5.2	83.2	69.2	0.0	2.8	42.7	**100.0**	0.0	0.0	95.3	0.0	0.2	0.7	83.4	66.7	1.7	5.0	81.7	66.0	0.6	2.4	81.7	73.9
GPT-4.1-mini-2025-04-14	🔒	12.2	37.1	92.1	67.1	5.2	21.6	91.2	75.9	4.9	17.5	65.7	72.0	4.7	7.5	96.2	37.5	3.5	16.6	85.6	79.1	**60.0**	**86.7**	98.3	27.4	9.5	27.4	89.2	65.3
GPT-4.1-2025-04-14	🔒	10.9	43.9	94.9	75.3	4.0	25.2	87.2	84.1	6.3	19.6	93.7	67.9	4.7	9.9	94.3	52.4	2.5	12.9	95.8	80.8	11.7	68.3	95.0	82.9	7.2	29.9	93.9	76.1
GPT-4.5-preview-2025-02-27	🔒	10.4	45.7	97.1	77.3	6.0	26.0	88.8	76.9	6.3	25.2	95.8	75.0	4.7	9.4	92.5	50.0	1.7	10.4	95.5	83.3	5.0	58.3	**100.0**	91.4	6.8	30.3	95.1	77.5
Grok-3-mini-fast	🔒	10.6	36.1	91.6	70.6	2.0	27.6	92.4	92.8	7.0	16.8	74.1	58.3	3.3	5.2	97.2	36.4	3.0	11.2	94.8	73.3	1.7	11.7	98.3	85.5	6.5	23.9	91.9	73.0
Grok-3-fast	🔒	16.6	42.8	97.0	61.3	8.4	27.6	96.0	69.6	5.6	16.8	**97.2**	66.7	4.2	9.0	97.6	52.6	5.0	18.9	98.0	73.7	1.7	60.0	**100.0**	97.2	10.3	30.5	97.2	66.1
Grok-3-mini	🔒	11.3	37.8	92.3	70.0	3.2	21.2	92.8	84.9	6.3	18.2	75.5	65.4	1.9	4.7	97.2	60.0	2.7	12.2	96.0	77.6	3.3	21.7	98.3	84.8	6.7	24.4	92.6	72.5
Grok-3	🔒	15.7	45.0	98.0	65.0	6.8	26.4	95.2	74.2	6.3	15.4	95.1	59.1	2.8	7.5	97.6	62.5	5.2	22.1	97.5	76.4	0.0	56.7	**100.0**	**100.0**	9.6	31.6	97.4	69.5
o1-mini-2024-09-12	🔒	9.8	28.7	88.7	66.0	1.2	15.2	84.4	92.1	6.3	15.4	61.5	59.1	1.9	3.8	90.1	50.0	2.2	7.4	92.6	70.0	20.0	45.0	83.3	55.6	64.9	19.1	86.9	67.5
o3-mini-2025-01-31	🔒	18.2	50.5	98.2	64.0	3.2	30.0	95.6	89.3	8.4	28.7	72.7	70.7	4.7	6.6	95.8	28.6	3.5	20.6	97.8	83.1	25.0	71.7	96.7	65.1	11.0	35.5	95.5	69.0
o4-mini-2025-04-16	🔒	**18.4**	**51.8**	95.7	64.5	5.2	**31.2**	94.0	83.3	**11.9**	**30.8**	83.9	61.4	**5.2**	**10.8**	99.1	52.2	6.5	**22.3**	93.5	71.1	45.0	58.3	98.3	22.8	**13.0**	**36.8**	94.6	64.7
Open-Source LLMs																													
Qwen2.5-Coder-32B-Instruct	32B	6.2	19.4	82.0	67.9	3.6	14.8	74.0	75.7	3.5	4.9	34.3	28.6	0.5	1.9	79.7	75.0	2.0	5.7	78.7	65.2	0.0	23.3	28.3	**100.0**	3.9	12.9	74.6	69.7
Seed-Coder-8B-Instruct	8B	5.4	9.4	41.1	42.9	2.8	8.0	31.2	65.0	1.4	2.1	17.5	33.3	0.9	1.4	51.9	33.3	0.5	1.5	46.7	66.7	1.7	3.3	33.3	48.5	3.1	5.9	40.1	47.7
DeepSeek-R1	37/671B	8.9	35.4	96.3	74.8	7.2	22.4	97.2	67.9	5.6	**12.6**	73.4	55.6	2.8	**7.5**	96.2	62.5	3.0	15.1	96.5	80.3	0.0	30.0	100.0	**100.0**	6.2	24.3	94.9	74.5
DeepSeek-V3-250324	37/671B	12.1	**40.4**	98.5	70.1	**14.4**	24.4	98.8	41.0	**6.3**	11.2	93.7	43.8	**3.8**	6.1	98.6	38.5	1.7	16.9	98.8	89.7	0.0	23.3	100.0	**100.0**	8.4	26.6	98.3	68.4
DeepSeek-V3	37/671B	15.4	39.0	99.0	60.6	13.2	**24.8**	99.2	46.8	5.6	11.9	91.6	52.9	3.3	6.1	99.5	46.2	2.0	15.4	98.8	87.1	0.0	28.3	98.3	**100.0**	9.6	26.0	98.5	62.9
QwQ-32B	32B	3.3	20.1	52.8	**83.6**	0.8	2.8	50.8	71.4	0.0	8.4	33.6	**100.0**	0.9	2.4	46.2	60.0	0.7	3.2	62.5	76.9	0.0	0.0	61.7	0.0	1.8	10.7	52.7	**83.2**
Qwen3-1.7B-Instruct Chat	1.7B	0.5	1.0	31.1	50.0	0.4	0.4	27.2	0.0	0.7	0.7	37.1	0.0	0.0	0.0	7.5	0.0	0.0	0.2	30.5	**100.0**	0.0	1.7	65.0	**100.0**	0.3	0.6	29.3	50.0
Qwen3-1.7B-Instruct Think	1.7B	1.5	4.5	52.0	67.6	0.8	0.8	47.2	0.0	0.0	0.0	40.6	0.0	0.0	0.0	65.6	0.0	0.2	0.5	49.1	50.0	0.0	8.3	76.7	**100.0**	0.8	2.4	52.2	67.4
Qwen3-14B-Instruct Chat	14B	6.6	21.6	96.6	69.5	2.4	13.2	91.6	81.8	0.7	5.6	69.2	87.5	0.0	0.5	95.8	**100.0**	2.0	6.2	93.8	68.0	0.0	25.0	83.3	**100.0**	3.7	13.7	92.7	73.4
Qwen3-14B-Instruct Think	14B	6.7	26.3	80.7	74.5	2.8	15.6	73.2	82.1	2.1	11.9	56.6	82.4	0.9	3.3	87.3	71.4	1.5	6.5	72.7	76.9	0.0	6.7	68.3	**100.0**	3.9	16.4	76.5	76.4
Qwen3-30B-A3B-Instruct Think	3/30B	7.2	23.5	75.4	69.4	2.0	7.6	72.4	73.7	1.4	9.1	57.3	84.6	1.4	1.9	57.1	25.0	1.0	6.7	79.2	85.2	11.7	26.7	81.7	56.2	4.2	14.4	72.6	70.6
Qwen3-30B-A3B-Instruct Chat	3/30B	11.1	27.7	96.7	59.9	2.0	10.4	94.8	80.8	3.5	7.0	79.0	50.0	0.0	0.5	91.0	**100.0**	2.5	6.5	94.8	61.5	5.0	25.0	95.0	80.0	6.0	16.2	94.0	62.6
Qwen3-32B-Instruct Think	32B	6.8	29.0	82.7	76.5	4.8	16.4	82.8	70.7	2.8	10.5	58.0	73.3	1.9	3.3	72.2	42.9	2.2	8.7	83.9	74.3	13.3	23.3	75.0	42.9	4.9	18.5	79.7	73.4
Qwen3-32B-Instruct Chat	32B	7.4	23.8	90.5	68.7	5.6	14.8	92.0	62.2	4.9	7.7	84.6	36.4	0.5	2.4	93.9	80.0	1.7	7.4	95.0	76.7	13.3	36.7	95.0	63.8	5.2	15.9	91.7	67.3
Qwen3-4B-Instruct Chat	4B	3.9	11.2	87.9	65.2	0.8	1.2	39.6	33.3	1.4	1.4	62.9	0.0	0.5	0.5	84.0	0.0	0.7	1.2	40.7	40.0	0.0	3.3	45.0	**100.0**	2.1	5.6	67.7	61.9
Qwen3-4B-Instruct Think	4B	4.4	13.8	72.0	68.1	1.6	3.6	65.2	55.6	0.0	2.1	46.9	**100.0**	0.0	0.9	66.0	**100.0**	0.0	1.7	60.5	**100.0**	0.0	11.7	48.3	**100.0**	2.1	7.5	65.3	71.6
Qwen3-8B-Instruct Chat	8B	6.5	17.3	93.0	62.7	8.8	14.8	95.6	40.5	0.7	2.1	76.9	66.7	0.0	0.9	94.8	**100.0**	1.0	4.0	92.6	75.0	**25.0**	46.7	83.3	46.5	5.0	12.1	91.9	58.3
Qwen3-8B-Instruct Think	8B	5.7	19.9	70.9	71.2	2.0	6.0	77.2	66.7	0.7	6.3	53.1	88.9	0.0	0.9	65.1	**100.0**	1.7	4.5	74.2	61.1	1.7	10.0	43.3	83.0	3.2	11.3	69.5	71.4
Qwen3-0.6B-Instruct Chat	0.6B	0.2	0.2	17.9	0.0	0.4	0.4	11.2	0.0	0.0	0.0	21.0	0.0	0.0	0.0	11.8	0.0	0.0	0.0	10.7	0.0	0.0	0.0	10.0	0.0	0.2	0.2	14.8	0.0
Qwen3-0.6B-Instruct Think	0.6B	0.2	0.2	12.3	0.0	0.8	0.8	4.0	0.0	0.0	0.0	13.3	0.0	0.0	0.0	16.0	0.0	0.0	0.0	8.7	0.0	0.0	0.0	3.3	0.0	0.2	0.2	10.6	0.0
OpenCoder-8B-Instruct	8B	0.2	0.2	**100.0**	0.0	0.8	0.8	**100.0**	0.0	0.0	0.0	**100.0**	0.0	0.0	0.0	**100.0**	0.0	0.0	0.0	**100.0**	0.0	0.0	0.0	**100.0**	0.0	0.2	0.2	**100.0**	0.0
DeepSeek-R1-Distill-Llama-8B	8B	1.0	2.8	72.2	65.2	0.0	0.4	74.8	100.0	0.0	0.0	24.5	0.0	0.0	0.0	70.8	0.0	0.2	0.2	52.9	0.0	28.3	30.0	78.3	5.6	1.4	2.3	64.8	39.5
DeepSeek-R1-Distill-Qwen-14B	14B	4.1	11.2	91.7	63.0	0.4	2.0	91.2	80.0	0.7	2.1	39.2	66.7	0.0	0.0	94.3	0.0	0.2	0.7	81.6	66.7	1.7	5.0	91.7	66.7	2.0	5.6	85.8	64.2
Llama3.1-8B-Instruct	8B	0.2	0.2	2.6	0.0	0.8	0.8	4.8	0.0	0.0	0.0	3.5	0.0	0.0	0.0	5.2	0.0	0.0	0.0	4.2	0.0	0.0	0.0	6.7	0.0	0.2	0.2	3.7	0.0
RepoReflectionCoder	32B	**15.7**	37.3	96.2	57.8	12.0	22.0	92.0	45.5	4.9	10.5	81.8	53.3	2.8	4.7	96.7	40.0	**5.2**	**18.1**	95.8	71.2	10.0	**85.0**	100.0	88.2	**10.5**	**27.0**	94.7	61.0

Table 2: Main Results for 'patch-based incremental edits'.

5 Experiments

5.1 Implementation Details

We fine-tune Qwen2.5-Coder-32B on 128 NVIDIA H800-80GB GPUs[3] using the RepoReflection-Instruct dataset and a set of roughly 500 K repositories, employing Adam with a cosine-decay scheduler: after 100 warm-up steps the learning rate peaks at 5e-5, a global batch size of 1,024, and inputs truncated to 32,768 tokens. In a second stage, we further train RepoReflectionCoder on 150,000 simulated multi-turn dialogue trajectories—covering code synthesis, iterative debugging, and style standardization—to sharpen its practical generative and reasoning skills in realistic coding scenarios.

5.2 Evaluation Metrics

Inspired by the aider polyglot benchmark setting, we evaluate LLMs with four key metrics:

Pass@1. Measures the proportion of coding tasks an LLM completes correctly on its first attempt, as verified by test cases (Zheng et al., 2023), directly reflecting the one-shot coding accuracy of LLMs.

Pass@2. After a failed attempt, LLMs can view their previous code and error messages before trying again, capturing the capacity of LLMs to improve via immediate feedback.

[3] https://github.com/QwenLM/Qwen2.5-Coder/tree/main/finetuning/sft

Fix Weight (FW). Defined as $\text{FW} = \frac{\text{Pass@2}-\text{Pass@1}}{\text{Pass@2}}$, it represents the fraction of successful second-attempt fixes among all second-attempt successes, emphasizing the relative contribution of LLM through error diagnosis and correction.

Well Format (WF). Measures the percentage of tasks where the LLM strictly follows the edit format specified in the system prompt, quantifying format compliance under constrained instructions.

5.3 Evaluation Edit Format

Figure 4 shows two different "edit formats" when evaluating different LLMs. The "full-file code generation" format is the easiest for an LLM to use, but it uses a lot of tokens and may limit how large a file can be edited. Models which can use one of the "patch-based incremental edits" formats are much more efficient, using far fewer tokens. Models that use a diff-like format are able to edit larger files with less cost and without hitting token limits. For fair comparison, we will show the scores in "full-file code generation" and "patch-based incremental edits" format to fully evaluate all LLMs.

5.4 Code LLMs

We evaluate 40+ LLMs, including GPT-4.1/GPT-4.5 (OpenAI, 2025), Claude-3.5/3.7 (Anthropic, 2023, 2025), o1-mini/o4-mini (Jaech et al., 2024), Gemini (Anil et al.), Qwen2.5-Coder/Qwen3/Qwq-32B (Hui et al., 2024; Yang et al., 2024a; Qwen, 2025), Grok (xAI, 2025), OpenCoder (Huang

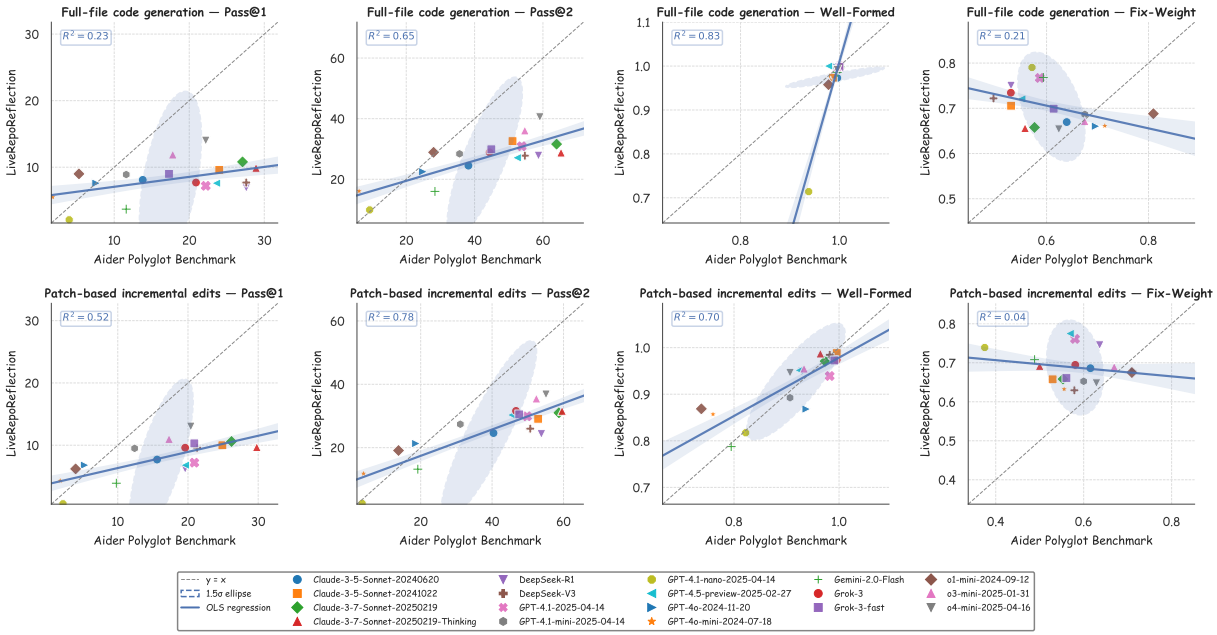

Figure 5: Performance comparison between LiveRepoReflection and the Aider Polyglot Benchmark over multiple large language models. Scatter points show individual model scores under full-file code generation and patch-based incremental edits across four metrics (Pass@1, Pass@2, Well-Formed, Fix-Weight). The dashed diagonal line denotes $y = x$, ellipses represent 1.5σ confidence regions, and solid lines are ordinary least squares fits annotated with their R^2 values.

et al., 2025), DeepSeek-R1/V3 (Guo et al., 2025; DeepSeek-AI et al., 2025) and Doubao-1-5-Thinking-pro-m-250415 (Seed et al., 2025). And to compare all LLMs fairly, we use a temperature 0, max_output_tokens 8192 and no additional custom parameters.

5.5 Main Result

Table 1 and Table 2 show that the results under both patch-based incremental edits and full-file code generation formats show that leading closed-source models consistently achieve the highest one-shot and post-feedback accuracies, while open-source models trail behind but display similar relative gains when allowed a second attempt. All systems find Python tasks easiest and struggle most with C++ and Rust. Nearly every model exceeds ninety percent format-compliance, and fix-weight values around sixty to eighty percent demonstrate that test feedback reliably drives improvements. Incremental patch edits narrow the gap between first and second passes compared with full-file generation. Our RepoReflectionCoder clearly outperforms its Qwen2.5-Coder base yet still falls short of the top closed-source performers, highlighting room for further advances in multi-file code reflection.

6 Analysis

6.1 Performance Comparison between LiveRepoReflection and Aider Polyglot Benchmark

Figure 5 reveals that while pass@1 and pass@2 scores on LiveRepoReflection correlate moderately with the older aider polyglot benchmark ($R^2 \approx 0.65$ for pass@1 and 0.78 for pass@2), nearly all points fall below the $y = x$ line, indicating consistently lower absolute pass rates on LiveRepoReflection and thus greater task difficulty. well-formedness remains highly consistent between the two datasets ($R^2 \approx 0.83$ full-file, 0.70 diff-based), showing that syntactic and structural constraints are enforced similarly. In stark contrast, fix-weight exhibits very weak alignment ($R^2 \approx 0.21$ and 0.04), underscoring divergent repair performance and demonstrating that LiveRepoReflection more reliably challenges and discriminates model debugging capabilities. Moreover, because the aider polyglot benchmark has been publicly available since mid-2024 and may have been subject to overfitting by recent model releases, our freshly curated LiveRepoReflection, compiled from never-before-seen repositories through May 2025, provides a more robust and less biased evaluation of real-world code generation and repair.

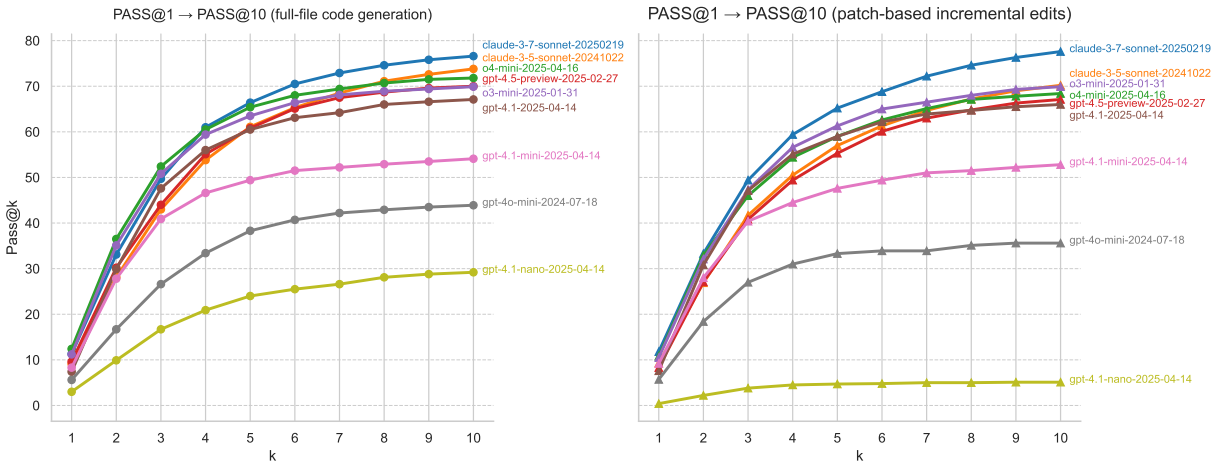

Figure 6: Pass@k (k=1–10) curves for nine LLMs under full-file code generation (left) and patch-based incremental editing (right). All models exhibit strictly increasing performance with diminishing marginal gains beyond k=2; patch-based editing yields more uniform improvements across samples, whereas full-file generation achieves higher absolute pass rates.

6.2 Pass@k Curves for Two Editing Formats

We evaluate 9 representative LLMs under two edit formats (full-file code generation and patch-based incremental edits) by measuring pass@k ($k = 1 - > 10$). Figure 6 illustrates pass@k of each LLM curve is strictly increasing yet exhibits pronounced diminishing returns: the largest marginal gain occurs at k=1→2, and thereafter each additional sample yields progressively smaller improvements. Although full-file code generation attains higher absolute pass rates, patch-based incremental edits produces notably more uniform gains across k, suggesting that iterative feedback helps smaller improvements stack more evenly. Some LLMs (e.g. o3-mini, claude-3-5-sonnet) enjoy larger early gains but plateau by k≂5–7, and other LLMs (e.g. o4-mini, gpt-4.1) start with high pass@1 and maintain smoother, lower-amplitude increments. The overall trends of all LLMs are consistent. As the number of iterations k increases and the feedback information increases, pass@k gradually increases, and the rate of increase gradually slows down. In addition, weaker LLMs (such as gpt4.1-nano-2025-04-14 and gpt4o-mini-2024-07-18) perform worse than full-file code generation in patch-based incremental edits, but there is no significant difference in stronger LLMs.

7 Related Work

Code Large Language Model. Leveraging advancements in NLP, pre-training techniques have significantly bolstered code understanding and synthesis in models like CodeBERT (Feng et al., 2020) and CodeT5 (Wang et al., 2021), leading to the adoption of NLP-inspired architectures and objectives for tasks such as code generation, infilling, summarization, refinement, and translation (Lu et al., 2021; Yan et al., 2023; Liu et al., 2023; Xie et al., 2023). The emergence of code-specific large language models (LLMs) (Li et al., 2023; Rozière et al., 2023; Guo et al., 2024a; Yang et al., 2024b,c), exemplified by CodeGen (Nijkamp et al., 2023) and Code Llama (Rozière et al., 2023), demonstrates foundational competence in code understanding and generation. Inspired by multi-agent collaboration (Guo et al., 2024b; Wang et al., 2023a), the concept of language-specific agents is introduced to create multilingual instruction datasets, building upon the success of instruction tuning (Ouyang et al., 2022; Zhang et al., 2023; Wang et al., 2023b) and innovations like code Evol-Instruct (Luo et al., 2023) and the utilization of real-world code in OSS-Instruct (Wei et al., 2023) and CodeOcean (Yu et al., 2023) to enhance instruction data quality and realism, with multilingual benchmarks (Cassano et al., 2023; Chai et al., 2024; Liu et al., 2024a,b; Zhuo et al., 2024) assessing these models' capabilities.

Code Reflection For code intelligence, repository-based code benchmarks such as SWEBench (Jimenez et al., 2024; Miserendino et al., 2025), Aider-polyglot (Aider Team), and ExecRepoBench (Yang et al., 2024c) evaluate the abilities of addressing real-world software engineering tasks, code reflection, and repository-

based code completion. DeepSeek-R1 (Guo et al., 2025) demonstrates capabilities such as reflection capabilities in long CoTs, marking a significant milestone for the research community. Reflectioncoder (Ren et al., 2024).

8 Conclusion

This paper proposes LiveRepoReflection, a high-difficulty code reflection benchmark for multi-file repository scenarios. By combining automated pipelines with manual verification, we built a total of 1,888 strictly screened test cases covering six programming languages. Based on multi-source high-quality corpora and multi-round dialogue generation strategies, we built the RepoReflection-Instruct instruction set and trained RepoReflectionCoder, achieving significant performance improvements. Experimental results show that LiveRepoReflection can truly and effectively measure the model's reflection and repair capabilities in cross-file dependency and iterative repair scenarios, providing a solid foundation for subsequent research.

9 Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant Nos. 62276017, 62406033, U1636211, 61672081), and the State Key Laboratory of Complex & Critical Software Environment (Grant No. SKLCCSE-2024ZX-18).

10 Limitations

We acknowledge the following limitations of this study: (1) The evaluation in repository-level multilingual scenarios are not fully explored. (2) The code completion model RepoReflectionCoder is mainly supervised fine-tuned on the 32B open-source base LLMs. In the future, we will try the (3) The fine-tuned model can be further improved using RLHF for better user experience, such as DPO.

Ethics Statement

This research adheres to ethical guidelines for AI development. We aim to enhance the capabilities of large language models (LLMs) while acknowledging potential risks such as bias, misuse, and privacy concerns. To mitigate these, we advocate for transparency, rigorous bias testing, robust security measures, and human oversight in AI applications. Our goal is to contribute positively to the field and to encourage responsible AI development and deployment.

References

Aider Team. Aider llm leaderboards.

Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: A family of highly capable multimodal models. corr, abs/2312.11805, 2023. doi: 10.48550. *arXiv preprint ARXIV.2312.11805*, pages 24–28.

Anthropic. 2023. Claude 2. Technical report, Anthropic.

Anthropic. 2025. Claude 3.7 sonnet and claude code.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. 2023. Multipl-e: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 49(7):3675–3691.

Linzheng Chai, Shukai Liu, Jian Yang, Yuwei Yin, Ke Jin, Jiaheng Liu, Tao Sun, Ge Zhang, Changyu Ren, Hongcheng Guo, et al. 2024. Mceval: Massively multilingual code evaluation. *arXiv preprint arXiv:2406.07436*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison 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, Joshua 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. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, abs/2107.03374.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. 2025. Deepseek-v3 technical report.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 1536–1547. Association for Computational Linguistics.

Paul Gauthier. 2024a. GPT code editing benchmarks — aider.chat. https://aider.chat/docs/leaderboards/#polyglot-leaderboard. [Accessed 28-01-2025].

Paul Gauthier. 2024b. GPT code editing benchmarks — aider.chat. https://aider.chat/docs/benchmarks.html#the-benchmark. [Accessed 21-01-2025].

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024a. Deepseek-coder: When the large language model meets programming – the rise of code intelligence.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. 2024b. Large language model based multi-agents: A survey of progress and challenges. *CoRR*, abs/2402.01680.

Siming Huang, Tianhao Cheng, J. K. Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, Jiaheng Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. 2025. Opencoder: The open cookbook for top-tier code large language models.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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 V, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Moustafa-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. 2023. Starcoder: may the source be with you! *CoRR*, abs/2305.06161.

Shukai Liu, Linzheng Chai, Jian Yang, Jiajun Shi, He Zhu, Liran Wang, Ke Jin, Wei Zhang, Hualei Zhu, Shuyue Guo, et al. 2024a. Mdeval: Massively multilingual code debugging. *arXiv preprint arXiv:2411.02310*.

Siyao Liu, He Zhu, Jerry Liu, Shulin Xin, Aoyan Li, Rui Long, Li Chen, Jack Yang, Jinxiang Xia, ZY Peng, et al. 2024b. Fullstack bench: Evaluating llms as full stack coder. *arXiv preprint arXiv:2412.00535*.

Yue Liu, Thanh Le-Cong, Ratnadira Widyasari, Chakkrit Tantithamthavorn, Li Li, Xuan-Bach Dinh Le, and David Lo. 2023. Refining chatgpt-generated code: Characterizing and mitigating code quality issues. *CoRR*, abs/2307.12596.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct. *CoRR*, abs/2306.08568.

Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. 2025. Swe-lancer: Can frontier llms earn $1 million from real-world freelance software engineering? *arXiv preprint arXiv:2502.12115*.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. Codegen: An open large language model for code with multi-turn program synthesis. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

OpenAI. 2025. Introducing gpt-4.5.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Qwen. 2025. Qwq-32b: Embracing the power of reinforcement learning.

Houxing Ren, Mingjie Zhan, Zhongyuan Wu, Aojun Zhou, Junting Pan, and Hongsheng Li. 2024. Reflectioncoder: Learning from reflection sequence for enhanced one-off code generation.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950.

ByteDance Seed, :, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiangpeng Wei, Wenyuan Xu, Yufeng Yuan, Yu Yue, Lin Yan, Qiying Yu, Xiaochen Zuo, Chi Zhang, Ruofei Zhu, Zhecheng An, Zhihao Bai, Yu Bao, Xingyan Bin, Jiangjie Chen, Feng Chen, Hongmin Chen, Riwei Chen, Liangqiang Chen, Zixin Chen, Jinsong Chen, Siyan Chen, Kaiyuan Chen, Zhi Chen, Jin Chen, Jiecao Chen, Jinxin Chi, Weinan Dai, Ning Dai, Jiahui Dai, Shihan Dou, Yantao Du, Zhengyin Du, Jianhui Duan, Chen Dun, Ting-Han Fan, Jiazhan Feng, Junda Feng, Ziyuan Feng, Yuwei Fu, Wenqi Fu, Hanjie Fu, Hao Ge, Hongyi Guo, Mingji Han, Li Han, Wenhao Hao, Xintong Hao, Qianyu He, Jerry He, Feng He, Wen Heng, Zehua Hong, Qi Hou, Liang Hu, Shengding Hu, Nan Hu, Kai Hua, Qi Huang, Ziyue Huang, Hongzhi

Huang, Zihao Huang, Ting Huang, Wenhao Huang, Wei Jia, Bin Jia, Xiaoying Jia, Yuhua Jiang, Haobin Jiang, Ziheng Jiang, Kaihua Jiang, Chengquan Jiang, Jianpeng Jiao, Xiaoran Jin, Xing Jin, Xunhao Lai, Zheng Li, Xiang Li, Liyi Li, Hongkai Li, Zheng Li, Shengxian Wan, Ya Wang, Yunshui Li, Chenggang Li, Niuniu Li, Siyu Li, Xi Li, Xiao Li, Aoyan Li, Yuntao Li, Nianning Liang, Xinnian Liang, Haibin Lin, Weijian Lin, Ye Lin, Zhicheng Liu, Guanlin Liu, Guanlin Liu, Chenxiao Liu, Yan Liu, Gaohong Liu, Juncai Liu, Chundian Liu, Deyi Liu, Kaibo Liu, Siyao Liu, Qi Liu, Yongfei Liu, Kang Liu, Gan Liu, Boyi Liu, Rui Long, Weiqiang Lou, Chenwei Lou, Xiang Luo, Yao Luo, Caiping Lv, Heyang Lv, Bole Ma, Qianli Ma, Hongzhi Ma, Yiyuan Ma, Jin Ma, Wenchang Ma, Tingting Ma, Chen Mao, Qiyang Min, Zhe Nan, Guanghan Ning, Jinxiang Ou, Haojie Pan, Renming Pang, Yanghua Peng, Tao Peng, Lihua Qian, Lihua Qian, Mu Qiao, Meng Qu, Cheng Ren, Hongbin Ren, Yong Shan, Wei Shen, Ke Shen, Kai Shen, Guangming Sheng, Jinlong Shi, Wenlei Shi, Guang Shi, Shuai Shuai Cao, Yuxin Song, Zuquan Song, Jing Su, Yifan Sun, Tao Sun, Zewei Sun, Borui Wan, Zihan Wang, Xiaohui Wang, Xi Wang, Shuguang Wang, Jun Wang, Qinlong Wang, Chenyuan Wang, Shuai Wang, Zihan Wang, Changbao Wang, Jiaqiang Wang, Shihang Wang, Xuwu Wang, Zaiyuan Wang, Yuxuan Wang, Wenqi Wang, Taiqing Wang, Chengzhi Wei, Houmin Wei, Ziyun Wei, Shufa Wei, Zheng Wu, Yonghui Wu, Yangjun Wu, Bohong Wu, Shuang Wu, Jingqiao Wu, Ning Wu, Shuangzhi Wu, Jianmin Wu, Chenguang Xi, Fan Xia, Yuqiao Xian, Liang Xiang, Boren Xiang, Bowen Xiao, Zhen Xiao, Xia Xiao, Yongsheng Xiao, Chao Xin, Shulin Xin, Yuwen Xiong, Jingjing Xu, Ziwen Xu, Chenyin Xu, Jiayi Xu, Yifan Xu, Wei Xu, Yufei Xu, Shikun Xu, Shipeng Yan, Shen Yan, Qingping Yang, Xi Yang, Tianhao Yang, Yuehang Yang, Yuan Yang, Ximing Yang, Zeyu Yang, Guang Yang, Yifan Yang, Xuesong Yao, Bairen Yi, Fan Yin, Jianian Yin, Ziqiang Ying, Xiangyu Yu, Hongli Yu, Song Yu, Menghan Yu, Huan Yu, Siyu Yuan, Jun Yuan, Yutao Zeng, Tianyang Zhan, Zheng Zhang, Yun Zhang, Mofan Zhang, Wang Zhang, Ru Zhang, Zhi Zhang, Tianqi Zhang, Xinyi Zhang, Zhexi Zhang, Sijun Zhang, Wenqiang Zhang, Xiangxiang Zhang, Yongtao Zhang, Yuyu Zhang, Ge Zhang, He Zhang, Yue Zhang, Renjie Zheng, Ningxin Zheng, Zhuolin Zheng, Yaowei Zheng, Chen Zheng, Xiaoyun Zhi, Wanjun Zhong, Cheng Zhong, Zheng Zhong, Baoquan Zhong, Xun Zhou, Na Zhou, Huan Zhou, Hang Zhu, Defa Zhu, Wenjia Zhu, and Lei Zuo. 2025. Seed1.5-thinking: Advancing superb reasoning models with reinforcement learning.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. 2023a. A survey on large language model based autonomous agents. *CoRR*, abs/2308.11432.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023b. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13484–13508. Association for Computational Linguistics.

Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 8696–8708. Association for Computational Linguistics.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2023. Magicoder: Source code is all you need. *CoRR*, abs/2312.02120.

xAI. 2025. Grok 3 beta — the age of reasoning agents.

Zhuokui Xie, Yinghao Chen, Chen Zhi, Shuiguang Deng, and Jianwei Yin. 2023. Chatunitest: a chatgpt-based automated unit test generation tool. *CoRR*, abs/2305.04764.

Weixiang Yan, Yuchen Tian, Yunzhe Li, Qian Chen, and Wen Wang. 2023. Codetransocean: A comprehensive multilingual benchmark for code translation. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 5067–5089. Association for Computational Linguistics.

An Yang, feng Li, and etc. 2024a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Jian Yang, Jiaxi Yang, Ke Jin, Yibo Miao, Lei Zhang, Liqun Yang, Zeyu Cui, Yichang Zhang, Binyuan Hui, and Junyang Lin. 2024b. Evaluating and aligning codellms on human preference. *arXiv preprint arXiv:2412.05210*.

Jian Yang, Jiajun Zhang, Jiaxi Yang, Ke Jin, Lei Zhang, Qiyao Peng, Ken Deng, Yibo Miao, Tianyu Liu, Zeyu Cui, et al. 2024c. Execrepobench: Multi-level executable code completion evaluation. *arXiv preprint arXiv:2412.11990*.

Zhaojian Yu, Xin Zhang, Ning Shang, Yangyu Huang, Can Xu, Yishujie Zhao, Wenxiang Hu, and Qiufeng Yin. 2023. Wavecoder: Widespread and versatile enhanced instruction tuning with refined data generation. *CoRR*, abs/2312.14187.

Renrui Zhang, Jiaming Han, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, Peng Gao, and Yu Qiao. 2023. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *CoRR*, abs/2303.16199.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. 2023. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. *arXiv preprint arXiv:2303.17568*, abs/2303.17568.

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. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*.

A Polyglot Repository Code File Structure Examples

We provide polyglot repository code file structure examples of LiveRepoReflection and instruction corpus of RepoReflection-Instruct in Figure 7.

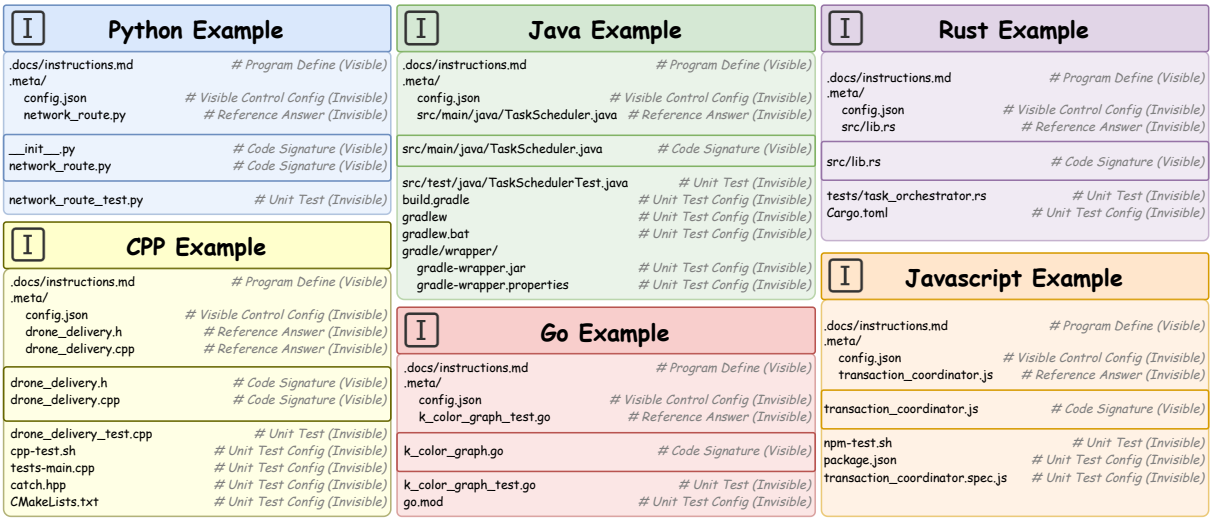

Figure 7: Polyglot repository code file structure examples of LiveRepoReflection and instruction corpus of RepoReflection-Instruct. Visibility ("Visiable" or "Invisiable") indicates whether the model is allowed to obtain and modify the file content during evaluation. The repository code file structure consists of five parts: 1) problem definition, 2) reference answer, 3) code signature, 4) unit test, 5) unit test environment support file.

B Language Distribution Comparison

Table 3 compares the number of test cases per programming language in LiveRepoReflection versus the Aider Polyglot Benchmark. LiveRepoReflection contains over eight times as many test cases overall (1,888 vs. 225) and expands coverage in every language. Python, as a dominant scripting language, grows from 34 to 820 cases, while compiled languages like Go (39→403), Java (47→250), C++ (26→143), and Rust (30→212) also see substantial increases. JavaScript remains smaller in absolute terms but still benefits from a slight boost (49→60). This balanced, large-scale distribution across both scripting and system languages ensures that models are evaluated on a wide spectrum of real-world coding tasks and paradigms.

Table 3: Language Distribution Comparison

Language	LiveRepoReflection	Aider Polyglot Benchmark
C++	143	26
Go	403	39
Java	250	47
JavaScript	60	49
Python	820	34
Rust	212	30
Total	1,888	225

C Prompts for Our Pipeline, LiveRepoReflection and RepoReflection-Instruct

We provided data generation stage prompts for our pipeline, LiveRepoReflection and RepoReflection-Instruct in Figure 8, Figure 9, Figure 10, Figure 11, Figure 12, Figure 13, Figure 14 and Figure 15.

Act as a high-level programming competition question setter and take requests for generating a new code problem.

1. Make sure your code problem concise but complete.
2. Make sure your code problem difficult and challenging.

Figure 8: System Prompt for Our Pipeline.

When Creating files, maintain a consistent folder structure as shown in the examples by providing the appropriate path/to/filename and adhere to the following format:

path/to/filename
```
// entire code or file content ...
```

- The first line should contain *only* the appropriate path/to/filename, without any additional markup, punctuation, comments, or other elements.
- The second line should start with three backticks (```)
- ... include the complete content of the file ...
- The last line should end with three closing backticks (```)

Please ensure that you *never* skip, omit, or abbreviate content using ellipsis (...) or by adding comments like "... rest of code...". Use only standard libraries in your code.

Figure 9: Format Reminder Prompt for Our Pipeline.

Here are some question examples to give you inspiration:

Question Example {index_placeholder}:

Project Name

{project_name}

Question Description

{question_description}

Answer File With Dependencies

{answer}

Unit Test File

{unit_test}

Figure 10: Sample Data Template Prompt for Our Pipeline.

Please generate a challenging and sophisticated {language} coding problem. Consider incorporating these elements to increase complexity from question example inspiration:

- Advanced data structures (trees, graphs, heaps)
- Multiple edge cases and constraints
- Optimization requirements
- Real-world practical scenarios
- System design aspects
- Algorithmic efficiency requirements
- Multiple valid approaches with different trade-offs

Make sure the difficulty is as high as possible, similar to the leetcode Hard level.

Please **only** describe the question clearly but challenge the solver with interesting constraints and requirements. Do not include any project name, code signature, answer, unit test or any other things at this stage.

{sample_data_str}

Now, begin!

Figure 11: Coding Program Definition Prompt for Our Pipeline.

Now, present the name of this {language} problem with snake case and keep it short and concise by using 1-3 words, like “hello_world”. Please generate the name in the following format:

“‘ project_name “‘

Please **only** generate the name in the above format. Do not include any question description, code signature, answer, unit test or any other things at this stage. Now, begin!

Figure 12: Coding Program Topic for Our Pipeline.

Please supply a comprehensive {language} unit test for this question. DO NOT include any answer or any other things at this stage.

{format_reminder}

Attention to follow and implement the ‘{project_name}‘ project structure, each file should replace in ‘{project_name}‘ folder and have similar filepath to ensure the unit test can be run successfully.

Now, begin! {end_suffix}

Figure 13: Unit Test Prompt for Our Pipeline.

Please supply a comprehensive {language} answer and necessary dependencies for this question.

{format_reminder}

Attention to follow and implement the ‘{project_name}‘ project structure, each file should replace in ‘{project_name}‘ folder and have similar filepath to ensure the answer can be run successfully.

Now, begin! {end_suffix}

Figure 14: Reference Answer Prompt for Our Pipeline.

1. Attention Our JavaScript Environment is ‘Node.js v16.20.2‘.
2. Attention Our Rust Environment is ‘rustc 1.75.0 (82e1608df 2023-12-21) (built from a source tarball)‘, only support rust edition <= 2021.

Figure 15: Optional End Suffix Prompt Examples for Our Pipeline.

D Unit Test Running Command Setup

We provided unit test commands for our pipeline, LiveRepoReflection and RepoReflection-Instruct in Figure 16.

“python”: “pytest”
“rust”: “cargo test – –include-ignored”
“go”: “go test ./...”
“javascript”: “./npm-test.sh”
“cpp”: “./cpp-test.sh”
“java”: “./gradlew test –no-daemon”

Figure 16: Unit Test Command Setup.