CodeArena: Evaluating and Aligning CodeLLMs on Human Preference

Jian Yang[1], Jiaxi Yang[2,3], Wei Zhang[1], Ke Jin[1], Yibo Miao[4], Lei Zhang[2,3], Liqun Yang[1], Zeyu Cui[4], Yichang Zhang[4], Zhoujun Li[1], Binyuan Hui[4], Junyang Lin[4]
[1]Beihang University; [2]Chinese Academy of Sciences;
[3]University of Chinese Academy of Sciences; [4]Alibaba Group
{jiayang}@buaa.edu.cn

Abstract

Code large language models (codeLLMs) have made significant strides in code generation. Most previous code-related benchmarks, which consist of various programming exercises along with the corresponding test cases, are used as a common measure to evaluate the performance and capabilities of code LLMs. However, the current code LLMs focus on synthesizing the correct code snippet, ignoring the alignment with human preferences, where the query should be sampled from the practical application scenarios and the model-generated responses should satisfy the human preference. To bridge the gap between the model-generated response and human preference, we present a rigorous human-curated benchmark **CodeArena** to emulate the complexity and diversity of real-world coding tasks, where 397 high-quality samples spanning 40 categories and 44 programming languages, carefully curated from user queries. Further, we propose a diverse synthetic instruction corpus SynCode-Instruct (nearly 20B tokens) by scaling instructions from the website to verify the effectiveness of the large-scale synthetic instruction fine-tuning, where Qwen2.5-SynCoder totally trained on synthetic instruction data can achieve top-tier performance of open-source code LLMs. The results find performance differences between execution-based benchmarks and CodeArena. Our systematic experiments of CodeArena on 40+ LLMs reveal a notable performance gap between open SOTA code LLMs (e.g. Qwen2.5-Coder) and proprietary LLMs (e.g., OpenAI o1), underscoring the importance of the human preference alignment.[1]

1 Introduction

Advanced large language models (LLMs)(OpenAI, 2023; Anthropic, 2023) have demonstrated impressive performance across a wide range of tasks, particularly excelling in code completion and generation. Code capabilities have established LLMs as

[1]The evaluation code and leaderboard will be released.

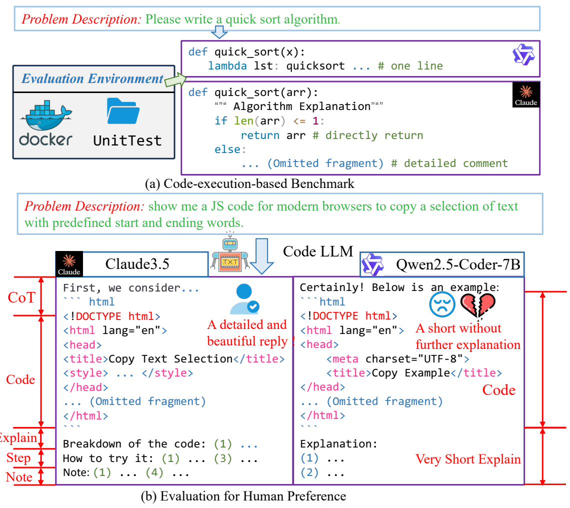

Figure 1: A comparison between the Claude3.5 with better human preference and Qwen2.5-Coder-7B-Instruct. Qwen2.5-Coder-7B-Instruct solves the user question by simply replying with the code snippet without details.

essential productivity tools in software engineering. Recently, open code-specific LLMs, such as StarCoder(Li et al., 2023), DeepSeekCoder (Guo et al., 2024a), and QwenCoder (Hui et al., 2024), have made significant progress, achieving performance on fundamental code generation tasks (Austin et al., 2021; Cassano et al., 2023) that approaches the level of top-tier proprietary models. Moreover, their open and transparent model weights address the concerns of developers about privacy, enabling the deployment of localized code assistants.

With the advancing code capabilities of LLMs, effectively evaluating performance on code-related tasks has emerged as a challenge. Popular code-related benchmarks typically focus on self-contained function snippets, relying on a limited number of test cases to verify code correctness, such as HumanEval (Chen et al., 2021a), MBPP (Austin et al., 2021) and BigCodeBench (Zhuo et al., 2024). While recent efforts have expanded the scope of test cases (Liu et al., 2023), tasks (Lai et al., 2022) and program-

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 9672–9683
November 4-9, 2025

ming languages (Chai et al., 2024; Kwiatkowski et al., 2019), these benchmarks remain constrained to validating the correctness of generated code snippets. However, ChatBot Arena (Chiang et al., 2024) has demonstrated that alignment between model-generated responses and user preferences is also a critical evaluation criterion. As shown in Figure 1, Qwen2.5-Coder primarily generates alone code snippets, while Claude3.5 produces responses that include detailed explanations, well-structured formatting, and code comments, making it more favorable in terms of human preference. Therefore, there is an urgent need to establish a human preference benchmark specifically for code-related tasks, enabling the community to evaluate and track the alignment between human preferences and model-generated responses in real-world scenarios. Furthermore, effective data for improving the human preference alignment of codeLLMs remains scarce. Achieving robust alignment across diverse coding tasks poses significant challenges, particularly in terms of the quantity and quality of data required during the supervised fine-tuning (SFT) stage.

To this end, we first introduce a comprehensive human-curated benchmark, **CodeArena**, comprising 397 high-quality samples across 40 categories derived from real-world user queries. Additionally, we develop a diverse synthetic instruction corpus, **SynCode-Instruct**, containing nearly 20 billion tokens, by scaling instructions from web sources. Our extensive evaluation of over nearly 40 large language models (LLMs) using CodeArena reveals significant performance differences between code-execution-based benchmarks and our human-curated benchmark. Notably, we observe a substantial performance gap between open-source code LLMs (such as Qwen-Coder) and closed-source LLMs (like the o1 and Claude series), emphasizing the critical role of aligning AI models with human preferences in coding tasks.

The contributions are summarized as follows: (1) We propose CodeArena comprised of 397 manually annotated samples, a comprehensive code evaluation benchmark for evaluating the alignment between the model-generated response and human preference, encompassing 7 major categories and 40 subcategories. (2) We introduce SynCode-Instruct, the large-scale synthetic code instruction corpora from the website. Based on SynCode-Instruct, an effective coder Qwen2.5-SynCoder is used as a strong baseline for CodeArena. (3) We systematically evaluate 40+ LLMs on CodeArena and create a leaderboard to dynamically update the results. Notably, extensive experiments suggest that CodeArena can effectively measure the alignment between the model-generated response and human preference.

2 CodeArena

Definition of Human Preference 'Human preference' is a multi-dimensional, comprehensive concept, which covers but is not limited to code correctness, readability, completeness of explanation, format standardization, quality of comments, and whether the overall response meets user expectations of practicality and interaction friendliness.

Dataset Statistics In Table 1 and Figure 2, CodeArena consists of nearly 400 problems. All samples can be classified into 7 main classes and 40 subclasses. Each sample in CodeArena includes (*question, gpt-4o-2024-05-13 response, gpt-4o-2024-08-06 response, gpt-4-turbo-2024-04-09 response*) and we adopt the *gpt-4-turbo-2024-04-09* as the baseline in this paper. We tokenized the question prompts using the Qwen2.5-Coder tokenizer, resulting in question lengths ranging from 5 to 6736 tokens, with an average length of 291 tokens, as detailed in Table 1.

Statistics	Number
Problems	397
User Interface&Experience	45
Development&Programming	131
Specialized Computing	91
Tools, Environments, and Application	39
Miscellaneous and General Inquiry	62
Databases&Data Handling	22
Miscellaneous and General Inquiry	7
#Difficulty Level	
- Easy/Medium/Hard	97/173/132
Length	
Question	
- *maximum length*	6736 tokens
- *minimum length*	5 tokens
- *avg length*	291 tokens
Baseline Answer	
- *maximum length*	5913 tokens
- *minimum length*	7 tokens
- *avg length*	4517 tokens

Table 1: CodeArena dataset statistics.

Multiple Programming Languages Figure 3 plots the distribution of programming languages, where we strive to cover common programming languages in CodeArena. Unlike previous stud-

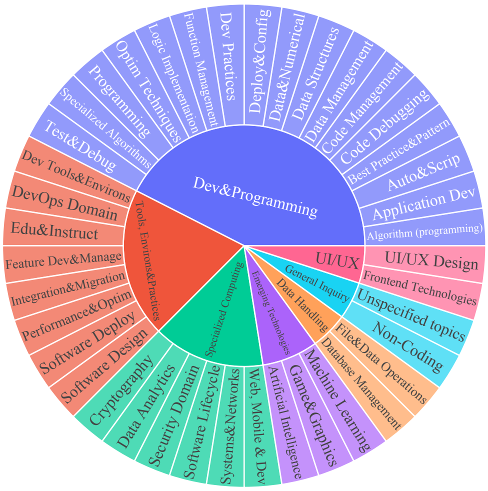

Figure 2: Task types of CodeArena.

ies (Cassano et al., 2023), our benchmarks emphasize a diverse range of programming languages that are commonly used in everyday programming tasks. For instance, we have incorporated languages like "Google Apps Script (GAS)" and "PowerShell" in CodeArena to better address the needs of practical Q&A scenarios.

Difficulty levels of CodeArena Figure 4 shows the difficulty levels of CodeArena, where all samples are classified into *easy*, *medium*, and *hard*. The majority of the samples are recognized as medium or hard, presenting a significant challenge to LLMs.

Human Annotation & Quality Control To make CodeArena a comprehensive evaluation benchmark, we implement a rigorous human annotation process involving 4 full-time employees proficient in various programming languages for human annotation and 4 other senior programming developers for quality check. All annotators participate in a annotation tutorial and learn the annotation guidelines. The annotation process involved creating a new question based on the given question, checking the difficulty level (easy/medium/hard) based on the complexity of the prompt, and annotating the corresponding programming languages. Following the classification in Figure 2, we uniformly sample 2K samples and assign them to annotators. The annotators select 822 suitable original samples to create queries. The process includes regular quality checks and feedback sessions to maintain high standards throughout the annotation phase, which results in a diverse and well-curated dataset spanning multiple programming languages and tasks, suitable for evaluating and improving alignment between the human preference and model-generated response. The other four senior programming developers vote on the same issue to determine whether it is valid and can be resolved. Finally, 397 samples are kept (at least 3 checkers reach a consensus) to from CodeArena, considering the cost of the LLM-as-a-judge.

Evaluation Inspired by the previous work (Chiang et al., 2024), we apply *GPT-4o-2024-08-06* as the judger to evaluate the model performance. Specifically, we use two games "compare A and B " and "compare B and A" (avoid the relative position of A and B affecting the results) to calculate the win rate of A compared to the baseline B.

Decontainmation. To avoid data leakage, we apply decontamination to ensure the uniqueness of prompts in CodeArena, by removing exact matches (10-gram word overlap) from MultiPL-E (Cassano et al., 2023), MBPP (Austin et al., 2021), McEval (Chen et al., 2021a), and Natural-CodeBench (Zhang et al., 2024).

Comparison with other benchmarks We compare CodeArena with other code benchmarks. Our benchmark provides a valuable comprehensive benchmark for 40 subtasks and 44 programming languages, which satisfies the evaluation in realistic scenarios. CodeArena provides many problems for evaluation under realistic scenarios, which are not suitable for verification through unit testing.

3 SynCode-Instruct

Recall from Common Crawl. A trained fast-text is used to distinguish the code-related text and other common raw text, which is used to recall and clean potential code data and filter out low-quality content using weak model-based classifiers and scorers.

Code Classification for Code Snippet. We extract the first layer of CodeBERT (Feng et al., 2020) and fine-tune the tiny classifier on nearly 100 programming languages to build a language identification model. We keep the main language data (e.g. C, Python, and Java) and downsample high-resource language data (e.g. HTML and Java) to keep the balance. Besides, we also remove the samples with no code snippets.

Scaling Code Instruction Initially, we adopt rule-based filtering to clean pre-extracted content

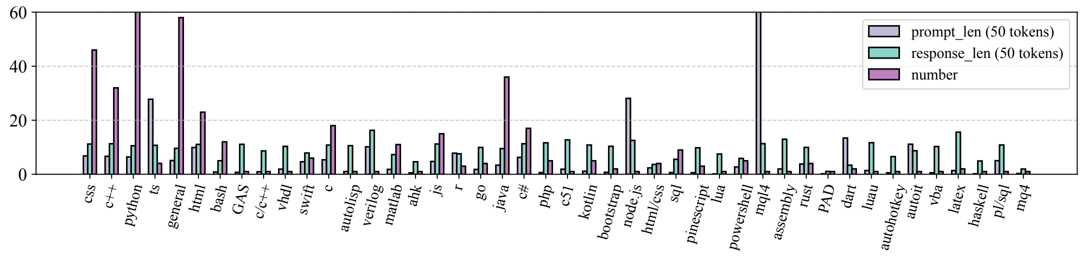

Figure 3: Statistics of programming languages in CodeArena.

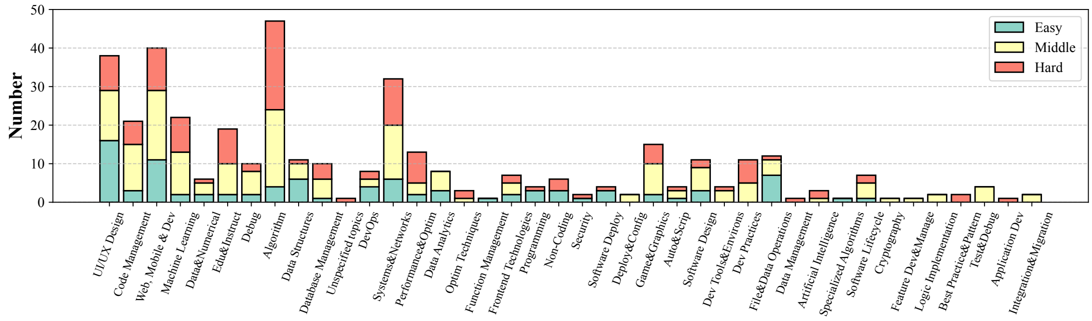

Figure 4: Number of samples of different difficulties (Easy/Medium/Hard) across categories in CodeArena.

Benchmark	#Programming Languages	#Task	Source	#Languages	Evaluation	Human Annotation
HumanEval (Chen et al., 2021a)	1	1	Human Creation	1	Execution	✓
MBPP (Austin et al., 2021)	1	1	Human Creation	1	Execution	✓
LiveCodeBench (Jain et al., 2024)	1	4	Scraped from Code Contest Website	1	Execution	✓
MultiPl-E (Cassano et al., 2023)	24	1	Translated from HumanEval & MBPP	1	Execution	✗
McEval (Chai et al., 2024)	40	3	Human Creation	1	Execution	✓
MdEval (Liu et al., 2024c)	18	3	Human Creation	1	Execution	✓
CruxEval (Gu et al., 2024)	1	2	LLM Generation	1	Execution	✗
NaturalCodeBench (Zhang et al., 2024)	2	6	Scrape & LLM Generation & Human Filtered	1	Execution	✗
DebugBench (Tian et al., 2024)	3	18	Scrape & LLM Generation & Human Filtered	1	Execution	✗
CodeEditorBench (Guo et al., 2024b)	3	4	Scrape & LLM Generation & Human Filtered	1	Execution	✗
CodeArena (Ours)	44	40	Online Q&A	2	Human Preference	✓

Table 2: Comparison between CodeArena and other benchmarks. CodeArena provides a comprehensive view by creating diverse user prompts to evaluation alignment between the model-generated response and human preference.

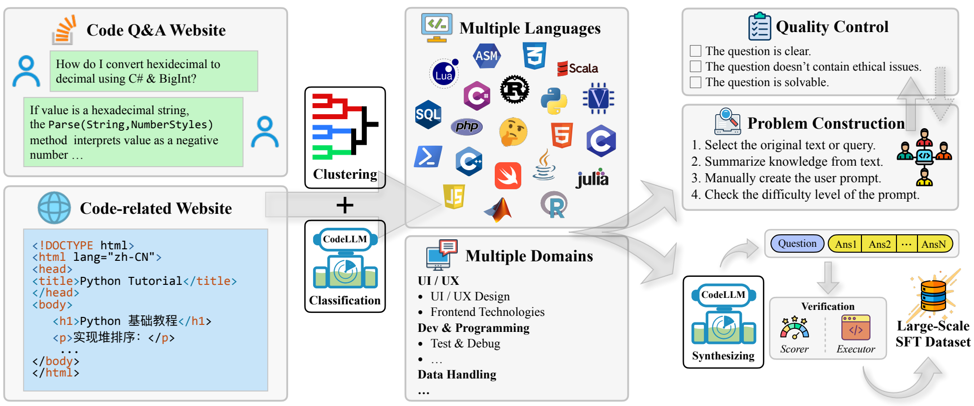

Figure 5: Overview of the CodeArena creation benchmark. We first collect the online code Q&A and code-related raw text from the website. We cluster the code-related data and classify them into different categories using LLM. We uniformly sample the samples from different subtasks as the seed data for manual annotation.

from recalled documents by removing site information, advertisements, and HTML tags, thereby significantly reducing document length for further processing. Different from the previous work (Yue et al., 2024), we utilize Qwen2.5-72B to create new questions instead of extracting question and answer pairs. As shown in Figure 6. We use the Qwen2.5-Coder to generate multiple responses by sampling

```
You are an expert programmer and educational
content creator specializing in crafting high-
quality programming questions.
Task: Create an engaging, self-contained
programming question inspired by the given text.
Guidelines:
1. Human Language: The question should be in
either Chinese or English.
2. Programming Language: The programming
language of the created problem must be
consistent with the programming language of the
given text.
3. Inspiration: Draw inspiration from the
concepts, techniques, or themes present in the
given text.
4. Independence: Ensure the question is
completely self-contained and does not require
knowledge of the original text to understand or
solve.
5. Clarity: Provide clear instructions and
requirements for the task.
6. Difficulty: Adjust the complexity to be
challenging yet solvable for an intermediate
programmer.
7. Relevance: Focus on practical, real-world
applications of programming concepts.
8. Originality: Avoid directly copying examples
from the text; instead, create a new scenario or
problem.
Given Text:
{given text}
Created Question:
{Your carefully crafted programming question
goes here}
```

Figure 6: Prompt of generating large-scale self-contained synthetic instruction data.

for the same document. For the algorithmic generated question and answer, we first adopt a fine-tuned generator to generate the test cases and adopt the multilingual sandbox to verify the correctness of the generated code snippet. As shown in Figure 5, for the non-algorithmic query, we first randomly generate four candidates and use the LLM to score the candidates (LLM scorer), where the candidates are fed into the LLM to select the best response with the reason. For the algorithmic queries, the generated test cases by LLM are used to verify the correctness of the responses (Executor). Finally, we select the response with the best score as the response to create SynCode-Instruct. The synthetic instruction corpora generated by Qwen2.5 is used for the first stage and the high-quality data from GPT-4o is used for the second stage. The large-scale code SFT corpus include nearly 20B tokens and the average length of question and answer is shorter than 2048 tokens.

4 Experimental Setup

4.1 Instruction Dataset

CodeLLMs We evaluate 40+ models with sizes ranging from 0.5B to 200B parameters, including general/code LLMs and open/closed-source models. For general models, we evaluate GPTs (Brown et al., 2020; OpenAI, 2023) (GPT-3.5-Turbo, GPT4-o), Qwen series (Qwen2.5 and Qwen-Max) (Bai et al., 2023), Claude series (Anthropic, 2023), Llama3/3.1 (Dubey et al., 2024), Yi (Young et al., 2024), and o1 series. For code models, we test CodeLlama (Rozière et al., 2023), OpenCoder (Huang et al., 2024), Qwen-Coder (Hui et al., 2024), DeepSeekCoder (Guo et al., 2024a), and CodeStral (MistralAI, 2024).

4.2 Evaluation Benchmark

The **EvalPlus** (Liu et al., 2023) is an upgraded version of the HumanEval (Chen et al., 2021a) and MBPP (Austin et al., 2021) to test the code generation capabilities. The benchmark reports the scores of HumanEval (HE)/MBPP with base test cases and HumanEval+ (HE+)/MBPP+ with plus test cases. The **MultiPL-E** test set (Cassano et al., 2023) contains the HumanEval (Python) and translated test set of other programming languages, i.e., Java, C++, Javascript, and Typescript. Different from the EvalPlus and MultiPL-E, **CodeArena** consists of many non-algorihtmic, which is not suitable for code-execution-based evaluation. Each question is scored twice to calculate the win rate and tie rate by GPT-4o using a different input order "A, B" and "B, A", where "A" is the baseline from `gpt-4-turbo-2024-04-09` and "B" is the model-generated response.

4.3 Evaluation Metrics

For the code execution benchmarks EvalPlus and MultiPL-E, we extract the expected function and feed the test cases into the extracted function to verify the correctness of the generation and report greedy Pass@1 (Chen et al., 2021a). Due to the high cost of collecting human preferences (Zheng et al., 2023a), we use pairwise comparison for judgment (LLM-as-a-Judge), where an LLM judger is fed with a question and two answers and determines which one is better or declares a tie[2]. We report win rate/tie rate for CodeArena.

4.4 Impletmentation Details

We fine-tune Qwen2.5-Coder-32B on nearly 20B synthetic tokens generated from website data, where GPT-4o generates 1B tokens and Qwen2.5-Coder-Instruct generates the left tokens. Qwen2.5-SynCoder is fine-tuned on the synthetic instruction

[2] https://github.com/lmarena/arena-hard-auto

Model	Size	UI&UX	Development& Programming	Specialized Computing	Tools, Environs, & Practices	Emerging Techs &Apps	Miscellaneous & General Inquiry	Databases& Data Handling	Avg.
Proprietary LLMs and 200B+ LLMs									
Claude-3.5-Sonnet-20240620	🔒	88.9/2.2	77.3/13.6	74.2/18.0	81.4/11.9	78.9/10.5	71.4/28.6	63.6/4.5	77.8/12.5
Claude-3.5-Sonnet-20241022	🔒	82.2/6.7	75.8/12.9	76.4/16.9	84.7/10.2	84.2/13.2	57.1/28.6	68.2/22.7	78.1/13.5
GPT-3.5-turbo-0125	🔒	17.8/24.4	11.4/20.5	4.5/19.1	11.9/18.6	10.5/21.1	13.6/9.1	0.0/14.3	10.5/19.6
GPT-4o-mini-2024-07-18	🔒	71.1/13.3	62.1/17.4	50.0/13.6	65.2/14.6	72.9/13.6	71.1/18.4	71.4/14.3	65.8/15.6
GPT-4o-2024-08-06	🔒	66.7/17.8	72.7/19.7	62.9/19.1	69.5/15.3	76.3/13.2	85.7/14.3	59.1/22.7	69.1/18.1
o1-mini	🔒	93.3/4.4	94.7/2.6	84.1/7.6	91.0/5.6	88.1/3.4	95.5/0.0	100.0/0.0	89.3/5.1
o1-preview	🔒	93.3/2.2	81.8/7.6	85.4/7.9	78.0/6.8	92.1/2.6	77.3/4.5	71.4/28.6	83.9/6.6
Yi-lightning	🔒	62.2/15.6	60.0/11.5	57.9/5.3	49.4/16.9	71.2/11.9	54.5/13.6	85.7/0.0	59.5/12.6
Doubao-Pro	🔒	51.1/20.0	40.8/18.5	55.3/26.3	38.2/19.1	47.5/22.0	36.4/31.8	42.9/57.1	43.6/21.5
Qwen-Max	🔒	75.6/17.8	74.2/13.6	59.6/24.7	78.0/6.8	68.4/23.7	100.0/0.0	81.8/4.5	71.9/15.8
0.5B+ Open-source LLMs									
Qwen2.5-0.5B-Instruct	0.5B	2.2/4.4	4.6/4.6	5.3/10.5	2.2/4.5	3.4/5.1	4.5/9.1	0.0/14.3	3.6/5.6
Qwen2.5-Coder-0.5B-Instruct	0.5B	2.2/2.2	4.6/6.9	2.6/5.3	4.5/2.2	3.4/5.1	4.5/0.0	28.6/14.3	4.4/4.6
1B+ Open-source LLMs									
DS-Coder-1.3B-Instruct	1.3B	66.7/2.2	2.3/5.4	2.6/10.5	1.7/6.8	0.0/9.1	2.2/3.4	0.0/14.3	2.6/5.6
Yi-Coder-1.5B-Chat	1.5B	11.1/2.2	5.1/3.4	5.4/4.6	2.6/5.3	2.2/5.6	4.5/4.5	14.3/14.3	7.4/5.1
Qwen2.5-Coder-1.5B-Instruct	1.5B	11.1/4.4	15.9/9.1	9.0/16.9	13.6/11.9	13.2/5.3	14.3/42.9	18.2/4.5	13.2/10.7
OpenCoder-1.5B-Instruct	1.5B	11.1/4.4	3.8/5.4	0.0/5.3	2.2/4.5	3.4/8.5	4.5/9.1	0.0/0.0	6.7/3.8
3B+ Open-source LLMs									
Qwen2.5-Coder-3B-Instruct	3B	35.6/11.1	29.5/10.6	27.0/15.7	20.3/18.6	28.9/10.5	42.9/14.3	27.3/13.6	28.3/13.3
6B+ Open-source Models									
CodeLlama-7B-Instruct	7B	33.3/8.9	28.8/18.6	23.8/13.8	18.2/9.1	31.6/5.3	29.2/14.6	71.4/0.0	28.2/12.8
Llama3-8B-Instruct	7B	20.0/17.8	14.6/11.5	15.8/2.6	13.5/9.0	16.9/11.9	22.7/0.0	57.1/14.3	16.7/10.3
Llama3.1-8B-Instruct	7B	2.2/8.9	4.5/10.1	3.8/6.2	3.4/6.8	5.3/2.6	9.1/9.1	14.3/0.0	7.9/4.4
DS-Coder-6.7B-Instruct	6.7B	11.1/17.8	13.1/13.8	13.6/8.5	13.2/7.9	9.0/7.9	13.6/4.5	28.6/0.0	12.3/10.8
CodeQwen1.5-7B-Chat	7B	17.8/15.6	13.8/12.3	15.8/0.0	15.7/9.0	15.3/15.3	18.2/13.6	14.3/42.9	15.4/11.8
Yi-Coder-9B-Chat	9B	15.6/17.8	15.4/9.2	15.8/7.9	13.5/13.5	10.2/20.3	18.2/13.6	28.6/28.6	14.6/13.3
DS-Coder-V2-Lite-Instruct	2.4/16B	42.2/20.0	33.3/17.4	31.5/16.9	35.6/20.3	39.5/21.1	71.4/14.3	31.8/22.7	35.5/18.6
Qwen2.5-Coder-7B-Instruct	7B	40.0/22.2	46.2/19.7	43.8/15.7	40.7/20.3	34.2/15.8	71.4/0.0	40.9/22.7	43.1/18.6
OpenCoder-8B-Instruct	8B	24.4/8.9	14.6/8.5	10.5/7.9	9.0/4.5	13.6/6.8	18.2/9.1	14.3/0.0	14.1/7.1
13B+ Models									
CodeLlama-13B-Instruct	13B	13.3/4.4	7.9/6.7	6.8/8.5	7.7/6.2	4.5/4.5	5.3/5.3	14.3/14.3	11.2/7.9
Starcoder2-15B-Instruct-v0.1	15B	6.7/6.7	6.8/12.9	4.5/15.7	6.8/6.8	5.3/13.2	13.6/13.6	0.0/14.3	6.4/12.0
Qwen2.5-Coder-14B-Instruct	14B	51.1/24.4	53.0/17.4	52.8/16.9	50.8/18.6	57.9/7.9	28.6/28.6	36.4/27.3	60.6/51.5
20B+ Models									
CodeLlama-34B-Instruct	34B	11.1/6.7	2.6/2.6	6.9/2.3	8.5/6.8	7.9/10.1	9.1/9.1	14.3/0.0	7.7/5.6
CodeStral-22B-v0.1	22B	17.8/22.2	27.3/13.6	14.6/14.6	25.4/10.2	18.4/10.5	14.3/42.9	22.7/22.7	21.7/15.8
DS-Coder-33B-Instruct	33B	13.3/11.1	22.0/9.8	12.4/12.4	13.6/6.8	13.2/18.4	28.6/42.9	22.7/18.2	16.8/12.0
CodeLlama-70B-Instruct	70B	11.1/22.2	9.2/10.0	10.5/5.3	9.0/6.7	16.9/8.5	9.1/13.6	0.0/0.0	15.5/10.5
DS-Coder-V2-Instruct	21/236B	55.6/11.1	62.1/18.2	60.7/14.6	50.8/18.6	52.6/21.1	71.4/14.3	40.9/31.8	57.4/17.6
DS-V2.5	21/236B	77.8/11.1	72.0/12.9	71.9/13.5	71.2/8.5	73.7/10.5	100.0/0.0	68.2/13.6	73.0/11.7
Llama3-70B-Instruct	7B	35.6/20.0	26.2/26.2	25.4/22.0	34.2/15.8	23.6/14.6	36.4/4.5	14.3/57.1	27.7/20.5
Llama3.1-70B-Instruct	7B	48.9/24.4	43.8/20.0	34.2/26.3	40.4/22.5	54.2/20.3	45.5/9.1	71.4/14.3	44.9/21.0
Qwen2.5-Coder-32B-Instruct	32B	71.1/13.3	66.7/15.9	67.4/16.9	74.6/13.6	65.8/18.4	100.0/0.0	63.6/18.2	68.9/15.6
Qwen2.5-32B-Instruct	32B	62.2/15.6	52.3/15.4	57.9/18.4	50.6/23.6	54.2/13.6	50.0/13.6	71.4/14.3	54.1/17.1
QwQ-32B-Preview	32B	53.3/15.6	56.8/16.7	50.6/16.9	64.4/5.1	52.6/21.1	85.7/0.0	63.6/9.1	56.6/14.5
Qwen2.5-72B-Instruct	72B	82.2/6.7	71.5/14.6	76.3/13.2	75.3/15.7	71.2/18.6	63.6/13.6	85.7/14.3	73.8/14.4
Qwen2.5-SynCoder	32B	55.6/26.7	49.2/20.8	36.8/36.8	50.6/20.2	52.5/20.3	40.9/18.2	57.1/0.0	49.2/22.3

Table 3: The win/tie rate of different instruction LLMs on CodeArena. The underlined numbers represent the best scores within the same model size range.

corpus SynCode-Instruct with 256 NVIDIA A100-80GB GPUs. The learning rate first increases into 3×10^{-4} with 100 warmup steps and then adopts a cosine decay scheduler. We adopt the Adam optimizer (Kingma and Ba, 2015) with a global batch size of 2048 samples and a tensor parallel size of 8, truncating sentences to 32K tokens.

5 Results and Discussion

5.1 Main Results

CodeArena. Table 3 shows that the win rate/tie rate of different instruction LLM on CodeArena. The closed-source LLMs such as Claude and o1 series still get a dominant advantage compared to Qwen2.5-Coder and DeepseekCoder. There still exists a notable performance gap between open codeLLMs (e.g. Qwen-Coder) and closed-source LLMs (e.g., o1 and Claude series), emphasizing the importance of alignment between model-generated response human preference. Qwen2.5-SynCoder totally trained on the large-scale synthetic instruction corpus SynCode-Instruct can still get a strong performance on CodeArena, which verifies the correctness of the route of taking large-scale synthetic data to improve model performance.

EvalPlus and MultiPL-E. Table 4 shows that Qwen2.5-SynCoder significantly beats previous strong open-source baselines using large-scale synthetic instruction, closing the gap with GPT-4o and Claude, which verifies that the large-scale synthetic data can bring more significant improvement for the base model in the code-execution-based bench-

Model	Size	HE	HE+	MBPP	MBPP+	Python	Java	C++	C#	TS	JS	PHP	Bash	Avg.
Closed-APIs														
Claude-3.5-Sonnet-20240620	🔒	89.0	81.1	87.6	72.0	89.6	86.1	82.6	85.4	84.3	84.5	80.7	48.1	80.2
Claude-3.5-Sonnet-20241022	🔒	92.1	86.0	91.0	74.6	93.9	86.7	88.2	87.3	88.1	91.3	82.6	52.5	83.8
GPT-4o-mini-2024-07-18	🔒	87.8	84.8	86.0	72.2	87.2	75.9	77.6	79.7	79.2	81.4	75.2	43.7	75.0
GPT-4o-2024-08-06	🔒	92.1	86.0	86.8	72.5	90.9	83.5	76.4	81.0	83.6	90.1	78.9	48.1	79.1
o1-mini	🔒	97.6	90.2	93.9	78.3	95.7	90.5	93.8	77.2	91.2	92.5	84.5	55.1	85.1
o1-preview	🔒	95.1	88.4	93.4	77.8	96.3	88.0	91.9	84.2	90.6	93.8	90.1	47.5	85.3
0.5B+ Models														
Qwen2.5-Coder-0.5B-Instruct	0.5B	61.6	57.3	52.4	43.7	61.6	57.3	52.4	43.7	50.3	50.3	52.8	27.8	49.6
1B+ Models														
DS-Coder-1.3B-Instruct	1.3B	65.9	60.4	65.3	54.8	65.2	51.9	45.3	55.1	59.7	52.2	45.3	12.7	48.4
Yi-Coder-1.5B-Chat	1.5B	69.5	64.0	65.9	57.7	67.7	51.9	49.1	57.6	57.9	59.6	52.2	19.0	51.9
Qwen2.5-Coder-1.5B-Instruct	1.5B	70.7	66.5	69.2	59.4	71.2	55.7	50.9	64.6	61.0	62.1	59.0	29.1	56.7
3B+ Models														
Qwen2.5-Coder-3B-Instruct	3B	84.1	80.5	73.6	62.4	83.5	74.7	68.3	78.5	79.9	75.2	73.3	43.0	72.1
6B+ Models														
CodeLlama-7B-Instruct	7B	40.9	33.5	54.0	44.4	34.8	30.4	31.1	21.6	32.7	-	28.6	10.1	-
DS-Coder-6.7B-Instruct	6.7B	74.4	71.3	74.9	65.6	78.6	68.4	63.4	72.8	67.2	72.7	68.9	36.7	66.1
CodeQwen1.5-7B-Chat	7B	83.5	78.7	77.7	67.2	84.1	73.4	74.5	77.8	71.7	75.2	70.8	39.2	70.8
Yi-Coder-9B-Chat	9B	82.3	74.4	82.0	69.0	85.4	76.0	67.7	76.6	72.3	78.9	72.1	45.6	71.8
DS-Coder-V2-Lite-Instruct	2.4/16B	81.1	75.6	82.8	70.4	81.1	76.6	75.8	76.6	80.5	77.6	74.5	43.0	73.2
Qwen2.5-Coder-7B-Instruct	7B	88.4	84.1	83.5	71.7	87.8	76.5	75.6	80.3	81.8	83.2	78.3	48.7	76.5
OpenCoder-8B-Instruct	8B	83.5	78.7	79.1	69.0	83.5	72.2	61.5	75.9	78.0	79.5	73.3	44.3	71.0
13B+ Models														
CodeLlama-13B-Instruct	13B	40.2	32.3	60.3	51.1	42.7	40.5	42.2	24.0	39.0	-	32.3	13.9	-
Starcoder2-15B-Instruct-v0.1	15B	67.7	60.4	78.0	65.1	68.9	53.8	50.9	62.7	57.9	59.6	53.4	24.7	54.0
Qwen2.5-Coder-14B-Instruct	14B	89.6	87.2	86.2	72.8	89.0	79.7	85.1	84.2	86.8	84.5	80.1	47.5	79.6
20B+ Models														
CodeLlama-34B-Instruct	34B	48.2	40.2	61.1	50.5	41.5	43.7	45.3	31.0	40.3	-	36.6	19.6	-
CodeStral-22B-v0.1	22B	81.1	73.2	78.2	62.2	81.1	63.3	65.2	43.7	68.6	-	68.9	42.4	-
DS-Coder-33B-Instruct	33B	81.1	75.0	80.4	70.1	79.3	73.4	68.9	74.1	67.9	73.9	72.7	43.0	69.2
CodeLlama-70B-Instruct	70B	72.0	65.9	77.8	64.6	67.8	58.2	53.4	36.7	39.0	-	58.4	29.7	-
DS-Coder-V2-Instruct	21/236B	85.4	82.3	89.4	75.1	90.2	82.3	84.8	82.3	83.0	84.5	79.5	52.5	79.9
Qwen2.5-Coder-32B-Instruct	32B	92.7	87.2	90.2	75.1	92.7	80.4	79.5	82.9	86.8	85.7	78.9	48.1	79.4
Qwen2.5-32B-Instruct	32B	87.8	82.9	86.8	70.9	88.4	80.4	81.0	74.5	83.5	82.4	78.3	46.8	76.9
Qwen2.5-72B-Instruct	32B	85.4	79.3	90.5	77.0	82.9	81.0	80.7	81.6	81.1	82.0	77.0	48.7	75.1
Qwen2.5-SynCoder	32B	92.7	87.8	86.2	74.7	92.1	80.4	80.7	81.6	83.0	85.7	77.6	49.4	78.8

Table 4: The performance of different instruction LLMs on EvalPlus and MultiPL-E. "HE" denotes the HumanEval, "HE+" denotes the plus version with more test cases, and "MBPP+" denotes the plus version with more test cases.

mark (code generation) compared to CodeArena.

5.2 Discussion

Examples of CodeArena. Figure 7 lists 6 examples from different subtasks, covering Python, HTML, CSS, and Java. Different from the previous benchmarks (Cassano et al., 2023; Jain et al., 2024) comprised of algorithmic questions in a fixed format, the queries of CodeArena are more consistent with the distribution of user questions in real Q&A scenarios. For example, the query "huggingface dataset move all the columns to metadata, except two, problem and solution" is closer to the question style of real users. GPT4o thinks model-generated response B beats the baseline A based on the judgment "B provides a correct and relevant solution using the appropriate library for Hugging Face datasets", which select responses that are more aligned with human preferences.

Difference between CodeArena and Execution-based Benchmark. Compared to MultiPL-E evaluated by code execution, CodeArena is created from real-world Q&A and evaluated by LLM-as-a-judge to evaluate the alignment between the model-generated response and human preference. For example, the LLMs tend to only generate the code without any natural description (even the code is correct) will bring an unsatisfactory experience to users, which will also lead to poor performance in CodeArena. In Figure 8, we can observe that the state-of-the-art closed-source LLMs (e.g. o1 and Claude series) get a balanced performance between the code execution benchmark and CodeArena. The open-source models (e.g. DeepseekCoder and Qwen-Coder) are likely to bring a bad experience

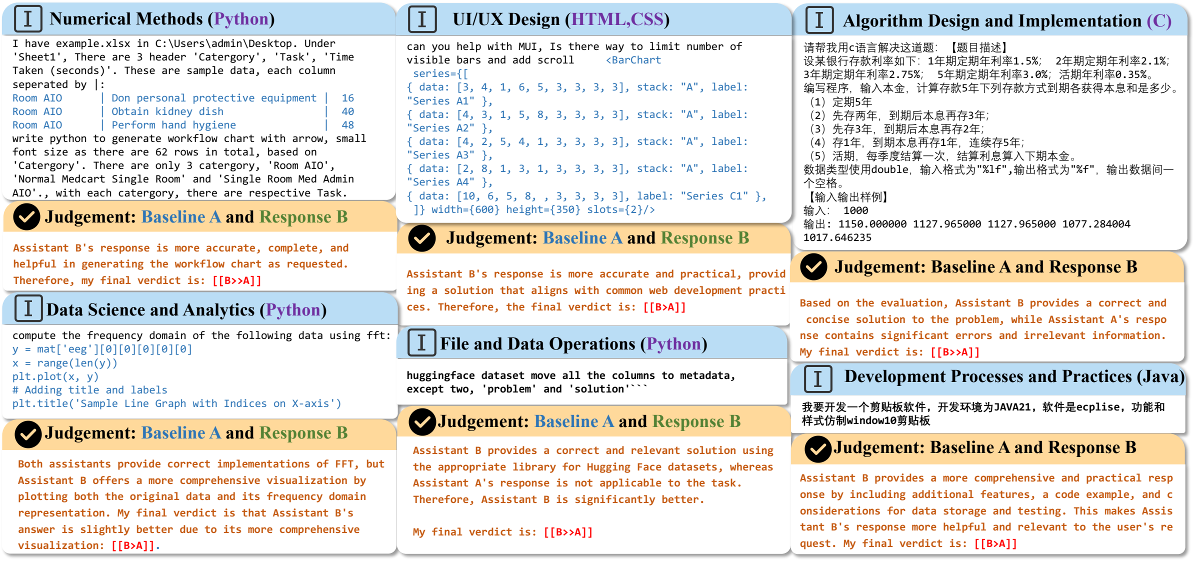

Figure 7: Examples of CodeArena. The LLM judger decides which response is better.

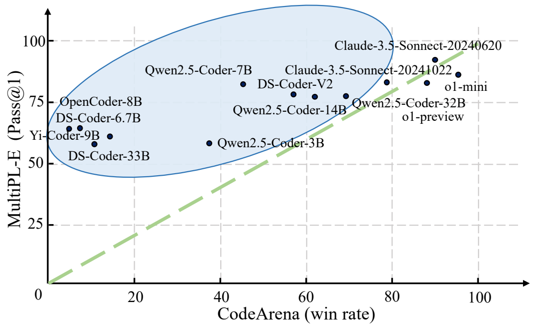

Figure 8: Comparison between MultiPL-E and CodeArena. LLMs in the blue circle present relatively mismatched performances on two benchmarks.

to users, where the generated response lacks a more detailed explanation or more complete details compared to closed-source LLMs.

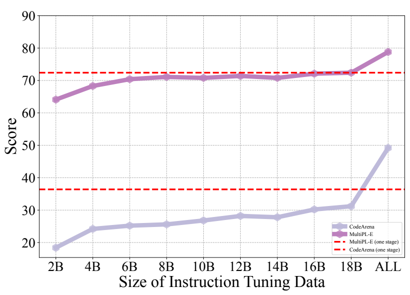

Figure 9: Results of CodeArena with different data size on MultiPL-E and CodeArena.

Scaling Synthetic Instruction Corpora. We would like to further analyze the performance of Qwen2.5-SynCoder in MultiPl-E and CodeArena given different sizes of instruction corpora. Therefore, we select the full instruction (19B synthetic data is at the front of the data and 1B high-quality data is at the end) set SynCode-Instruct and extract the first K billion tokens as the fine-tuned data. We set $K = \{2, 4, \ldots, 20\}$. We randomly extract specific data from the whole corpus. Figure 9 shows the performance on CodeArena. With the increase of instruction data, Qwen2.5-SynCoder still can get significant improvement, which emphasizes the importance of the scaling instruction corpora. The two-stage SFT gets a better performance compared to the one-stage training (red line), where the high-quality data brings a huge improvement at last.

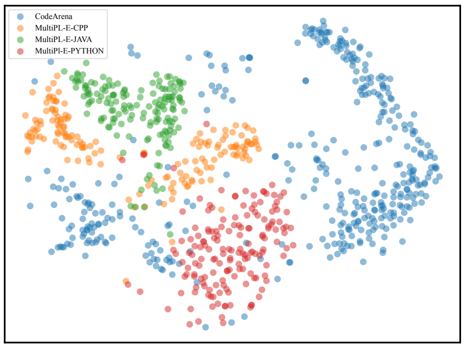

Figure 10: Distribution of CodeArena and MultiPL-E of different languages.

Distribution of different benchmarks. The queries of CodeArena and MultiPL-E (Python, Java, and CPP) are visualized by extracting the last BERT representations for t-SNE (Van der Maaten and Hinton, 2008). The average of all hidden states of the last encoder layer is viewed as the query representation. In Figure 10, the representations of CodeArena are distributed in the whole area, while the representations of different languages in MultiPL-E are separately located in a narrow area, which shows that the distribution of queries in CodeArena is very diverse, which is suitable for evaluating human preferences in realistic scenarios.

6 Related Work

Code Large Language Model. Large language models (LLMs) designed for coding tasks have demonstrated exceptional capabilities in code generation, debugging, translation, and other essential functions for modern software engineering (Chen et al., 2021b; Anthropic, 2023; OpenAI, 2023; Fried et al., 2023; Xu et al., 2022; Sun et al., 2024). Numerous in-file benchmarks have been developed to evaluate these capabilities; however, many of them focus on a limited selection of programming languages, such as Python and Java (Zheng et al., 2023b; Austin et al., 2021; Jain et al., 2024). Recent advancements in code LLMs, including models like Code Llama (Roziere et al., 2023), DeepSeek-Coder (Guo et al., 2024a), OpenCoder (Huang et al., 2024), and Qwen2.5-Coder (Hui et al., 2024), have made significant strides in multilingual code generation and debugging tasks. These models have been effectively evaluated using benchmarks such as MultiPL-E (Cassano et al., 2023), McEval (Chai et al., 2024), and MdEval (Liu et al., 2024c).

Code Benchmarks. To holistically assess diverse code capabilities, numerous benchmarks have been developed for tasks like code translation (Jiao et al., 2023; Yan et al., 2023; Zhu et al., 2022), retrieval (Huang et al., 2021; Husain et al., 2019; Lu et al., 2021), completion (Bavarian et al., 2022; Liu et al., 2024a; Zhang et al., 2023), debugging (Huq et al., 2022; Tian et al., 2024; Liu et al., 2024c), and structured data understanding (Wu et al., 2024; Su et al., 2024). Nonetheless, many of these studies concentrate on assessing only a single aspect of LLM capabilities, often overlooking the evaluation of LLMs as comprehensive program developers across a variety of real-world coding scenarios. Besides, CodePrefBench mainly (Liu et al., 2024b) emphasizes the correctness, efficiency, and security along with human preference, which are all from the existing benchmarks (e.g. EvalPlus). CodeArena focuses on building a complete pipeline to construct human preference data at scale from realistic and real-time web Q&A data.

7 Conclusion

In this work, we introduce CodeArena, a meticulously human-curated benchmark composed of 397 high-quality samples spanning 40 categories, derived from real-world user queries, to address discrepancies between model-generated responses and human preferences in coding tasks. Additionally, we create SynCode-Instruct, a diverse synthetic instruction corpus containing nearly 20 billion tokens, by scaling web-sourced instructions. Our evaluation of over 40+ LLMs using CodeArena highlights significant performance discrepancies between code-execution-based benchmarks and our human-curated benchmark. Notably, there is a remarkable performance gap between open-source code LLMs and closed-source LLMs (such as the o1 series), underscoring the importance of aligning LLMs with human preferences in coding tasks.

Limitations

We acknowledge the following limitations of this study: (1) The evaluation depends on the LLM-as-a-Judge, which requires extra API costs (GPT-4o) and may lead to a biased evaluation. In the future, we will try to design a fairer evaluation metric for coding tasks, which can not be evaluated by unit tests. (2) The effectiveness of the synthetic data is only evaluated on code benchmarks. Its effectiveness in other domains has not been evaluated, limiting the generalizability of the method.

Ethics Statement

CodeArena, as an evaluation benchmark, can comprehensively assess the capability of large language models in realistic scenarios with a wide range of programming tasks and languages, thereby advancing the development of LLM evaluation in the coding domain. However, unsafe queries CodeArena may contain pornographic and personal privacy information. Therefore, to ensure the security and reliability of the queries, the annotators are asked to rephrase the original question to a safe query.

Acknowledgement

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).

References

Anthropic. 2023. Introducing Claude.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*, abs/2309.16609.

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. 2022. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

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. 2021a. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, abs/2107.03374.

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

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael I. Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot arena: An open platform for evaluating llms by human preference. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. 2023. Incoder: A generative model for code infilling and synthesis. In *The Eleventh International Conference on Learning Representations*.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I Wang. 2024. Cruxeval: A benchmark for code reasoning, understanding and execution.

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

Jiawei Guo, Ziming Li, Xueling Liu, Kaijing Ma, Tianyu Zheng, Zhouliang Yu, Ding Pan, Yizhi Li, Ruibo Liu, Yue Wang, et al. 2024b. Codeeditorbench: Evaluating code editing capability of large language models. *arXiv preprint arXiv:2404.03543*.

Junjie Huang, Duyu Tang, Linjun Shou, Ming Gong, Ke Xu, Daxin Jiang, Ming Zhou, and Nan Duan. 2021. Cosqa: 20,000+ web queries for code search and question answering.

Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J Yang, JH Liu, Chenchen Zhang, Linzheng Chai, et al. 2024. Opencoder: The open cookbook for top-tier code large language models. *arXiv e-prints*, pages arXiv–2411.

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*.

Faria Huq, Masum Hasan, Md Mahim Anjum Haque, Sazan Mahbub, Anindya Iqbal, and Toufique Ahmed. 2022. Review4repair: Code review aided automatic program repairing. 143:106765.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, abs/1909.09436.

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*.

Mingsheng Jiao, Tingrui Yu, Xuan Li, Guanjie Qiu, Xiaodong Gu, and Beijun Shen. 2023. On the evaluation of neural code translation: Taxonomy and benchmark. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1529–1541. IEEE.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur P. Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: a benchmark for question answering research. *Trans. Assoc. Comput. Linguistics*, 7:452–466.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-Tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2022. Ds-1000: A natural and reliable benchmark for data science code generation. *ArXiv*, abs/2211.11501.

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! *arXiv preprint arXiv:2305.06161*, abs/2305.06161.

Jiaheng Liu, Ken Deng, Congnan Liu, Jian Yang, Shukai Liu, He Zhu, Peng Zhao, Linzheng Chai, Yanan Wu, Ke Jin, et al. 2024a. M2rc-eval: Massively multilingual repository-level code completion evaluation. *arXiv preprint arXiv:2410.21157*.

Jiawei Liu, Thanh Nguyen, Mingyue Shang, Hantian Ding, Xiaopeng Li, Yu Yu, Varun Kumar, and Zijian Wang. 2024b. Learning code preference via synthetic evolution. *CoRR*, abs/2410.03837.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *arXiv preprint arXiv:2305.01210*, abs/2305.01210.

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

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin 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 *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

MistralAI. 2024. Codestral. `https://mistral.ai/news/codestral`. 2024.05.29.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

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

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

Aofeng Su, Aowen Wang, Chao Ye, Chen Zhou, Ga Zhang, Guangcheng Zhu, Haobo Wang, Haokai Xu, Hao Chen, Haoze Li, et al. 2024. Tablegpt2: A large multimodal model with tabular data integration. *arXiv preprint arXiv:2411.02059*.

Tao Sun, Linzheng Chai, Yuwei Yin Jian Yang, Hongcheng Guo, Jiaheng Liu, Bing Wang, Liqun Yang, and Zhoujun Li. 2024. Unicoder: Scaling code large language model via universal code. *ACL*.

Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2024. Debugbench: Evaluating debugging capability of large language models.

Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xinrun Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, et al. 2024. Tablebench: A comprehensive and complex benchmark for table question answering. *arXiv preprint arXiv:2408.09174*.

Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, pages 1–10.

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.

Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, et al. 2024. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*.

Xiang Yue, Tuney Zheng, Ge Zhang, and Wenhu Chen. 2024. Mammoth2: Scaling instructions from the web. *arXiv preprint arXiv:2405.03548*.

Fengji Zhang, Bei Chen, Yue Zhang, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. RepoCoder: Repository-level code completion through iterative retrieval and generation. *arXiv preprint arXiv:2303.12570*, abs/2303.12570.

Shudan Zhang, Hanlin Zhao, Xiao Liu, Qinkai Zheng, Zehan Qi, Xiaotao Gu, Yuxiao Dong, and Jie Tang. 2024. Naturalcodebench: Examining coding performance mismatch on humaneval and natural user queries. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 7907–7928.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023a. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

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. 2023b. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. *arXiv preprint arXiv:2303.17568*, abs/2303.17568.

Ming Zhu, Aneesh Jain, Karthik Suresh, Roshan Ravindran, Sindhu Tipirneni, and Chandan K Reddy. 2022. Xlcost: A benchmark dataset for cross-lingual code intelligence.

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*.