InfiCoder-Eval: Systematically Evaluating the Question-Answering Capabilities of Code Large Language Models

Linyi Li [1 2] **Shijie Geng** [1 3] **Zhenwen Li** [* 4] **Yibo He** [* 4] **Hao Yu** [* 4] **Ziyue Hua** [* 4] **Guanghan Ning** [1] **Siwei Wang** [1] **Tao Xie** [4] **Hongxia Yang** [1]

Abstract

Large Language Models for understanding and generating code (code LLMs) have witnessed tremendous progress in recent years. With the rapid development of code LLMs, many popular evaluation benchmarks, such as HumanEval, DS-1000, and MBPP, have emerged to measure the performance of code LLMs with a particular focus on code generation tasks. However, they are insufficient to cover the full range of expected capabilities of code LLMs, which span beyond code generation to answering diverse coding-related questions. To fill this gap, we propose InfiCoder-Eval, a large-scale **freeform question-answering (QA) benchmark for code**, comprising 234 carefully selected high-quality Stack Overflow questions that span across 15 programming languages. To evaluate the response correctness, InfiCoder-Eval supports four types of model-free metrics and domain experts carefully choose and concretize the criterion for each question. We conduct a systematic evaluation for more than 80 code LLMs on InfiCoder-Eval, leading to a series of insightful findings. Furthermore, our detailed analyses showcase possible directions for further improvement of code LLMs. InfiCoder-Eval is fully open source at `https://infi-coder.github.io/inficoder-eval/` and continuously maintaining and expanding to foster more scientific and systematic practices for evaluating code LLMs.

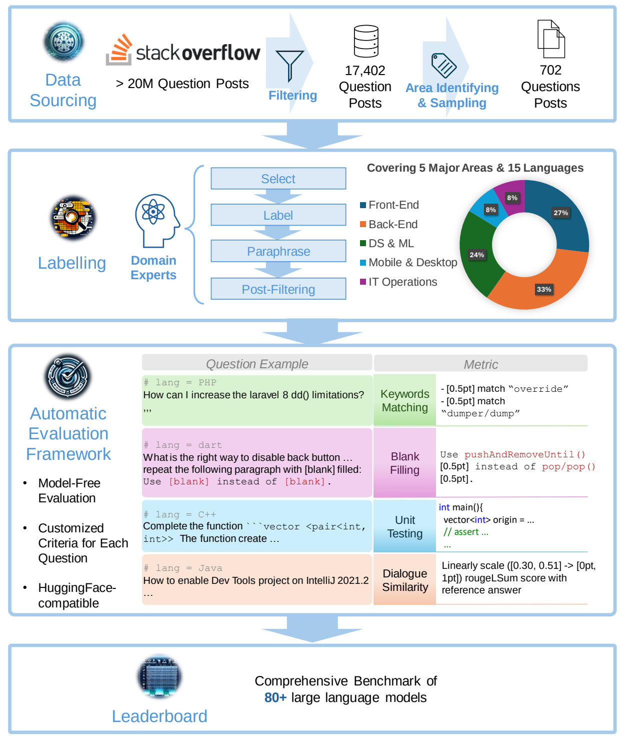

Figure 1: InfiCoder-Eval benchmark overview. We construct InfiCoder-Eval by filtering high-quality and diverse question posts from Stack Overflow and annotating question-level evaluation criteria with domain experts. With an efficient evaluation framework, we evaluate over 80 code LLMs (most extensive for code LLMs to our best knowledge) which leads to several insightful findings.

1. Introduction

In recent years, Large Language Models (LLMs) have been revolutionizing the software development landscape (Hou et al., 2023; Fan et al., 2023), where LLMs are showing exceedingly strong and comprehensive capabilities in comprehending, generating, debugging, and summarizing code (Chen et al., 2021; Li et al., 2022). For example, code LLMs-powered products like GitHub Copilot (Github, 2023) have reached millions of active users within just one year of their launch.

[*]Equal contribution [1]Bytedance Inc. [2]University of Illinois Urbana-Champaign [3]Rutgers University [4]Peking University. Correspondence to: Linyi Li <linyi2@illinois.edu>, Hongxia Yang <hx.yang@bytedance.com>.

Preprint.

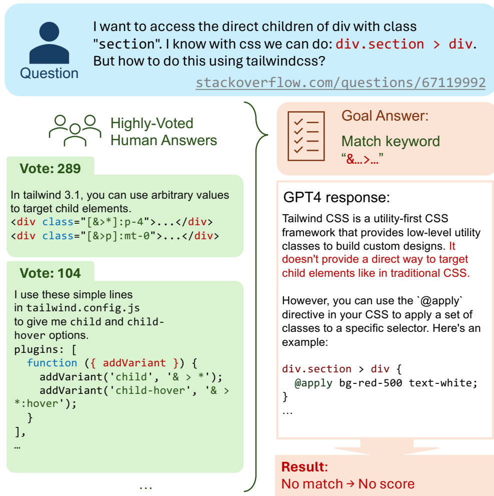

Figure 2: A challenging question paraphrased from Stack Overflow (post #67119992) where GPT-4 fails to answer.

Alongside the huge success of proprietary LLMs such as GPT-3.5 / GPT-4 (OpenAI, 2023) and Gemini (Gemini Team et al., 2023), the development of open-source code LLMs[1] (Nijkamp et al., 2023; Touvron et al., 2023b; Roziere et al., 2023; Luo et al., 2023) has been advancing at an unprecedented fast pace. As of January 2024, the Hugging Face Open LLM Leaderboard (Beeching et al., 2023) has cataloged over 2,000 submissions of such models.

Given the plethora of code LLMs available, the development of reliable code benchmarks seems to lag behind. Benchmarks for code LLMs typically focus on a specific task or domain, often centering on code generation. For example, the widely-used HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) purely focus on Python code generation, and DS-1000 (Lai et al., 2023) concentrates on Python code generation in the field of data science. Though recent efforts have extended code generation benchmarks to include more scenarios, languages (Muennighoff et al., 2023), and tests (Liu et al., 2023a), these extensions often evolve from an existing source (e.g., HumanEval Python problems), thus still lacks diversity. As a result, there is a misalignment between the model's capabilities and evaluated ones which may result in an overemphasis on coding problem-solving in LLM evaluation. Moreover, strong LLMs are saturating existing benchmarks, e.g., GPT-4 has already achieved 88.4% Pass@1 score on HumanEval (Liu et al., 2023a), while in real-world scenarios, GPT-4 can still fail as exemplified in Figure 2. This raises the question: *Can we systematically and comprehensively evaluate code LLMs' abilities in challenging real-world usage scenarios?*

To fulfill this urgent demand, we introduce InfiCoder-Eval, a systematic benchmark for evaluating the free-form question-answering capabilities of code LLMs. The core principle of InfiCoder-Eval is to maximize its representativeness of how developers interact with and utilize such models in real-world scenarios. To achieve this, InfiCoder-Eval comprises 234 questions[2] that are carefully selected and proportionally filtered from the natural high-quality question distribution of Stack Overflow, without any constraints on topics, programming languages, question types, or answer forms. As a result, the curated 234 questions span 15 programming languages and encompass five major areas: *front-end*, *back-end*, *data science and machine learning (DS&ML)*, *mobile and desktop*, and *information technology (IT) operations*.

The expense of such diversity is the increase of evaluation complexity. Unlike code generation or multiple-choice benchmarks, which can be evaluated through standardized methods like unit testing, there is no universal metric for response correctness for free-form questions. On the other hand, model-based evaluations such as those involving GPT-4 are not only costly but also raise concerns about privacy and bias. To mitigate the evaluation challenge, InfiCoder-Eval includes an evaluation framework that integrates four types of model-free metrics: keyword matching, blank filling, unit testing, and dialogue similarity. For each question, we invite domain experts to paraphrase the prompt, select the most appropriate metric, and write down the concrete criteria using domain-specific knowledge, with highly-voted answers from Stack Overflow as a reference. Consequently, the entire benchmark can be evaluated directly in a sandbox environment given any model responses.

As a novel and systematic benchmark disjoint with existing ones in terms of both forms and data sources, we believe that InfiCoder-Eval is an ideal tool to measure existing code LLMs in an objective manner. Hence, we conduct a systematic evaluation for **over 80 code LLMs** using the InfiCoder-Eval framework — the most extensive evaluation for code LLMs to our best knowledge. Our evaluation leads to several insightful findings: (1) On InfiCoder-Eval, GPT-4 achieves a score of 70.64%, being far from perfect but still far exceeding the most capable open-source models as of Jan 2024. On the other hand, GPT3.5 is surpassed by four open-source models. (2) At similar model sizes, coding LLMs are usually visibly stronger than general LLMs; fine-tuning LLMs are usually visibly stronger than base LLMs. (3) The performance differences between different model families are huge, where one model could surpass another with less than 1/10 parameters, highlighting the importance of training data and training techniques. (4) The scaling law is empirically verified for open-source models with fewer than 50B parameters, but not for those with more. Per the prediction of the empirical scaling law, code LLMs can match GPT-4 capabilities at around 70B parameters while generic

[1]We define code LLMs as LLMs that show decent capabilities in the code domain, no matter whether they are exclusively trained with code data or not.

[2]At similar magnitude as HumanEval which has 164 questions.

Table 1: Comparison between InfiCoder-Eval and common existing benchmarks. Existing benchmarks weigh heavily on code generation, unit-test-based evaluation, and a limited set of programming languages. InfiCoder-Eval processes a much higher diversity to reflect real-world code LLMs' usage scenarios and is far from saturation.

Benchmark	Domain	# Question	Evaluation	Data Source	GPT-4 Score
HumanEval (Chen et al., 2021)	Python Programming	164	Test Cases	Hand-Written	88.4%
MBPP (Austin et al., 2021)	Python Programming	974	Test Cases	Hand-Written	81.1%
APPS (Hendrycks et al., 2021)	Python Programming	10,000	Test Cases	Competitions	/ (no report yet)
DS-1000 (Lai et al., 2023)	Python Programming	1,000	Test Cases + Surface Form Constraints	StackOverflow	/ (no report yet)
HumanEval+ (Liu et al., 2023b)	Python Programming	164	Augmented Test Cases	HumanEval	76.2%
HumanEvalPack (Muennighoff et al., 2023)	Repair, Explain, Generation in 6 Languages	2,952	Test Cases	HumanEval	47.8%/52.1%/78.3%
InfiCoder-Eval	Free-Form Code Question Answering in 18 Languages	234	Keyword + Blank Filling + Test Cases + Text Similarity	Stack Overflow	70.64%

LLMs need around 300B parameters. InfiCoder-Eval is fully open source at `https://infi-coder.github.io/inficoder-eval/` and continuously[3] expanding.

2. Benchmark Creation

The InfiCoder-Eval benchmark is created from a high-quality subset of Stack Overflow questions up until June 14, 2023. In this section, we describe the data curation process and the evaluation framework in detail.

2.1. Data Curation

Stack Overflow is a question-and-answer website for developers with more than 20 million registered users as of Jan 2024 (StackExchange, 2024). Since the website is a large collection of natural and diverse coding questions from real-world developers, we believe that questions from Stack Overflow can effectively evaluate code LLM's capabilities in real-world usage scenarios.

The full Stack Overflow dataset contains 23.54 million question posts and 34.68 million answer posts, all transformed to Markdown format. Each question post has a total view count. Each answer post is attached to a question and has a vote count, where website visitors can either give a positive or negative vote to the answer. The question creator can choose one answer as the officially accepted answer.

As we aim to create a benchmark where the correctness evaluation criteria are clear, we view the positively voted answers as an important reference source. Hence, we choose to keep only the questions that have at least three positively voted answers and an officially accepted answer, which turn out to be 1,090,238 questions. For these one million questions, we choose to keep questions that are frequently viewed and relatively new. To fulfill this criterion, we draw a scatter plot of these ≈ 1 million questions, plotting the number of days since their creation until June 14, 2023 (data collection end-date) on the x-axis against the logarithm of their view counts on the y-axis. As shown in Figure 3, we empirically determine to keep questions that lie above the

[3] In other words, infinitely, after which the benchmark is named.

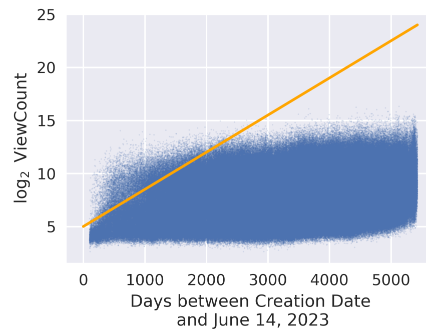

Figure 3: Scatter plot of filtered Stack Overflow questions. We keep the questions above the orange line.

line connecting $(0, 5)$ and $(3000, 15.5)$, resulting in a subset of 17,402 questions.

Utilizing the mandatory question tags of these questions, we then manually construct a tag tree that covers the 200 most frequent tags, enabling us to identify the top programming languages and areas for 14,330 questions. These questions are from 24 programming languages, with each language being categorized into one primary area among the five (front-end, back-end, data science and machine learning (DS&ML), mobile and desktop, and information technology (IT) operations). Lastly, we exclude 6 programming languages that either describe data or are domain-specific: JSON, regex, Markdown, YAML, CSV, and SQL. As a result, we compile 13,854 questions that serve as the *initial seed set*.

2.2. Sampling and Calibration

Based on an user study of developers' demand from our organization, we allocate the tentative area quota to be 25%, 25%, 25%, 15%, and 10% for front-end, back-end, DS&ML, mobile and desktop, and IT operations, respectively. Inspired by HumanEval size and considering the labeling labor cost, we set 200 questions as the target benchmark size. Hence, the tentative size quotas by area are 50, 50, 50, 30, and 20 respectively. We then proportionally distribute the area quotas to language quotas based on the frequency of each language in the initial seed set. However, we ob-

Table 2: Data sampling and calibration statistics for InfiCoder-Eval. From the initial seed set, we uniformly sample and get the inspecting set within each language bin. Domain experts then select high-quality questions according to the language quota and annotate the final benchmark.

Area	Language	Initial Seed Set # Questions	Tentative # Questions Quota	Final InfiCoder-Eval Benchmark # Questions Quota	% Questions Quota	# Area Quota	% Area Quota
Front-End	Javascript	4912	44	44	18.80%	63	26.92%
	CSS	87	10	10	4.27%		
	HTML	600	10	9	3.85%		
Back-End	Java	930	18	17	7.26%	77	32.91%
	C#	629	12	12	5.13%		
	PHP	462	10	9	3.85%		
	Go	117	10	9	3.85%		
	Ruby	71	10	10	4.27%		
	Rust	96	10	10	4.27%		
	C/C++	287	10	10	4.27%		
DS & ML	Python	2779	47	47	20.09%	56	23.93%
	R	184	10	9	3.85%		
Mobile & Desktop	Dart	1562	19	19	8.12%	19	8.12%
	Kotlin	383	10				
	Swift	551	10				
	VBA	16	9				
IT Ops.	Bash	188	21	19	8.12%	19	8.12%
Total		13854	270	234			

serve that following this rule, certain languages such as CSS, C/C++, and VBA end up with fewer than 10 questions, which is insufficient for providing a reliable language sub-score, so, for these languages, we set their quotas to 10.

As a result, we get the *tentative* question quota for each language as shown in Table 2, which sums up to 234 questions. After determining the tentative question quota, we uniformly sample from the initial seed set a roughly two times larger pool for the domain expects to select and annotate.

2.3. Human Annotation

We recruited five domain experts to create the benchmark, each in charge of one area. The annotation process is composed of three steps:

- **Step 1: Question Selection and Type Annotation.** In this step, the domain expert selects high-quality questions from the inspecting set to include into the benchmark and also annotates the question type to be one of the four: code completion, code debugging, config and environment debugging, and knowledge question-answering.
- **Step 2: Prompt Paraphrasing.** In this step, the domain expert paraphrases and simplifies the original question body into succinct and explicit instructions. We include this step for two main purposes: (1) Reduce domain gap. From user-shared conversations collected from ShareGPT, we observe that when interacting with code LLMs, users tend to provide short and direct instructions like "Fix the problem..." and "Debug the code...". However, when posting questions on Stack Overflow, users tend to be lengthy and detailed with courtesy words. We ask the domain experts to paraphrase the question to code LLM user's style without changing the semantics. (2) Prevent memorization. Some code LLMs may be trained or fine-tuned with Stack Overflow data. Paraphrasing the questions can help to mitigate the result advantages of these models.
- **Step 3: Correctness Criterion Annotation.** In this step, the domain expert chooses one or multiple evaluation metrics from our framework and annotates the detailed correctness criterion (see Section 2.4) in a domain-specific language. External files can be attached if needed, e.g., unit tests and reference answers.

Table 3: InfiCoder-Eval statistics.

(a) Question type.

Question Type	Ratio
Code Completion	30.37%
Knowledge Question-Answering	27.04%
Code Debugging	26.67%
Config & Environment Debugging	15.93%

(b) Metric type.

Metric Type	Ratio
Keywords Matching	57.41%
Blank Filling	12.22%
Unit Testing	19.26%
Dialogue Similarity	11.85%

(c) Prompt token length with Code Llama tokenizer.

min	25% quantile	median	mean	75% quantile	max
43	145.75	223	338.46	359.50	5047

To mitigate individual discrepancy, we introduce a few checkpoints for domain experts to read others' annotated cases, discuss, and reach consensus for controversial cases.

Post-Annotation Correction and Filtering. After the 270 tentative questions were annotated, we ran an initial evaluation of all these questions on over 30 code LLMs. From the evaluation, we observe that for many questions whose scores are computed based on dialogue similarity (see Section 2.4), the similarity scores cannot differentiate model responses much, where correct answers have many different forms. Hence, we decided to remove seven questions exhibiting this and all questions from Kotlin, Swift, and VBA languages since more than half of their questions exhibit this phenomenon. From the evaluation, We also identified a few wrong criteria specifications and manually fixed them. After this process, the final benchmark contains 234 questions spanning over 15 languages. Their detail statistics are shown in Table 2.

2.4. Evaluation Framework and Score Computing

In response to the diversified questions, InfiCoder-Eval evaluation framework integrates four model-free metric types: keywords matching, blank filling, unit testing, and dialogue similarity. Domain experts choose one or multiple metric types along with their corresponding weights and concretize.

- **Keywords Matching.** Though the responses can be in diverse forms, for a significant portion of benchmark questions, we find that the existence of some keywords strongly determines the quality of the response. We allow domain experts to write rules that match keywords and regular expressions or construct recursive logical expressions on top of keyword-matching results. When multiple keywords exist, each matching result can have its own weight in the final score.

- **Blank Filling.** For some questions, it is challenging to measure the correctness given the response uncertainty. In this case, domain experts can instruct the model to answer the question by following a given template and filling in the blanks in the template. The blanks can correspond to either natural language or code snippet. Then, similar to keywords matching, each blank can match potential keywords, regular expressions, or recursive logic expressions built upon matching results. This metric type tests not only the model's QA ability but also its instruction-following ability.
- **Unit Testing.** For code-intensive questions, we can follow the traditional benchmarks to evaluate the response correctness by unit tests. For this type, domain experts add detailed requirements to allow for unit-test-based evaluation, such as requirements on generated function name, input arguments, and output format. Besides the test, domain experts can further import the context setup script and context cleanup script.
- **Dialogue Similarity.** For natural-language-intensive questions, domain experts can extract and shorten the reference answers from Stack Overflow, and then use the ROUGE score (Lin, 2004) to evaluate the response similarity with reference answers. The ROUGE score was initially proposed and widely used in evaluating the quality of text summarization and machine translation. To map the ROUGE score back to our benchmark scale, we allow domain experts to tune the mapping interval and scores within the interval are then linearly mapped to our score scale.

The example questions and corresponding criteria are illustrated in Figure 1. Detail statistics of metric type ratios, question type ratios, and prompt length are shown in Table 3.

Score Computation. We treat each question equally with one point each. Given 234 questions in the benchmark, the full score is 234, and we by default report the percentage score (achieved score divided by 234) unless otherwise noted. The one point for each question can be further decomposed into a few scoring points within each question. For example, a question may contain four keywords with weights 2, 1, 1, and 1 each. Then, matching each keyword can contribute to 0.4, 0.2, 0.2, and 0.2 points respectively to the final score.

Difficulty Groupings. We systematically evaluated GPT-4 and GPT-3.5-turbo on the benchmark following the evaluation protocol in Section 3.1, based on which we classify the benchmark questions into five disjoint difficulty groups.

- Level 1 (93 questions): GPT-3.5-turbo can achieve a mean score ≥ 0.5.
- Level 2 (55 questions): Among the rest questions, those where GPT-4 can achieve a mean score ≥ 0.5.
- Level 3 (44 questions): Among the rest questions, those where GPT-4 with sampling temperature 1.0 can achieve a maximum score ≥ 0.5 among 10 trials.
- Level 4 (18 questions): Among the rest questions, those GPT-4 with sampling temperature 0.2 can achieve a positive score among 100 trials.
- Level 5 (24 questions): The remaining questions.

The full result table (Table 7) shows each code LLM's score in each difficulty group. We observe that the scores roughly decreases for higher difficulty levels, justifying our level assignment. We can conduct further studies according to the difficulty grouping.

Framework Implementation. We have implemented an automated evaluation framework supporting all 234 benchmark questions with Python. Specifically, for blank-filling evaluation, we implement longest common subsequence matching via dynamic programming to capture the filled blanks in the response. For unit-testing evaluation, we support the unit test execution for nine languages. Specifically, the Javascript support is based on `node.js` (with Typescript support); the C# support is based on the `Mono` framework.

2.5. Comparison with Existing Benchmarks

In Table 1, we compare our InfiCoder-Eval benchmark with several existing benchmarks for code LLMs. As reflected in the table, our benchmark strongly complements existing ones by providing a much higher level of diversity from both the question and evaluation aspects. Moreover, measured by GPT-4 score, InfiCoder-Eval is not saturated yet. On the other hand, the benchmark is limited in size due to the high cost of correctness criteria labeling, and we are working on continuously expanding the benchmark.

Discussion. During the benchmark creation process, we did not explicitly introduce a data decontamination process. The reason is that we believe it is not always feasible to detect or prevent the same data source from being used for training by existing or future models. If the same data source (Stack Overflow) is used for training, achieving a full comprehension level would theoretically be able to solve this benchmark completely. Instead of viewing this as a threat to benchmark validness, we view achieving a full comprehension or information retrieval ability on such a large data source (over 20M questions) is itself great progress in LLM research, which also, e.g., opens a venue for benchmarking retrieval-augmented generation (RAG) for LLMs.

3. Evaluation and Leaderboard

We systematically evaluate 88 proprietary models and open-source models on the InfiCoder-Eval benchmark.

3.1. Evaluation Protocol

We adopt best@10 as the main evaluation metric, where 10 responses are sampled and evaluated for each question and the best score per question is recorded and summed up. Throughout the evaluation, we set sampling temperature $T = 0.2$ and top $p = 0.9$.

We also conducted a comprehensive evaluation of other generation parameters with GPT4, and find that for maximizing the performance under best@10, the best temperature is $T = 1.0$ and the top $p = 0.9$, leading to a score of $76.15\% \pm 0.21\%$. In particular, the temperature T affects much and the effect of top p is minor. We decided to stick to the original parameter $T = 0.2$ and $p = 0.9$ in the main evaluation as this setting is more akin to the real-world scenario where user generates once with low temperature.

We use the system prompt "`You are a professional assistant for programmers. By default, questions and answers are in Markdown format.`" for normal questions, and the system prompt "`You are a professional assistant for programmers. By default, questions and answers are in Markdown format. You are chatting with programmers, so please answer as briefly as possible.`" for questions evaluated by the dialogue similarity metric to encourage short answers. For generic models, we generate the prompt with "`{system prompt}\n{content prompt}`" format; for instruction-finetuned or chat models, we generate the prompt with their own prompt templates.

For proprietary models, we focus on OpenAI models GPT-4, GPT-3.5(-turbo), and Davinci-002 at the current stage. The API version date is fixed to June 13, 2023. We did not specify the max tokens to generate and found out that the longest response generated by GPT-4 has 662 tokens with Code Llama tokenizer. We repeat each evaluation three times and report the error bars.

For open-source models, we evaluate on an 8xA100 server with our forked version of `https://github.com/bigcode-project/bigcode-evaluation-harness`. For models with over 30B parameters, due to the GPU memory limit and efficiency concerns, we impose the longest context constraint of 4,096 tokens and conduct the experiment just once. Since there is only one question whose GPT-4 context (prompt + GPT-4 response) can exceed 4,096 tokens, we think this context constraint has little effect, reducing the score by 0.37% at most. For models within 30B parameters, since GPT-4 response has at most 662 tokens, we set the max number of tokens to generate to be $\min\{1024, \text{context length - prompt length}\}$, providing some wiggle room. Meanwhile, we repeat the evaluation three times for models within 30B parameters.

Table 4: An aggregated version of InfiCoder-Eval leaderboard where the best model within each model family is presented. Evaluation protocol in Section 3.1. The "Size" column records number of parameters. For MoE models, "total parameters / used parameters during inference" is recorded. Bar colors stand for General Base, General Finetuned, Code Base, and Code Finetuned models respectively. Full results in Table 6.

Family	Best Model Name	Size	InfiCoder-Eval Score
GPT4	GPT4-0613	?	70.64% ± 0.82%
DeepSeek Coder	deepseek-coder-33b-instruct	33B	62.96%
Phind	Phind-CodeLlama-34B-v2	34B	59.00%
DeepSeek LLM	deepseek-llm-67b-chat	67B	57.41%
GPT3.5	GPT-3.5-turbo-0613	?	56.47% ± 1.34%
Mixtral	mixtral-8x7B-Instruct	46.7B / 12.9B	55.55%
Qwen	Qwen-72B	72B	55.34%
Magicoder	Magicoder-S-CL-7B	7B	52.71% ± 0.72%
WizardLM	WizardCoder-Python-34B-V1.0	34B	52.59%
Code Llama	CodeLlama-34b-Instruct	34B	50.45%
01.AI	Yi-34B-Chat	34B	49.58%
Zephyr	Zephyr 7B beta	7B	46.31% ± 1.11%
DeepSeek MoE	deepseek-moe-16b-chat	16B / 2.8B	45.18% ± 1.65%
OctoPack	OctoCoder	15.5B	44.55% ± 0.79%
Llama 2	Llama2-70B-Chat	70B	39.30%
Mistral	Mistral-7B-Instruct-v0.1	7B	37.55% ± 1.10%
InternLM	InternLM-Chat-20B	20B	37.41% ± 0.75%
DeepSeek LLM	deepseek-llm-7b-chat	7B	36.75% ± 1.40%
Baichuan2	Baichuan2-13B-Chat	13B	34.40% ± 1.34%
Code Llama	CodeLlama-7b-Python	7B	32.89% ± 0.45%
StarCoder	StarCode+	15.5B	30.67% ± 1.57%
CodeGen2.5	CodeGen2.5-7B-Instruct	7B	29.57% ± 1.53%
ChatGLM	ChatGLM3-6B	6B	28.23% ± 0.58%
davinci	davinci-002	?	21.25% ± 1.17%
Phi	Phi1.5	1.5B	20.56% ± 0.09%
CodeGeeX	CodeGeeX2-6B	6B	19.88% ± 0.36%
CodeGen2	CodeGen2-16B	16B	16.97% ± 1.15%
IEITYuan	Yuan2-51B-hf	51B	15.25%
CodeGen	CodeGen-16B-multi	16B	13.62% ± 1.18%

3.2. Leaderboard

We present the full results in Table 6 in the appendix and an aggregated table in Table 4 where the best model among each model family is digested. The results are also presented as a scatter plot in Figure 4. In the figure, normal models are shown as scatters with error bars, MoE models are shown as horizontal segments with error ranges connecting the used parameters during inference and total number of parameters, and OpenAI proprietary models are shown as horizontal lines with error ranges.

In both tables and the figure, we classify LLMs by general/code and base/finetuned. The general LLMs are claimed to have strong capabilities beyond code, e.g., in various natural language tasks, while the code LLMs are exclusively optimized for the code domain. The based LLMs only went through the pertaining phase, while the finetuned LLMs are claimed to have instruction-following capabilities or are finetuned.

4. Analysis and Discussion

We summarize our findings below.

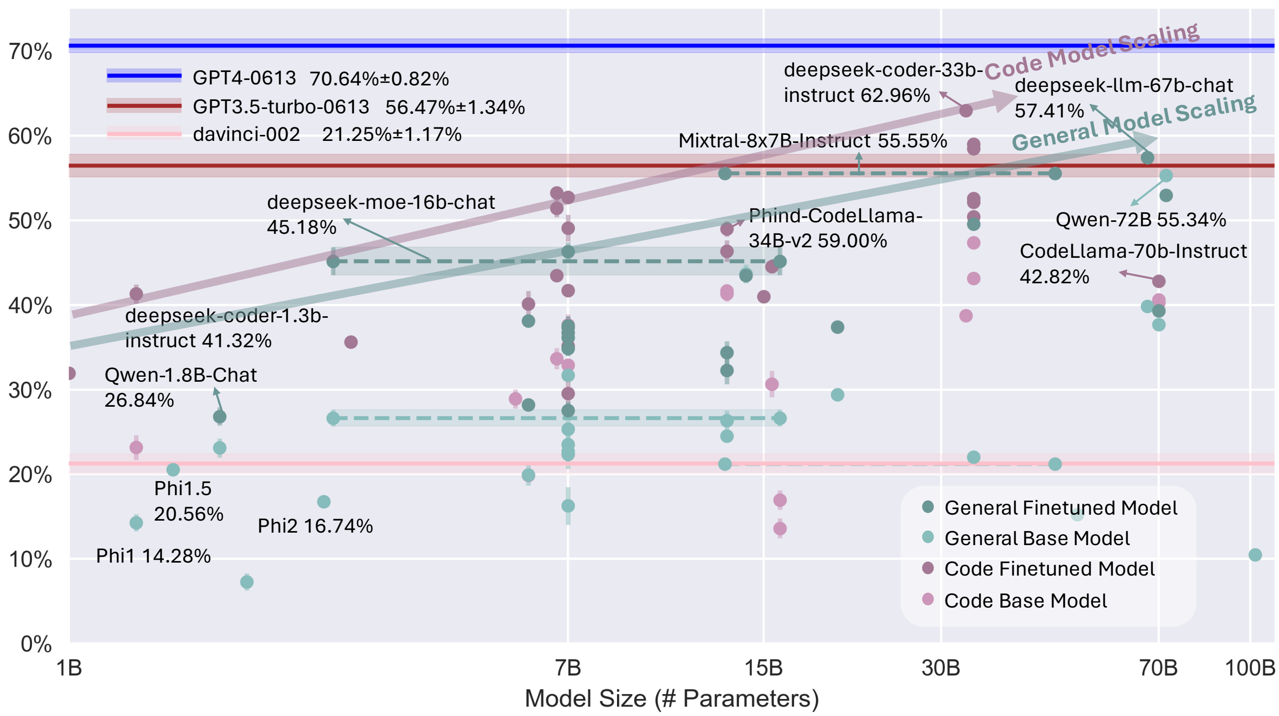

Figure 4: Scatter plot for all evaluated LLMs on InfiCoder-Eval. Normal models are shown as scatters with error bars, MoE models are shown as horizontal segments with error ranges connecting the used parameters during inference and total number of parameters, and OpenAI proprietary models are shown as horizontal lines with error ranges. Projected empirical scaling laws for both general and code models are drawn. See detailed discussion in Section 4.

Best open-source models are competitive but still far from GPT-4. As expected, GPT-4 achieves the highest score 70.64%. The runner score is achieved by an open-source model, deepseek-coder-33b-instruct (DeepSeekAI, 2023), with a 62.96% score. The result implies that: (1) GPT-4 is still far from perfect. Noting that the full score of InfiCoder-Eval is 100%, even the powerful GPT-4 is still far from perfect, which is in contrast to the near 90% rate in HumanEval. We inspect the score breakdown. For the two most frequent metric types, keywords matching and unit testing, GPT-4 achieves similar scores 66.61% and 76.00% respectively. For blank filling, the score is relatively lower at 58.08%. These scores may imply that GPT-4 may still lack generic ability in answering diversified real-world questions related to code. When being instructed to follow a given template to answer (blank filling), due to the more strict requirement and narrower solution space, such ability shortage becomes more pronounced. (2) There is still a visible gap between open-source models and GPT-4. The gap between deepseek-coder-33b-instruct and GPT-4 is roughly 8 points. Hence, GPT-4 is still the best LLM to our best knowledge in InfiCoder-Eval. However, noticing that GPT-3.5-turbo achieves 56.47%, open-source models are now reliably better than GPT-3.5-turbo, lying between GPT-3.5-turbo and GPT-4 (slightly closer to GPT-3.5-turbo) end.

Among open-source models, different models have very different performances. Figure 4 systematically visualizes the performance of different open-source models at diverse scales. Although there is a general tendency that larger models achieve higher scores, the scores among different models at the similar scale differ largely. For example, at the scale 7B, the best-performing model is at around 55%, pretty close to GPT3.5, while the low-performing model stays at around 15%. Moreover, consider the best model at scale 1.3B, deepseek-coder-1.3b-instruct, which achieves 41.32%, it even surpasses a few models at the scale 70B or 100B. This result implies that though scaling matters, the training techniques and training data are equally important or even more, helping to reduce the required scale for achieving certain score by more than 10× size.

Instruction-finetuning is important for QA. Among models of similar scales and the same family, we find that the best-performing ones almost always include an instruction-finetuning phase, such as deepseek-llm-67b-chat, deepseek-coder-33b-instruct, CodeLlama-34B-Instruct, and Qwen-18B-Chat. In contrast, the pretraining models, such as davinci-002 and phi models, usually perform poorly despite

Table 5: The full evaluation of Code Llama (Roziere et al., 2023) models showcases intense single-language finetuning may hurt free-form QA abiltiy, despite achieving higher HumanEval scores (compare "Base" and "Python" columns).

	Benchmark	Base	Python	Instruct
7B	HumanEval	33.5%	38.4% (+4.9%)	34.8% (+1.3%)
	InfiCoder-Eval	$37.62\%_{\pm 1.28\%}$	$32.89\%_{\pm 0.45\%}$ (−4.73%)	$35.15\%_{\pm 1.28\%}$ (−2.47%)
13B	HumanEval	36.0%	43.3% (+7.3%)	42.7% (+6.7%)
	InfiCoder-Eval	$41.66\%_{\pm 0.84\%}$	$41.31\%_{\pm 0.90\%}$ (−0.35%)	$46.37\%_{\pm 1.26\%}$ (+4.71%)
34B	HumanEval	48.8%	53.7% (+4.9%)	41.5% (−7.3%)
	InfiCoder-Eval	47.36%	43.13% (−4.23%)	50.45% (+3.09%)
70B	HumanEval	48.8%	53.7% (+4.9%)	41.5% (−7.3%)
	InfiCoder-Eval	40.60%	40.29% (−0.31%)	42.82% (+2.22%)

their capacities and strong performances in code generation benchmarks. This implies that instruction-finetuning is critical for equipping the models with QA ability in the code domain.

Some models may focus too much on code generation, especially the small ones. In Table 6, we observe that for large models (>30B) and top entries, the InfiCoder-Eval scores and HumanEval pass1 scores coincide well. However, for smaller models, the score tendencies start to diverge, where some models are relatively stronger in InfiCoder-Eval (Zypher-7b-β) and more are relatively stronger in HumanEval (OctoCoder, Qwen-14B-Chat, phi-1.5, CodeGeeX2). This phenomenon may imply that a few models may focus heavily on simpler code generation benchmarks while ignoring the performance in generic code scenarios. Our InfiCoder-Eval benchmark, as a free-form QA benchmark in the code domain, is a great tool for detecting and evaluating such imbalance in model ability.

Furthermore, we conduct a complete evaluation for all Code Llama models. As shown in Table 5, we found finetuning on Python data improves HumanEval scores but consistently hurts InfiCoder-Eval scores, while instruction finetuning usually improves InfiCoder-Eval scores but may hurt HumanEval scores.

Code models and general models may exhibit different scaling laws, and open-source models scale well only within 50B. In Figure 4, we connect the top-performing code and general models respectively to predict within which size the models are on par with the strongest proprietary model, GPT-4. As we can observe, for general models that need to cover a broad range of capabilities, the model size may need to be around 300B (note the logarithm scale of the x-axis); while for code models, the model size may only need to be around 70B which is much closer. Hence, for strong code capabilities for developers' use, maybe training exclusively or heavily in the code domain is a better choice compared to building a general model.

Another important finding is that among all open-source models benchmarked so far, models larger than 50B do not perform significantly better than those within 50B. For example, among general models, deepseek-llm-67b-chat achieves 57.41%, which is just 2% higher than Mixtral-8x7B-Instruct which is within 50B. More astonishingly, CodeLlama-70b models are even poorer than CodeLlama-34b counterparts. This is a sharp contrast to the scaling pace within 50B, where models have significantly better performance from 1B to 3B, from 3B to 7B, ..., until 50B. This may imply that there may be some non-trivial barrier when scaling the model beyond 50B, or the scaling law may change at such a large scale.

5. Related Work

Large language models (Vaswani et al., 2017; Devlin et al., 2018; Brown et al., 2020) are revolutionizing people's lives. Especially, in the coding domain, code LLMs (Chen et al., 2021; Li et al., 2022) are shown to be capable of completing a wide range of tasks such as code generation, debugging, and question-answering. Recently, code LLMs are booming with new models, including both proprietary ones (Github, 2023; OpenAI, 2023) and open-source ones (Beeching et al., 2023; Nijkamp et al., 2023; Touvron et al., 2023a;b; Li et al., 2023; Luo et al., 2023; Roziere et al., 2023), are released almost every month.

At the same time, benchmarks for code LLMs are developing, though at a relatively slower pace. Common benchmarks (Hendrycks et al., 2021; Austin et al., 2021; Chen et al., 2021) focus on code generation and unit-test-based evaluation. Recent efforts augment these benchmarks by language translation (Athiwaratkun et al., 2023; Zheng et al., 2023), test augmentation (Liu et al., 2023b), and task generalization (Muennighoff et al., 2023). In contrast, our InfiCoder-Eval benchmark is built for evaluating free-form question-answering ability in the code domain which is essential for code LLMs as developers' assistants. InfiCoder-Eval benchmark is a strong complement of existing benchmarks.

6. Conclusion and Future Work

We proposed InfiCoder-Eval, a systematic benchmark for evaluating the question-answering ability of large language models for code. InfiCoder-Eval comprises 234 high-quality questions from Stack Overflow and supports automatic execution and evaluation of model responses with four types of model-free metrics such as unit testing and keywords matching. A comprehensive evaluation of over 80 code LLMs reveals several interesting findings and takeaways. The benchmark and evaluation framework will be made publicly available, and we will continue to maintain and expand this benchmark.

Impact Statements

In this work, we propose an evaluation framework InfiCoder-Eval for code LLMs to systematically evaluate code LLM's capabilities in answering and assisting developers' questions. We anticipate that the framework may be widely adopted and used to measure future code LLM capabilities. Hence, we would like to address that the benchmark is mainly designed to solely focus on evaluating model capabilities. We did not consider model alignment and trustworthiness (e.g., hallucinations, privacy, etc) evaluation in the evaluation criterion. As a result, the evaluation may not fully reflect LLM's helpfulness, and we strongly suggest practitioners to evaluate LLMs using systematic criteria integrating multiple benchmarks to cover alignment aspects.

References

Athiwaratkun, B., Gouda, S. K., Wang, Z., Li, X., Tian, Y., Tan, M., Ahmad, W. U., Wang, S., Sun, Q., Shang, M., Gonugondla, S. K., Ding, H., Kumar, V., Fulton, N., Farahani, A., Jain, S., Giaquinto, R., Qian, H., Ramanathan, M. K., Nallapati, R., Ray, B., Bhatia, P., Sengupta, S., Roth, D., and Xiang, B. Multi-lingual evaluation of code generation models. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=Bo7eeXm6An8`.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Beeching, E., Fourrier, C., Habib, N., Han, S., Lambert, N., Rajani, N., Sanseviero, O., Tunstall, L., and Wolf, T. Open llm leaderboard. `https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard`, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

DeepSeekAI. Deepseek coder: Let the code write itself. `https://deepseekcoder.github.io/`, 2023.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., and Zhang, J. M. Large language models for software engineering: Survey and open problems. *arXiv preprint arXiv:2310.03533*, 2023.

Gemini Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Github. Github copilot - your ai pair programmer. `https://github.com/features/copilot`, 2023.

Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., Burns, C., Puranik, S., He, H., Song, D., et al. Measuring coding challenge competence with apps. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., and Wang, H. Large language models for software engineering: A systematic literature review. *arXiv preprint arXiv:2308.10620*, 2023.

Lai, Y., Li, C., Wang, Y., Zhang, T., Zhong, R., Zettlemoyer, L., Yih, W.-t., Fried, D., Wang, S., and Yu, T. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pp. 18319–18345. PMLR, 2023.

Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Lin, C.-Y. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pp. 74–81, 2004.

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

Liu, J., Xia, C. S., Wang, Y., and Zhang, L. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *arXiv preprint arXiv:2305.01210*, 2023b.

Luo, Z., Xu, C., Zhao, P., Sun, Q., Geng, X., Hu, W., Tao, C., Ma, J., Lin, Q., and Jiang, D. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*, 2023.

Muennighoff, N., Liu, Q., Zebaze, A., Zheng, Q., Hui, B., Zhuo, T. Y., Singh, S., Tang, X., von Werra, L., and Longpre, S. Octopack: Instruction tuning code large language models. *arXiv preprint arXiv:2308.07124*, 2023.

Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S., and Xiong, C. Codegen: An open large language model for code with multi-turn program synthesis. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=iaYcJKpY2B_.

OpenAI. Gpt-4 technical report. *OpenAI*, 2023. URL https://cdn.openai.com/papers/gpt-4.pdf.

Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

StackExchange. All sites — stackexchange. 2024. URL https://stackexchange.com/sites?view=list#users.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Zheng, Q., Xia, X., Zou, X., Dong, Y., Wang, S., Xue, Y., Wang, Z., Shen, L., Wang, A., Li, Y., Su, T., Yang, Z., and Tang, J. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. In *KDD*, 2023.

Table 6: Overall results of all benchmarked LLMs. Evaluation protocol in Section 3.1. HumanEval Scores are digested from (Liu et al., 2023a). Bar colors stand for General Base, General Finetuned, Code Base, and Code Finetuned models respectively.

No	Family	Model Name	Domain	Type	Size	Context Length	InfiCoder-Eval Score	HumanEval Score
1	GPT4	GPT4-0613	General	Finetuned	?	8192	70.64% ± 0.82%	88.4
2	GPT4	GPT4-turbo-1106	General	Finetuned	?	8192	68.42% ± 0.38%	85.4
3	DeepSeek Coder	deepseek-coder-33b-instruct	Code	Finetuned	33B	16384	62.96%	80.02
4	Phind	Phind-CodeLlama-34B-v2	Code	Finetuned	34B	4096	59.00%	71.95
5	Phind	Phind-CodeLlama-34B-v1	Code	Finetuned	34B	4096	58.47%	65.85
6	DeepSeek LLM	deepseek-llm-67b-chat	General	Finetuned	67B	4096	57.41%	/
7	GPT3.5	GPT-3.5-turbo-0613	General	Finetuned	?	4096	56.47% ± 1.34%	72.6
8	Mixtral	mixtral-8x7B-Instruct	General	Finetuned	46.7B / 12.9B	32768	55.55%	37.8
9	Qwen	Qwen-72B	General	Base	72B	32768	55.34%	/
10	DeepSeek Coder	deepseek-coder-6.7b-instruct	Code	Finetuned	6.7B	16384	53.25% ± 0.40%	80.22
11	Qwen	Qwen-72B-Chat	General	Finetuned	72B	32768	52.97%	/
12	Magicoder	Magicoder-S-CL-7B	Code	Finetuned	7B	16384	52.71% ± 0.72%	70.7
13	WizardLM	WizardCoder-Python-34B-V1.0	Code	Finetuned	34B	16384	52.59%	70.73
14	Phind	Phind-CodeLlama-34B-Python-v1	Code	Finetuned	34B	4096	52.17%	70.22
15	Magicoder	Magicoder-S-DS-6.7B	Code	Finetuned	6.7B	16384	51.46% ± 1.09%	76.8
16	Code Llama	CodeLlama-34b-Instruct	Code	Finetuned	34B	16384	50.45%	50.79
17	01.AI	Yi-34B-Chat	General	Finetuned	34B	4096	49.58%	/
18	WizardLM	WizardCoder-Python-7B-V1.0	Code	Finetuned	7B	16384	49.10% ± 1.59%	48.2
19	WizardLM	WizardCoder-Python-13B-V1.0	Code	Finetuned	13B	16384	48.99% ± 0.92%	62.19
20	Code Llama	CodeLlama-34b	Code	Base	34B	16384	47.36%	45.11
21	Code Llama	CodeLlama-13b-Instruct	Code	Finetuned	13B	16384	46.37% ± 1.26%	50.6
22	Zephyr	Zephyr 7B beta	General	Finetuned	7B	32768	46.31% ± 1.11%	/
23	DeepSeek MoE	deepseek-moe-16b-chat	General	Finetuned	16B / 2.8B	16384	45.18% ± 1.65%	/
24	OctoPack	OctoCoder	Code	Finetuned	15.5B	8192	44.55% ± 0.79%	45.3
25	Qwen	Qwen-14B	General	Base	14B	8192	43.69% ± 1.09%	/
26	Qwen	Qwen-14B-Chat	General	Finetuned	14B	8192	43.49% ± 0.63%	40.9
27	Magicoder	Magicoder-DS-6.7B	Code	Finetuned	6.7B	16384	43.47% ± 0.21%	/
28	Code Llama	CodeLlama-34b-Python	Code	Base	34B	16384	43.13%	53.29
29	Code Llama	CodeLlama-70b-Instruct	Code	Finetuned	70B	4096	42.82%	75.6
30	Magicoder	Magicoder-CL-7B	Code	Finetuned	7B	16384	41.71% ± 0.76%	/
31	Code Llama	CodeLlama-13b	Code	Base	13B	16384	41.66% ± 0.84%	35.07
32	DeepSeek Coder	deepseek-coder-1.3b-instruct	Code	Finetuned	1.3B	16384	41.32% ± 1.12%	64.6
33	Code Llama	CodeLlama-13b-Python	Code	Base	13B	16384	41.31% ± 0.90%	42.89
34	WizardLM	WizardCoder-15B-V1.0	Code	Finetuned	15B	2048	41.01% ± 0.22%	58.12
35	Code Llama	CodeLlama-70b	Code	Base	70B	4096	40.60%	55.5
36	Code Llama	CodeLlama-70b-Python	Code	Base	70B	4096	40.29%	55.49
37	OctoPack	OctoGeeX	Code	Finetuned	6B	8192	40.14% ± 1.55%	42.28
38	DeepSeek LLM	deepseek-llm-67b-base	General	Base	67B	4096	39.87%	42.7
39	Llama 2	Llama2-70B-Chat	General	Finetuned	70B	4096	39.30%	/
40	DeepSeek Coder	deepseek-coder-33b-base	Code	Base	33B	16384	38.75%	52.45
41	01.AI	Yi-6B-Chat	General	Finetuned	6B	4096	38.14% ± 0.58%	/
42	Llama 2	Llama2-70B	General	Base	70B	4096	37.69%	28.7
43	Code Llama	CodeLlama-7b	Code	Base	7B	16384	37.62% ± 1.28%	29.98
44	Mistral	Mistral-7B-Instruct-v0.1	General	Finetuned	7B	32768	37.55% ± 1.10%	/
45	InternLM	InternLM-Chat-20B	General	Finetuned	20B	16384	37.41% ± 0.75%	/
46	Qwen	Qwen-7B-Chat	General	Finetuned	7B	32768	37.36% ± 1.29%	36.0
47	DeepSeek LLM	deepseek-llm-7b-chat	General	Finetuned	7B	4096	36.75% ± 1.40%	/
48	Llama 2	Llama2-7B-Chat	General	Finetuned	7B	4096	36.14% ± 1.05%	/
49	WizardLM	WizardCoder-3B-V1.0	Code	Finetuned	3B	2048	35.61% ± 0.42%	32.92
50	Code Llama	CodeLlama-7b-Instruct	Code	Finetuned	7B	16384	35.15% ± 1.02%	45.65
51	InternLM	InternLM-Chat-7B	General	Finetuned	7B	8192	34.86% ± 0.90%	/
52	Baichuan2	Baichuan2-13B-Chat	General	Finetuned	13B	4096	34.40% ± 1.34%	19.5
53	DeepSeek Coder	deepseek-coder-6.7b-base	Code	Base	6.7B	16384	33.66% ± 1.24%	45.83
54	Code Llama	CodeLlama-7b-Python	Code	Base	7B	16384	32.89% ± 0.45%	40.48
55	Llama 2	Llama2-13B-Chat	General	Finetuned	13B	4096	32.29% ± 1.66%	/
56	WizardLM	WizardCoder-1B-V1.0	Code	Finetuned	1B	2048	31.94% ± 0.70%	23.17
57	Qwen	Qwen-7B	General	Base	7B	32768	31.69% ± 0.29%	/
58	StarCoder	StarCode+	Code	Base	15.5B	8192	30.67% ± 1.57%	/
59	StarCoder	StarCoder	Code	Base	15.5B	8192	30.66% ± 0.69%	33.57
60	CodeGen2.5	CodeGen2.5-7B-Instruct	Code	Finetuned	7B	2048	29.57% ± 1.53%	/
61	InternLM	InternLM-20B	General	Base	20B	16384	29.41% ± 0.76%	/
62	DeepSeek Coder	deepseek-coder-5.7bmqa-base	Code	Base	5.7B	16384	28.92% ± 1.12%	/
63	ChatGLM	ChatGLM3-6B	General	Finetuned	6B	8192	28.23% ± 0.58%	52.4
64	Baichuan2	Baichuan2-7B-Chat	General	Finetuned	7B	4096	27.53% ± 1.07%	17.7
65	Qwen	Qwen-1.8B-Chat	General	Finetuned	1.8B	32768	26.84% ± 1.08%	/
66	DeepSeek MoE	deepseek-moe-16b-base	General	Base	16B / 2.8B	16384	26.65% ± 0.97%	/
67	Baichuan2	Baichuan2-13B-Base	General	Base	13B	4096	26.32% ± 1.23%	/
68	DeepSeek LLM	deepseek-llm-7b-base	General	Base	7B	4096	25.34% ± 1.08%	26.2
69	Llama 2	Llama2-13B	General	Base	13B	4096	24.50% ± 0.73%	/
70	Baichuan2	Baichuan2-7B-Base	General	Base	7B	4096	23.50% ± 1.56%	/
71	DeepSeek Coder	deepseek-coder-1.3b-base	Code	Base	1.3B	16384	23.17% ± 1.47%	32.13
72	Qwen	Qwen-1.8B	General	Base	1.8B	32768	23.12% ± 1.13%	/
73	Mistral	Mistral-7B-v0.1	General	Base	7B	32768	22.72% ± 1.51%	28.7
74	Llama 2	Llama2-7B	General	Base	7B	4096	22.35% ± 1.70%	14.6
75	01.AI	Yi-34B	General	Base	34B	4096	22.01%	/
76	davinci	davinci-002	General	Base	?	16384	21.25% ± 1.17%	/
77	Mixtral	mixtral-8x7B	General	Base	46.7B / 12.9B	32768	21.21%	/
78	Phi	Phi1.5	General	Base	1.5B	2048	20.56% ± 0.09%	/
79	01.AI	Yi-6B	General	Base	6B	4096	19.93% ± 1.24%	/
80	CodeGeeX	CodeGeeX2-6B	Code	Base	6B	8192	19.88% ± 0.36%	33.49
81	CodeGen2	CodeGen2-16B	Code	Base	16B	2048	16.97% ± 1.15%	/
82	Phi	Phi2	General	Base	2.7B	2048	16.74% ± 0.64%	48.2
83	InternLM	InternLM-7B	General	Base	7B	8192	16.26% ± 2.21%	/
84	IEITYuan	Yuan2-51B-hf	General	Base	51B	4096	15.25%	/
85	Phi	Phi1	General	Base	1.3B	2048	14.28% ± 0.99%	51.22
86	CodeGen	CodeGen-16B-multi	Code	Base	16B	2048	13.62% ± 1.18%	19.26
87	IEITYuan	Yuan2-102B-hf	General	Base	102B	4096	10.48%	/
88	IEITYuan	Yuan2-2B-hf	General	Base	2B	8192	7.28% ± 1.01%	/

Table 7: Scores of all benchmarked LLMs by difficulty levels, problem types, and evaluation metrics.

No	Family	Model Name	Overall	Difficulty Levels					Problem Types				Evaluation Metrics			
			InfiCoder-Eval Score	Level 1	Level 2	Level 3	Level 4	Level 5	Code Completion	Code Debugging	Knowledge QA	Config & Env Debugging	Keyword Matching	Unit Testing	Blank Filling	Text Similarity
		max	70.64%	93.08%	92.48%	54.16%	31.91%	17.36%	75.23%	69.74%	68.55%	66.63%	66.61%	77.00%	58.08%	84.27%
		mean	36.78%	55.88%	36.87%	24.32%	9.91%	5.50%	37.76%	35.41%	40.78%	31.85%	37.20%	36.30%	19.09%	22.78%
		min	7.28%	9.11%	7.77%	5.56%	0.99%	0.00%	4.01%	8.29%	6.71%	4.76%	8.41%	3.33%	0.00%	0.00%
1	GPT4	GPT4-0613	70.64%	92.31%	92.48%	51.90%	31.91%	0.00%	75.23%	69.74%	68.55%	66.63%	66.61%	76.00%	58.08%	84.27%
2	GPT4	GPT4-turbo-1106	68.42%	89.90%	78.57%	54.16%	30.93%	16.20%	74.82%	65.36%	67.47%	62.98%	64.98%	76.40%	53.91%	52.85%
3	DeepSeek Coder	deepseek-coder-33b-instruct	62.96%	87.58%	72.02%	44.12%	15.83%	16.67%	71.26%	57.14%	63.14%	56.81%	59.01%	77.00%	30.00%	36.09%
4	Phind	Phind-CodeLlama-34B-v2	59.00%	83.67%	55.57%	53.12%	15.09%	14.93%	58.24%	58.30%	63.60%	55.33%	59.63%	58.40%	35.26%	24.19%
5	Phind	Phind-CodeLlama-34B-v1	58.47%	81.38%	63.85%	47.05%	22.63%	5.21%	66.13%	56.94%	56.79%	49.48%	55.71%	66.00%	38.78%	35.39%
6	DeepSeek LLM	deepseek-llm-67b-chat	57.41%	82.96%	63.03%	39.09%	22.60%	5.21%	61.42%	52.73%	58.72%	55.63%	53.14%	63.00%	51.41%	36.68%
7	GPT3.5	GPT-3.5-turbo-0613	56.47%	93.08%	49.77%	31.36%	14.30%	7.64%	64.91%	48.50%	59.47%	49.64%	51.28%	70.07%	40.90%	40.13%
8	Mixtral	mixtral-8x7B-Instruct	55.55%	82.19%	56.72%	31.53%	24.00%	17.36%	54.01%	51.57%	63.69%	53.59%	56.14%	50.40%	35.58%	61.75%
9	Qwen	Qwen-72B	55.34%	81.98%	57.40%	41.61%	13.24%	4.17%	61.06%	53.16%	58.79%	44.03%	50.43%	64.00%	45.96%	36.41%
10	DeepSeek Coder	deepseek-coder-6.7b-instruct	53.25%	77.88%	56.30%	35.18%	18.89%	9.72%	65.95%	46.44%	52.46%	42.12%	48.24%	70.40%	26.90%	23.48%
11	Qwen	Qwen-72B-Chat	52.97%	82.44%	47.00%	36.09%	18.34%	9.38%	58.67%	45.81%	60.12%	44.31%	49.26%	59.00%	43.08%	33.95%
12	Magicoder	Magicoder-S-CL-7B	52.71%	77.97%	50.42%	40.20%	13.45%	12.50%	51.39%	51.98%	56.97%	50.58%	53.28%	56.67%	21.41%	26.97%
13	WizardLM	WizardCoder-Python-34B-V1.0	52.59%	78.51%	52.50%	34.25%	20.05%	10.42%	60.32%	46.39%	55.86%	44.01%	48.73%	64.00%	37.56%	24.72%
14	Phind	Phind-CodeLlama-34B-Python-v1	52.17%	80.54%	48.44%	42.58%	8.57%	1.04%	54.41%	52.34%	57.11%	41.47%	51.04%	57.80%	27.18%	39.76%
15	Magicoder	Magicoder-S-DS-6.7B	51.46%	78.93%	51.02%	28.91%	25.93%	6.48%	62.54%	46.45%	55.74%	33.84%	45.64%	69.13%	31.45%	27.86%
16	Code Llama	CodeLlama-34b-Instruct	50.45%	72.60%	55.07%	33.16%	18.43%	9.72%	51.71%	48.37%	61.36%	37.04%	48.14%	51.20%	47.76%	28.55%
17	01.AI	Yi-34B-Chat	49.58%	76.81%	47.15%	29.32%	26.39%	4.17%	44.10%	44.75%	62.29%	49.84%	53.14%	35.40%	36.15%	33.07%
18	WizardLM	WizardCoder-Python-7B-V1.0	49.10%	76.42%	48.08%	29.09%	12.50%	9.72%	58.60%	41.63%	50.67%	41.49%	46.38%	59.40%	25.30%	23.00%
19	WizardLM	WizardCoder-Python-13B-V1.0	48.99%	76.21%	46.76%	34.19%	16.17%	0.35%	52.69%	48.29%	50.67%	41.32%	48.71%	53.73%	20.45%	29.61%
20	Code Llama	CodeLlama-34b	47.36%	72.07%	43.34%	29.32%	21.20%	13.54%	53.74%	50.09%	51.52%	26.59%	43.18%	57.33%	37.37%	24.85%
21	Code Llama	CodeLlama-13b-Instruct	46.37%	69.07%	45.99%	34.37%	11.42%	7.52%	48.65%	45.18%	49.67%	39.83%	47.71%	50.47%	20.90%	12.45%
22	Zephyr	Zephyr 7B beta	46.31%	68.41%	49.99%	31.11%	14.99%	3.59%	44.26%	44.86%	54.89%	40.85%	49.28%	35.07%	27.91%	27.66%
23	DeepSeek MoE	deepseek-moe-16b-chat	45.18%	68.15%	46.72%	27.55%	10.17%	11.23%	47.19%	46.54%	45.58%	39.09%	45.71%	44.73%	25.85%	20.70%
24	OctoPack	OctoCoder	44.55%	68.19%	41.61%	29.39%	12.96%	11.11%	46.56%	37.62%	53.57%	39.56%	44.18%	47.07%	20.09%	39.20%
25	Qwen	Qwen-14B	43.69%	67.61%	47.64%	21.87%	9.63%	7.52%	44.59%	42.15%	47.09%	39.99%	41.61%	44.40%	34.19%	28.21%
26	Qwen	Qwen-14B-Chat	43.49%	68.91%	36.25%	27.73%	10.28%	15.39%	45.39%	42.12%	46.33%	38.48%	41.87%	42.73%	36.18%	34.79%
27	Magicoder	Magicoder-DS-6.7B	43.47%	67.04%	48.33%	23.11%	13.64%	0.69%	52.73%	40.42%	48.14%	25.61%	38.37%	56.73%	29.81%	38.07%
28	Code Llama	CodeLlama-34b-Python	43.13%	66.02%	40.76%	36.06%	6.94%	0.00%	50.14%	40.48%	43.64%	34.13%	40.40%	51.00%	27.63%	16.67%
29	Code Llama	CodeLlama-70b-Instruct	42.82%	59.08%	44.14%	38.48%	12.22%	7.64%	38.20%	44.99%	46.87%	42.38%	48.34%	32.00%	16.09%	5.62%
30	Magicoder	Magicoder-CL-7B	41.71%	70.38%	36.48%	23.06%	10.33%	0.35%	49.26%	35.11%	45.41%	33.47%	37.85%	52.27%	19.91%	39.21%
31	Code Llama	CodeLlama-13b	41.66%	62.77%	40.40%	31.11%	7.97%	7.41%	38.17%	44.56%	43.00%	41.72%	45.44%	34.80%	14.79%	2.47%
32	DeepSeek Coder	deepseek-coder-1.3b-instruct	41.32%	65.48%	41.42%	25.48%	6.30%	2.78%	41.91%	42.56%	42.88%	36.38%	41.80%	42.20%	16.52%	24.32%
33	Code Llama	CodeLlama-13b-Python	41.31%	62.93%	40.80%	28.61%	10.37%	5.21%	49.95%	44.60%	36.68%	27.22%	40.58%	51.07%	11.92%	13.64%
34	WizardLM	WizardCoder-15B-V1.0	41.01%	66.19%	40.34%	21.72%	12.42%	1.74%	44.80%	34.54%	47.68%	35.29%	38.43%	47.60%	22.31%	35.01%
35	Code Llama	CodeLlama-70b	40.60%	60.59%	37.42%	35.68%	7.59%	4.17%	47.18%	39.10%	39.09%	33.21%	40.54%	45.00%	19.23%	8.56%
36	Code Llama	CodeLlama-70b-Python	40.29%	59.14%	36.07%	41.06%	7.59%	0.00%	42.03%	43.04%	40.76%	32.46%	41.78%	41.00%	10.96%	19.50%
37	OctoPack	OctoGeeX	40.14%	62.54%	37.84%	26.39%	15.67%	2.20%	42.24%	33.23%	46.02%	39.10%	39.85%	39.96%	20.90%	31.11%
38	DeepSeek LLM	deepseek-llm-67b-base	39.87%	57.15%	48.73%	24.32%	9.17%	4.17%	35.50%	43.17%	46.15%	34.40%	39.98%	36.00%	30.00%	24.46%
39	Llama 2	Llama2-70B-Chat	39.30%	56.95%	38.02%	33.71%	7.96%	7.64%	35.65%	42.87%	42.56%	36.11%	40.89%	34.40%	22.44%	28.14%
40	DeepSeek Coder	deepseek-coder-33b-base	38.75%	56.73%	44.55%	19.85%	14.95%	8.33%	33.36%	43.73%	46.06%	31.23%	43.99%	25.50%	14.49%	28.02%
41	01.AI	Yi-6B-Chat	38.14%	52.73%	38.20%	34.37%	12.53%	7.64%	33.36%	39.81%	42.54%	38.33%	43.26%	23.83%	15.32%	15.69%
42	Llama 2	Llama2-70B	37.69%	51.51%	42.58%	28.48%	10.19%	10.42%	36.26%	42.99%	37.12%	32.98%	39.52%	28.00%	30.45%	0.00%
43	Code Llama	CodeLlama-7b	37.62%	59.81%	38.25%	19.37%	9.32%	4.86%	42.19%	38.60%	37.37%	28.41%	37.87%	41.80%	15.13%	0.00%
44	Mistral	Mistral-7B-Instruct-v0.1	37.55%	56.31%	41.34%	24.07%	7.47%	3.47%	39.74%	30.74%	47.10%	31.40%	34.17%	39.80%	34.44%	29.90%
45	InternLM	InternLM-Chat-20B	37.41%	59.98%	32.30%	20.40%	18.44%	7.06%	45.38%	34.67%	34.25%	31.63%	34.51%	46.20%	18.18%	23.51%
46	Qwen	Qwen-7B-Chat	37.36%	60.23%	36.20%	19.77%	7.65%	5.90%	43.44%	32.38%	38.22%	32.98%	34.06%	43.07%	29.02%	30.11%
47	DeepSeek LLM	deepseek-llm-7b-chat	36.75%	55.46%	39.38%	22.94%	6.30%	6.37%	34.08%	29.75%	46.76%	38.83%	39.15%	30.13%	15.90%	35.98%
48	Llama 2	Llama2-7B-Chat	36.14%	54.17%	35.35%	24.72%	9.44%	9.03%	35.53%	33.29%	39.16%	37.51%	37.64%	28.50%	21.35%	27.76%
49	WizardLM	WizardCoder-3B-V1.0	35.61%	57.44%	35.61%	15.23%	11.30%	6.60%	39.25%	32.08%	41.34%	26.96%	35.83%	35.40%	19.25%	26.50%
50	Code Llama	CodeLlama-7b-Instruct	35.15%	53.69%	35.79%	24.82%	7.59%	1.39%	36.46%	37.13%	35.00%	30.05%	35.97%	34.87%	15.77%	13.83%
51	InternLM	InternLM-Chat-7B	34.86%	55.80%	32.39%	20.76%	12.70%	1.85%	35.31%	34.30%	39.75%	28.52%	35.23%	34.57%	17.65%	16.86%
52	Baichuan2	Baichuan2-13B-Chat	34.40%	53.77%	27.69%	24.19%	6.85%	14.12%	37.03%	35.93%	36.39%	24.88%	34.62%	31.07%	22.63%	18.28%
53	DeepSeek Coder	deepseek-coder-6.7b-base	33.66%	53.26%	37.95%	14.02%	8.56%	2.78%	36.56%	32.40%	37.83%	25.00%	35.17%	33.47%	11.92%	8.81%
54	Code Llama	CodeLlama-7b-Python	32.89%	51.02%	28.69%	24.32%	7.59%	6.94%	30.38%	38.34%	32.37%	29.81%	35.27%	30.40%	8.97%	11.31%
55	Llama 2	Llama2-13B-Chat	32.29%	51.19%	29.18%	22.80%	7.59%	2.08%	27.51%	28.98%	42.86%	31.84%	37.07%	21.07%	9.17%	19.77%
56	WizardLM	WizardCoder-1B-V1.0	31.94%	46.90%	30.00%	27.37%	1.36%	9.72%	28.75%	30.77%	36.80%	32.94%	34.50%	25.00%	16.65%	20.69%
57	Qwen	Qwen-7B	31.69%	52.65%	32.18%	15.18%	2.10%	1.85%	33.78%	30.71%	36.78%	22.83%	31.09%	34.07%	15.71%	17.12%
58	StarCoder	StarCode+	30.67%	50.99%	29.51%	14.29%	4.01%	4.63%	31.83%	29.19%	36.04%	23.84%	33.63%	27.47%	8.72%	2.08%
59	StarCoder	StarCoder	30.66%	45.97%	30.30%	23.18%	5.93%	4.40%	24.67%	29.21%	41.06%	29.78%	36.68%	16.33%	13.27%	0.00%
60	CodeGen2.5	CodeGen2.5-7B-Instruct	29.57%	50.36%	22.07%	20.25%	6.67%	0.46%	28.76%	25.96%	37.70%	25.77%	32.35%	24.76%	11.54%	0.00%
61	InternLM	InternLM-20B	29.41%	49.21%	25.17%	18.01%	4.81%	1.74%	28.48%	24.69%	35.00%	30.79%	29.58%	26.23%	14.62%	37.60%
62	DeepSeek Coder	deepseek-coder-5.7bmqa-base	28.92%	45.62%	33.11%	11.67%	4.54%	4.51%	26.82%	27.41%	37.87%	23.17%	30.64%	24.93%	10.64%	19.54%
63	ChatGLM	ChatGLM3-6B	28.23%	42.48%	26.87%	20.78%	6.64%	6.02%	30.57%	21.80%	29.69%	31.85%	28.92%	28.23%	8.25%	27.01%
64	Baichuan2	Baichuan2-7B-Chat	27.53%	42.14%	28.83%	16.84%	3.55%	5.56%	29.02%	26.36%	32.91%	19.63%	28.30%	27.40%	3.65%	49.66%
65	Qwen	Qwen-1.8B-Chat	26.84%	40.35%	25.15%	22.50%	1.23%	5.56%	27.97%	27.23%	26.91%	24.18%	29.00%	25.27%	5.81%	19.65%
66	DeepSeek MoE	deepseek-moe-16b-base	26.65%	41.68%	31.71%	12.27%	4.21%	0.00%	28.09%	25.69%	31.15%	19.66%	27.77%	27.11%	5.38%	22.26%
67	Baichuan2	Baichuan2-13B-Base	26.32%	43.01%	21.48%	16.87%	6.47%	4.98%	22.46%	26.54%	31.79%	25.63%	30.05%	16.24%	9.23%	13.52%
68	DeepSeek LLM	deepseek-llm-7b-base	25.34%	36.58%	30.59%	15.33%	2.01%	5.56%	19.59%	27.42%	29.23%	27.22%	28.67%	15.00%	8.97%	25.29%
69	Llama 2	Llama2-13B	24.50%	38.09%	25.73%	15.00%	6.48%	0.00%	21.25%	25.09%	28.38%	24.29%	26.79%	19.80%	9.68%	4.62%
70	Baichuan2	Baichuan2-7B-Base	23.50%	36.59%	23.93%	13.01%	5.99%	4.17%	21.05%	22.93%	28.98%	21.49%	26.03%	19.33%	4.68%	10.70%
71	DeepSeek Coder	deepseek-coder-1.3b-base	23.17%	37.06%	26.74%	8.03%	2.84%	4.17%	16.05%	20.93%	34.16%	24.68%	27.02%	14.40%	4.68%	24.92%
72	Qwen	Qwen-1.8B	23.12%	37.81%	18.94%	14.04%	3.70%	6.94%	22.07%	24.68%	26.30%	18.44%	25.70%	18.40%	3.72%	24.29%
73	Mistral	Mistral-7B-v0.1	22.72%	34.86%	23.00%	15.83%	6.30%	0.00%	20.01%	25.52%	24.24%	21.32%	25.01%	17.47%	10.32%	0.00%
74	Llama 2	Llama2-7B	22.35%	37.45%	21.33%	10.00%	1.85%	4.17%	20.57%	18.80%	28.69%	22.51%	25.28%	18.27%	0.77%	12.64%
75	01.AI	Yi-34B	22.01%	34.64%	26.46%	7.73%	1.85%	4.17%	23.15%	16.96%	31.36%	15.32%	23.10%	22.40%	6.15%	11.46%
76	davinci	davinci-002	21.25%	33.66%	19.26%	13.61%	5.27%	3.70%	15.05%	19.63%	30.35%	22.70%	25.42%	13.36%	2.61%	4.33%
77	Mixtral	mixtral-8x7B	21.21%	32.76%	20.54%	15.23%	3.70%	2.08%	18.04%	14.02%	32.51%	22.78%	23.57%	13.50%	10.00%	16.03%
78	Phi	Phi1.5	20.56%	32.15%	20.61%	14.27%	3.40%	0.00%	21.04%	22.86%	21.15%	15.53%	21.83%	21.80%	1.92%	13.97%
79	01.AI	Yi-6B	19.93%	31.84%	18.91%	13.13%	0.99%	2.78%	13.75%	23.72%	23.54%	20.37%	23.42%	14.58%	0.00%	4.54%
80	CodeGeeX	CodeGeeX2-6B	19.88%	31.40%	17.41%	14.02%	2.22%	4.86%	18.78%	19.97%	25.39%	14.44%	22.08%	16.11%	4.10%	9.78%
81	CodeGen2	CodeGen2-16B	16.97%	27.46%	18.30%	8.08%	1.23%	1.39%	13.00%	17.28%	24.04%	14.23%	20.77%	7.58%	7.05%	0.00%
82	Phi	Phi2	16.74%	28.96%	13.97%	8.23%	5.12%	0.00%	17.45%	14.49%	17.17%	18.28%	18.62%	13.33%	1.73%	18.18%
83	InternLM	InternLM-7B	16.26%	25.17%	14.34%	10.86%	2.59%	6.25%	8.48%	16.95%	30.14%	10.71%	20.19%	4.89%	1.92%	24.80%
84	IEITYuan	Yuan2-51B-hf	15.25%	25.61%	12.20%	6.06%	2.78%	8.33%	20.16%	16.37%	15.38%	4.76%	15.09%	16.83%	1.92%	29.55%
85	Phi	Phi1	14.28%	20.78%	17.23%	8.08%	5.87%	0.00%	8.26%	18.93%	18.09%	12.91%	18.04%	3.33%	1.28%	26.65%
86	CodeGen	CodeGen-16B-multi	13.62%	20.79%	13.19%	10.86%	2.84%	0.00%	11.36%	17.90%	13.45%	11.43%	16.44%	6.90%	3.08%	8.77%
87	IEITYuan	Yuan2-102B-hf	10.48%	18.18%	7.77%	6.82%	1.85%	0.00%	17.12%	9.45%	6.71%	5.24%	8.41%	18.33%	0.00%	19.11%
88	IEITYuan	Yuan2-2B-hf	7.28%	9.11%	8.11%	5.56%	5.56%	2.78%	4.01%	8.29%	10.28%	7.62%	8.80%	4.27%	0.00%	6.31%