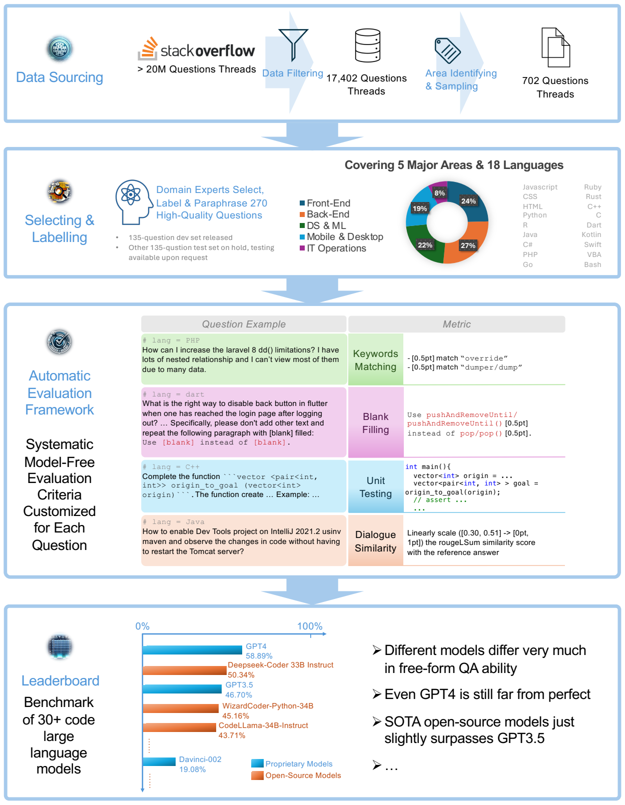
InfiCoder-Eval: Systematically Evaluating Question-Answering for Code Large Language Models

The InfiCoder Team [1]

Abstract

Large language models for code (code LLMs) have made huge progress. Evaluation benchmarks for code LLMs, such as HumanEval, DS-1000, and MBPP, predominantly focus on code generation. But they are insufficient to evaluate code LLMs' multifaceted ability. To fill this gap, we propose **InfiCoder-Eval**, a large-scale **free-form question-answering (QA) benchmark for code**. InfiCoder-Eval comprises 270 carefully picked high-quality Stack Overflow questions, covering 18 programming languages. To tackle the evaluation challenge, InfiCoder-Eval includes an evaluation framework integrating four types of model-free metrics, and domain experts design the concrete criteria for each question. We conduct a systematic evaluation with InfiCoder-Eval for more than 30 code LLMs, leading to several interesting findings. For example, though open-source code LLMs show competitive performance with proprietary models in code generation (e.g., HumanEval), they still have a large gap compared to proprietary ones in InfiCoder-Eval and even the best proprietary LLM (GPT4) is still far from perfect (best open-source model Deepseek-Coder 33B Instruct achieves 50.34% and GPT4 achieves 58.89%). Furthermore, our detailed analysis reveals several weaknesses of current code LLMs. Benchmark, evaluation tools, and detailed results are publicly available at `https://github.com/infi-coder/inficoder-eval`.


Data Sourcing
stackoverflow
> 20M Questions Threads
Data Filtering
17,402 Questions Threads
Area Identifying & Sampling
702 Questions Threads
Selecting & Labelling
Domain Experts Select, Label & Paraphrase 270 High-Quality Questions
135-question dev set released
Other 135-qustion test set on hold, testing available upon request
Covering 5 Major Areas & 18 Languages
Front-End
Back-End
DS & ML
Mobile & Desktop
IT Operations
24%
27%
22%
19%
8%
Javascript
CSS
HTML
Python
R
Java
C#
PHP
Go
Ruby
Rust
C++
C
Dart
Kotlin
Swift
VBA
Bash
Automatic Evaluation Framework
Systematic Model-Free Evaluation Criteria Customized for Each Question
Leaderboard
Benchmark of 30+ code large language models
0%
100%
GPT4 58.89%
Deepseek-Coder 33B Instruct 50.34%
GPT3.5 46.70%
WizardCoder-Python-34B 45.16%
CodeLLama-34B-Instruct 43.71%
Davinci-002 19.08%
Proprietary Models
Open-Source Models

Question Example	Metric	
`# lang = PHP` How can I increase the laravel 8 dd() limitations? I have lots of nested relationship and I can't view most of them due to many data.	Keywords Matching	- [0.5pt] match "`override`" - [0.5pt] match "`dumper/dump`"
`# lang = dart` What is the right way to disable back button in flutter when one has reached the login page after logging out? ... Specifically, please don't add other text and repeat the following paragraph with [blank] filled: `Use [blank] instead of [blank].`	Blank Filling	`Use pushAndRemoveUntil/ pushAndRemoveUntil()` [0.5pt] `instead of pop/pop()` [0.5pt].
`# lang = C++` Complete the function ` ```vector <pair<int, int>> origin_to_goal (vector<int> origin)``` `. The function create ... Example: ...	Unit Testing	`int main(){ vector<int> origin = ... vector<pair<int, int> > goal = origin_to_goal(origin); // assert ... ...`
`# lang = Java` How to enable Dev Tools project on IntelliJ 2021.2 usinv maven and observe the changes in code without having to restart the Tomcat server?	Dialogue Similarity	Linearly scale ([0.30, 0.51] -> [0pt, 1pt]) the rougeLSum similarity score with the reference answer

- ➢ Different models differ very much in free-form QA ability
- ➢ Even GPT4 is still far from perfect
- ➢ SOTA open-source models just slightly surpasses GPT3.5
- ➢ ...

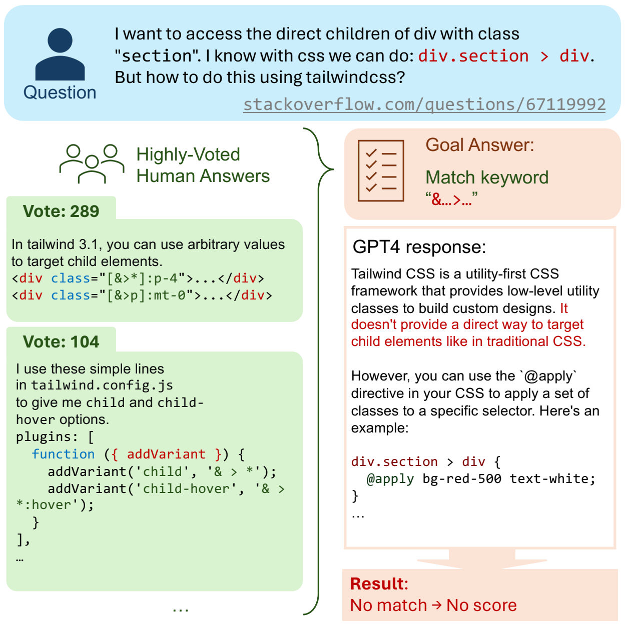
Figure 1: InfiCoder-Eval benchmark overview.

1. Introduction

In recent years, large language models (LLMs) are revolutionizing the software development regime, where LLMs are showing exceedingly strong and generic capabilities in comprehending, generating, debugging, and summarizing code (Chen et al., 2021; Li et al., 2022). Products built upon code LLMs such as GitHub Copilot (Github, 2023) hit millions active users within just one year. Along with the huge success of proprietary LLMs such as GPT3.5 and GPT4 (OpenAI, 2023a), open-source code LLMs[1] (Nijkamp et al., 2023b; Touvron et al., 2023b; Roziere et al., 2023; Luo et al., 2023) are developing at an unprecedented fast pace. As of November 2023, there are over 2,000 open LLMs submitted to the Hugging Face Open LLM Leaderboard (Beeching et al., 2023).

[1]ByteDance Ltd. and Peking University. Correspondence to: Linyi Li <linyi.li@bytedance.com>, Hongxia Yang <hx.yang@bytedance.com>.

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



I want to access the direct children of div with class "section". I know with css we can do: `div.section > div`. But how to do this using tailwindcss?
stackoverflow.com/questions/67119992

Vote: 289

In tailwind 3.1, you can use arbitrary values to target child elements.

```
<div class="[&>*]:p-4">...</div>
<div class="[&>p]:mt-0">...</div>
```

Vote: 104

I use these simple lines in `tailwind.config.js` to give me `child` and `child-hover` options.

```
plugins: [
  function ({ addVariant }) {
    addVariant('child', '& > *');
    addVariant('child-hover', '& >
*:hover');
  }
],
…
```

GPT4 response:

Tailwind CSS is a utility-first CSS framework that provides low-level utility classes to build custom designs. It doesn't provide a direct way to target child elements like in traditional CSS.

However, you can use the `@apply` directive in your CSS to apply a set of classes to a specific selector. Here's an example:

```
div.section > div {
  @apply bg-red-500 text-white;
}
...
```

Result:
No match → No score

Figure 2: A challenging question paraphrased from Stack Overflow (post #67119992) where GPT4 fails to answer.

Given tons of code LLMs available, the development of reliability code benchmarks seems to lag. Especially, the benchmarks for code LLMs typically focus on a specific task or domain, especially on code generation or multi-choice questions. For example, HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) focus on Python code generation for simple algorithms and standard library, DS-1000 (Lai et al., 2023) focuses on code generation in data science. Though recent efforts extend code generation benchmarks to other scenarios and languages (Muennighoff et al., 2023), the extension evolves from single-domain source data, e.g., HumanEval Python programs, and still lacks diversity. As a result, these benchmarks are relatively simple and strong code LLMs such as GPT-4 tend to saturate when evaluated on these benchmarks, e.g., GPT-4 can already achieve 86.6% pass@1 score on HumanEval (Muennighoff et al., 2023).

In contrast, users interact with code LLMs in diverse scenarios, including but not limited to code generation, debugging, usage questioning, and opinion consulting. In contrast to strong performance on existing benchmarks, current code LLMs may be still unhelpful in some real-world usage scenarios, as exemplified in Figure 2. *Can we systematically and comprehensively evaluate code LLMs' ability in challenging real-world usage scenarios?*

We introduce InfiCoder-Eval, a systematic benchmark for evaluating the free-form question-answering ability of code LLMs, that fulfills this urgent demand. The core principle of InfiCoder-Eval is to maximize its representativeness of real-world developers' usage. To achieve so, InfiCoder-Eval comprises 270 questions sampled and filtered proportionally from the natural high-quality question distribution of Stack Overflow, without any constraint on topic, language, question, or answer form. As a result, these 270 questions cover 18 programming languages spanning 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 evaluation complexity. Unlike code generation benchmarks or multiple-choice benchmarks, it is challenging to unify the evaluation criteria into a single form like unit testing or choice checking. On the other hand, relying on model-based evaluation such as GPT4 evaluation requires much cost and incurs concerns about privacy and bias. InfiCoder-Eval includes an evaluation framework integrating four types of model-free metrics (keyword matching, blank filling, unit testing, and dialogue similarity) to mitigate the evaluation challenge. For each question, we invite domain experts to paraphrase the prompt, choose the metric, and write down the concrete criteria with domain-specific knowledge taking highly-voted answers from Stack Overflow as the reference. As a result, the whole benchmark can be directly evaluated in a sandbox environment given any model responses.

We conduct an extensive evaluation for both proprietary models and open-source models with InfiCoder-Eval framework, totaling over 30 models with diverse model sizes (1B — >30B). In contrast to the competitive performance of open-source models in existing benchmarks, on InfiCoder-Eval, GPT4 achieves 58.89% score which still far exceeds the most competitive open-source code LLM (up to November 2023), Deepseek-Coder 33B Instruct (DeepSeekAI, 2023), which achieves 50.34% score. On the other hand, the GPT4 score suggests that even the most advanced LLM is far from perfect and this unsaturated benchmark may be a better fit for code LLMs. Furthermore, a large group of existing code LLMs, such as WizardCoder (Luo et al., 2023), Code LLama (Roziere et al., 2023), and Zephyr-7b-β (Tunstall et al., 2023), approaches GPT3.5 but has not surpassed it yet. The area, language, and metric type labeling of InfiCoder-Eval questions further allow us to conduct a deep analysis of the performance of different code LLMs, which reveals how GPT4 falls behind human experts and how open-source models fall behind GPT4.

The evaluation tools and detailed results are publicly available. We uniformly sample 135 benchmark questions as the dev set and release it to the public. The remaining 135 questions are held as the test set where the whole set evaluation is available upon request. All resources are available at `https://github.com/infi-coder/inficoder-eval` and will be actively maintained and extended as part of the InfiCoder project.

2. Benchmark Creation

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

2.1. Data Curation

We inspect the full Stack Overflow dataset. The dataset contains 23.54 million question posts and 34.68 million answer posts, all transformed to Markdown format. Each question post has the 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. Over 1 million questions is still a large source to sample from. Then, 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 by their days between creation and June 14, 2023 (data collection end-date) as the x-axis and the logarithm of the view count as the y-axis. As shown in Figure 3, we empirically determine to keep questions that lie above the line connecting $(0, 5)$ and $(3000, 15.5)$, which results in 17,402 questions. For these questions, relying on the mandatory question tags, we manually construct a tag tree that covers the 200 most frequent tags and thus are able to determine the language and area for 14,330 questions. These questions are from 24 programming languages, and each language is assigned a primary area among five (front-end, back-end, data science and machine learning (DS&ML), mobile and desktop, and information technology (IT) operations). Lastly, we filter out 6 programming languages that describe data or are domain-specific: JSON, regex, Markdown, YAML, CSV, and SQL. As a result, we get 13,854 questions that are used as the *initial seed set*.

2.2. Sampling and Calibration

Based on the internal developers' demand, we assign 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. Then, based on our domain expert availability, 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 break these area quotas down to language quotas proportionally according to the language frequency. However, we observe that according to this rule, some languages, such as CSS, C/C++, and VBA, are too few (less than 10) to provide a reliable language sub-score, so we determine a lower bound of 10 questions for these languages. As a result, we get the final question quota at the area and language level as shown in Table 1, which sums up to 270 questions. After determining the final question quota, we also determine the inspecting question quota, so that we can randomly sample questions from the initial seed set for domain experts to inspect, select, and label the final benchmark questions.

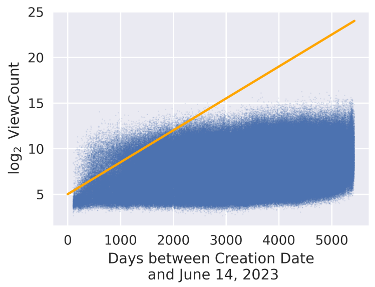


Figure 3: Scatter plot of filtered Stack Overflow questions. We keep the questions above the orange line.

Table 1: 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 from the inspecting set according to the language quota and annotate the final benchmark.

Area	Language	Initial Seed Set	Inspecting Set	Final InfiCoder-Eval Benchmark			
		# Questions	# Questions Quota	# Questions Quota	% Questions Quota	# Area Quota	% Area Quota
Front-End	Javascript	4912	132	44	16.30%	64	23.70%
	CSS	87	20	10	3.70%		
	HTML	600	20	10	3.70%		
Back-End	Java	930	54	18	6.67%	80	29.63%
	C#	629	36	12	4.44%		
	PHP	462	27	10	3.70%		
	Go	117	20	10	3.70%		
	Ruby	71	20	10	3.70%		
	Rust	96	20	10	3.70%		
	C/C++	287	20	10	3.70%		
DS & ML	Python	2779	141	47	17.41%	57	21.11%
	R	184	20	10	3.70%		
Mobile & Desktop	Dart	1562	56	19	7.04%	48	17.78%
	Kotlin	383	20	10	3.70%		
	Swift	551	20	10	3.70%		
	VBA	16	16	9	3.33%		
IT Ops.	Bash	188	60	21	7.78%	21	7.78%
Total		13854	702	270			

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,

Table 2: InfiCoder-Eval statistics.

(a) Question type.

Question Type	Ratio
Code Completion	30.37%
Knowledge Question-Answering	27.04%
Code Debugging	26.67%
Non-Code 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

non-code debugging (resolving the problem does not need to write or revise code), 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.

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

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 2.

Score Computation. We treat each question equally with one point each. Given 270 questions in the benchmark, the full score is 270, and we by default report the percentage score (achieved score divided by 270) unless otherwise noted.[2] 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

[2]In the future, we plan to partition the benchmark into simple, medium, hard, and unsolved questions.

can contribute to 0.4, 0.2, 0.2, and 0.2 points respectively to the final score.

Framework Implementation. We have implemented an automated evaluation framework supporting all 270 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, 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 3, 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 GPT4 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.

2.6. Data and Tool Availability

The InfiCoder-Eval evaluation framework is publicly available. The InfiCoder-Eval benchmark questions are uniformly and evenly split into two sets: dev set and test set. The dev set is released to the public. The test set is on-held to prevent potential memorization and overfitting. The whole set evaluation is available upon request. Resources are at `https://github.com/infi-coder/inficoder-eval` and will be actively maintained and extended as part of the InfiCoder project.

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 over 30 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 leave the exploration of other hyperparameters as the future work.

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 GPT4, GPT3.5(-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 GPT4 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 GPT4 context (prompt + GPT4 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 GPT4 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.

3.2. Results and Analysis

We plot the evaluation results of over 30 LLMs in Figure 4, where each open source model is represented by a dot (with error bars for models whose parameter size is smaller than 30B), and each proprietary model is represented by a line with error bar (since their sizes are not known). The error bars are computed by repeating the evaluation three times.

Table 3: 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	GPT4 Score
HumanEval (Chen et al., 2021)	Python Programming	164	Test Cases	Hand-Written	98.1%
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., 2023)	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	270	Keyword and Blank + Test Cases + Text Similarity	StackOverflow	58.89%

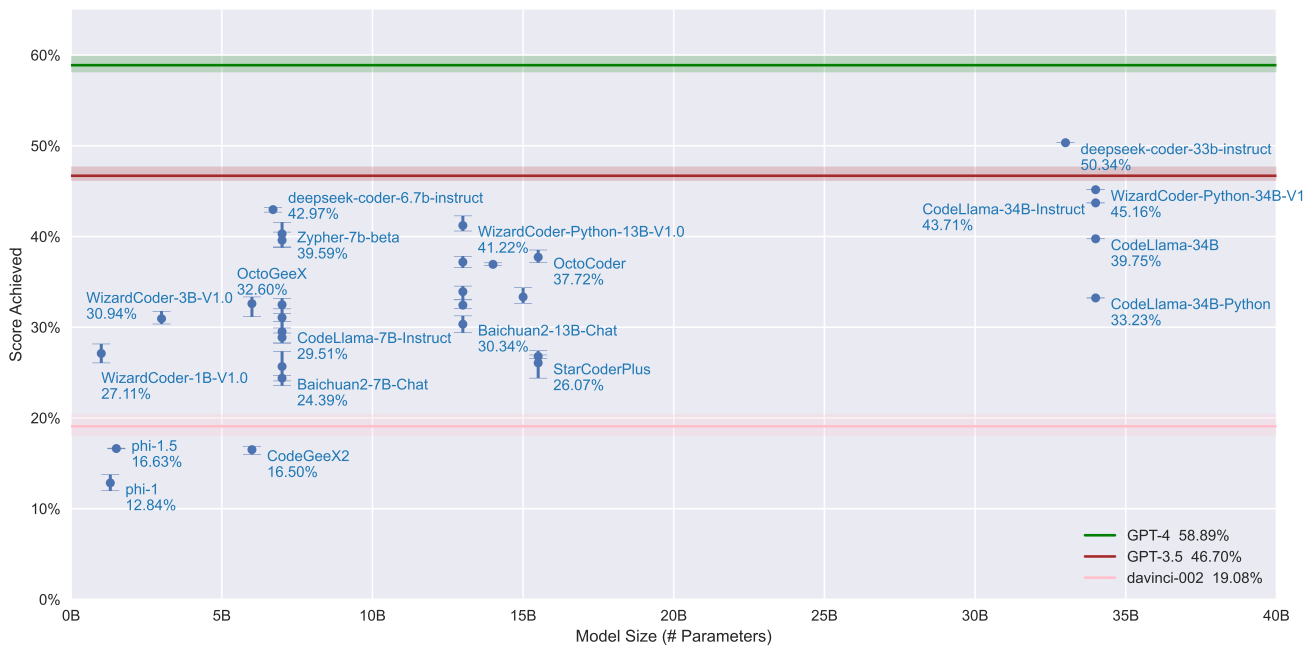


Figure 4: Scatter plot for all evaluated LLMs on InfiCoder-Eval with open-source models shown as points and proprietary models shown as lines. See detail discussion in Section 3.2.

Names and scores are shown for representative models. The best-performing model within each model family is shown in Table 5. We summarize the findings below.

Best open-source models are competitive but still far from GPT4. As expected, GPT4 achieves the highest score 58.89%. The runner score is achieved by an open-source model, deepseek-coder-33b-instruct (DeepSeekAI, 2023), with a 50.34% score. On the one hand, the 8.55% gap between GPT4 and this open-source model is still visibly large. On the other hand, GPT3.5 achieves a 46.70% score, which has been surpassed by 3.64%. In summary, evaluated with InfiCoder-Eval, the state-of-the-art open-source model has its ability between GPT3.5 and GPT4, more on the GPT3.5 end. Interestingly, this result coincides with the pass@1 rate on HumanEval, though the open-source model is slightly stronger there.

GPT4 is still far from perfect. Noting that the full score of InfiCoder-Eval is 100%, even the powerful GPT4 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, GPT4 achieves similar scores 64.32% and 62.42% respectively. For blank filling, the score is relatively lower at 45.25%. These scores may imply that GPT4 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.

Among open-source models, different models have very different performances. Figure 4 systematically visualizes the performance of different open-source models at diverse scales. Imagining a fitting line for all points, one can observe that scaling still matters, as models smaller than 5B have scores between 10% and 30%, models between 5B

Table 4: 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	$31.07\%_{\pm 0.87\%}$	$28.88\%_{\pm 0.45\%}$ (−2.19%)	$29.51\%_{\pm 0.97\%}$ (−1.56%)
13B	HumanEval	36.0%	43.3% (+7.3%)	42.7% (+6.7%)
	InfiCoder-Eval	$33.92\%_{\pm 0.64\%}$	$32.43\%_{\pm 0.42\%}$ (−1.49%)	$37.18\%_{\pm 0.51\%}$ (+3.26%)
34B	HumanEval	48.8%	53.7% (+4.9%)	41.5% (−7.3%)
	InfiCoder-Eval	39.75%	33.23% (−6.52%)	43.71% (+3.96%)

and 20B have scores between 15% and 45%, and models larger than 30B have scores between 30% and 50%. This phenomenon has an important implication: scaling is important, but not all: We can see that when scaling up, the lower bound of model performance steadily increases, as models >30B all have scores >30%. On the other hand, advanced data and training techniques can help to significantly boost performance, up to 10% to 20%. As a result, some small-scale models, such as deepseek-coder-6.7b-instruct, can even outperform large models with >30B parameters by around 10%.

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-coder-33b-instruct, CodeLlama-34B-Instruct, Zypher-7b-β, and OctoCoder. In contrast, the pretraining models, such as davinci-002 and phi models, usually perform poorly despite 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 5, 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 4, 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.

Table 5: Results of best-performing LLM with each LLM family along with their HumanEval scores and number of parameters (if known). Full leaderboard available at `https://github.com/infi-coder/inficoder-eval`.

Model	# Params. (in B)	InfiCoder-Eval Score	HumanEval pass@1
GPT-4 (OpenAI, 2023a)	/	$58.89\%_{\pm 0.74\%}$	88.4%
deepseek-coder-33b-instruct (DeepSeekAI, 2023)	33	50.34%	79.3%
GPT-3.5 (OpenAI, 2023b)	/	$46.70\%_{\pm 0.70\%}$	73.2%
WizardCoder-Python-34B (Luo et al., 2023)	34	45.16%	73.2%
CodeLlama-34B-Instruct (Roziere et al., 2023)	34	43.71%	41.5%
Zypher-7b-β (Tunstall et al., 2023)	7	$39.59\%_{\pm 0.68\%}$	32.38%
OctoCoder (Muennighoff et al., 2023)	15.5	$37.72\%_{\pm 0.58\%}$	46.2%
Qwen-14B-Chat (Bai et al., 2023)	14	$36.93\%_{\pm 0.12\%}$	43.9%
Baichuan2-13B-Chat (Yang et al., 2023)	13	$30.34\%_{\pm 0.76\%}$	17.07%[3]
StarCoder (Li et al., 2023a)	15.5	$26.79\%_{\pm 0.18\%}$	33.6%
CodeGen2.5-7B-instruct (Nijkamp et al., 2023a)	7	$25.67\%_{\pm 1.57\%}$	36.20%
davinci-002 (OpenAI, 2023b)	/	$19.08\%_{\pm 1.00\%}$	47.0%[4]
phi-1.5 (Li et al., 2023b)	1.5	$16.63\%_{\pm 0.03\%}$	34.1%
CodeGeeX2 (Zheng et al., 2023)	6	$16.50\%_{\pm 0.39\%}$	35.9%

4. 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, 2023a) and open-source ones (Beeching et al., 2023; Nijkamp et al., 2023b; Touvron et al., 2023a;b; Li et al., 2023a; 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., 2023), and task gen-

[3]This is the score of the base model. Chat model number not reported yet.

[4]This is the score of code-davinci-002. Score of davinci-002 base model not reported yet.

eralization (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.

5. Conclusion

We proposed InfiCoder-Eval, a systematic benchmark for evaluating the question-answering ability of code large language models. InfiCoder-Eval comprises 270 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 30 code LLMs reveals several interesting findings and takeaways. The benchmark and evaluation framework is open source as part of the InfiCoder project, and we will be continuously maintaining and expanding the benchmark.

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.

Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

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.

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.

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*, 2023a.

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.

Li, Y., Bubeck, S., Eldan, R., Del Giorno, A., Gunasekar, S., and Lee, Y. T. Textbooks are all you need ii: phi-1.5 technical report. *arXiv preprint arXiv:2309.05463*, 2023b.

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. *arXiv preprint arXiv:2305.01210*, 2023.

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., Hayashi, H., Xiong, C., Savarese, S., and Zhou, Y. Codegen2: Lessons for training llms on programming and natural languages. *arXiv preprint arXiv:2305.02309*, 2023a.

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*, 2023b. URL https://openreview.net/forum?id=iaYcJKpY2B_.

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

OpenAI. Models - openai api. https://platform.openai.com/docs/models/gpt-3-5, 2023b.

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.

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.

Tunstall, L., Beeching, E., Lambert, N., Rajani, N., Rasul, K., Belkada, Y., Huang, S., von Werra, L., Fourrier, C., Habib, N., et al. Zephyr: Direct distillation of lm alignment. *arXiv preprint arXiv:2310.16944*, 2023.

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.

Yang, A., Xiao, B., Wang, B., Zhang, B., Yin, C., Lv, C., Pan, D., Wang, D., Yan, D., Yang, F., et al. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*, 2023.

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.