Environment Free Coding Benchmarks: Evaluating Language Model Coding Capabilities without a Dedicated Environment

Laurence Liang
McGill University
Rootly AI Labs
laurence.liang@mail.mcgill.ca

Abstract

The increasing adoption of language models for coding tasks has prompted researchers to develop coding benchmarks to better assess and quantify a language model's coding abilities on a variety of tasks. Existing benchmarks effectively evaluate model code generation and understanding abilities, but typically require an external environment to verify code, which can slow down and complicate model evaluation. As a result, this paper presents the Environment-Free Coding Bencharks (EFCB) suite - a collection of 5,512 questions from real-world GitHub pull requests -that introduces multiple advantages relative to existing coding benchmarks: eliminating the need to use an external coding environment, a larger and more diverse question bank spanning different programming languages and industry use cases, and a multi-faceted collection of tasks that evaluate different indicators with respect to model coding abilities. By evaluating EFCB with o4-mini and Llama-3.3-70B as state of the art (SOTA) models, we observe that current SOTA models achieve approximately uniform performance across different programming languages and use cases, and we identify areas of improvement for existing SOTA models given that current EFCB results have not yet attained benchmark saturation.

1 Introduction

Since the introduction of the transformer model, attention-based language models have shown impressive performances on a variety of text-based tasks. (Vaswani et al., 2023) In particular, recent developments have also shown that including code in pre-training and post-training model data, as well as increasing model size, can lead to emergent properties. (Wei et al., 2022) Notably, one of these observed emergent properties is a transformer model's ability to understand and write code. Accordingly, models such as OpenAI's reasoning models (o1/o3/o4 family), Qwen-2.x and Claude 4 Sonnet/Opus all perform strongly as the current state of the art models for coding capabilities.

Given the increase in the performance of transformer models on coding-related tasks, it becomes relevant to rigorously quantify their strengths and limitations to better define their capabilities to complete real world coding tasks. Commonly used benchmarks such as HumanEval and SWE-Bench have become an industry standard in benchmarking models for their coding abilities, as their tasks are representative of real-world use cases. (Chen et al., 2021; Jimenez et al., 2024) However, these benchmarks present multiple limitations: (i) they generally require an external environment to run and test code and (ii) only evaluate a single aspect of a model's coding abilities (i.e. code generation) without evaluating other equally important aspects (i.e. model hallucination). While dedicated environments for coding benchmarks are often desirable, coding tasks that require highly specific environment versions or depend on outputs that cannot be evaluated in a single test/fail manner (i.e. user interface-dependent tasks) make it challenging to adapt refined environments for every new test case.

As a result, this paper introduces a new suite of coding benchmarks called the Environment-Free Coding Benchmarks, which are based on 1,800 examples of real-world pull requests from popular repositories on GitHub. In total, this evaluation suite comprises 5,512 unique questions across 5 different tasks, which we evaluate with Llama-3.3-70B and OpenAI's o4-mini model as two commonly used models with state of the art capabilities. As a result, this paper makes the following contributions:

- **We introduce EFCB as a new coding benchmark**. EFCB offers 5,512 unique questions across 5 different tasks related to code under-

Benchmark	Number of Questions	Multiple Languages ?
HumanEval (Chen et al., 2021)	164	No
SWE-Bench (Jimenez et al., 2024)	2,294	No
CRUXEval (Gu et al., 2024)	800	No
LiveCodeBench (Jain et al., 2024)	511	No
LiveBench (White et al., 2025)	128	No
EFCB (ours)	5,512	Yes

Table 1: Comparision between EFCB and other commonly used coding benchmarks

Organization	Repository	Type	Number of Questions	Primary Language
mastodon	mastodon	social media platform	163 to 239	Ruby
bluesky-social	indigo	social media platform	141 to 176	Go
duckdb	duckdb	database system	174 to 215	C++
chroma-core	chroma	database system	160 to 196	Rust
cloudflare	cloudflared	tunnel	82 to 148	Go
tailscale	tailscale	tunnel	148 to 187	Go

Table 2: Description of the 6 repositories used for EFCB.

standing, code generation and hallucination mitigation. These questions and code samples are sourced from pull requests from 6 GitHub repositories with high contributor activity and active industry use cases.

- **We evaluate o4-mini and Llama-3.3-70B**. Our results show that model performance is generally uniform across different programming languages and use cases. Furthermore, we show that even the current state of the art models do not attain benchmark saturation, and we highlight areas to further develop languge model coding abilities.

2 Method

2.1 Data Collection

To populate the questions in the EFCB suite, we use pull requests and code patches from six GitHub repositories, which we curate using the GitHub API through a Python client. We only keep code patches (including lines with file names) that have at least 15 lines but do not exceed 7,500 characters. This filter ensures that code patches are non-trivial to solve, while preventing overly long code patches that may exceed the input context length of select language models (since some questions such as GMCQ-Hard can contain a very similar code patch four times, thus quadrupling the question length).

2.2 Model Evaluation

2.2.1 Choice of Models

This paper evaluates an open source model and a closed source model, which correspond to two types of language models that are actively in use. We select OpenAI's o4-mini as our choice of closed source model, and Meta's Llama-3.3-70B model as an open source model. We chose both models given their reported state-of-the-art coding abilities and their widespread use in industry. We use OpenAI's public API to access o4-mini, and Together AI's public API to access its Llama 3.3 70B implementation: llama-3.3-70b-turbo.

2.2.2 Choice of Code Patches

To ensure that the code samples do not exceed the a language model's context length, we discard any pull request whose code patch contains more than 7,500 characters. Given that some models have a maximum context window of around 32,000 tokens (such as some variants of the Llama-3.3 model), this upper bound for each code patch's character count ensures that each code patch can fit within smaller context windows. Most language models as of this writing support context lengths of around 128k tokens, though we still impose this upper limit to ensure that as many language models as possible can run the EFCB tasks. Furthermore, given that two of the EFCB tasks (GMCQ-Easy and GMCQ-Hard) use four code samples, this this upper bound on the character count further enforces the ability

to integrate multiple choice questions with respect to a 32,000 token context window.

All tasks have a lower bound where each code sample must not be empty. In particular, the MPR-Gen task requires that each code patch contain at least 15 lines of code, to ensure that the mask (which covers 5 lines of code) still leaves sufficient inline context.

2.2.3 Task-Specific Performance

Each task has its own scoring metric. GMCQ-Easy and GMCQ-Hard use accuracy (from 0% to 100%) based on a multiple choice question format (with a total of four questions). MPR-Gen and Reverse-QA use an LLM-as-a-judge (gpt-4.1-nano) to score the target model from 0 to 10. Reverse-QA-Hallu measures the hallucination rate (from 0% to 100%) also using an LLM-as-ajudge (gpt-4.1-nano). Our choice to use an LLM-as-a-judge is based on earlier work that shows that certain language models provide sufficiently reliable results to be used to evalute model outputs. (Zheng et al., 2023)

2.2.4 Overall Performance

We use the following metric to evaluate the overall model performance on the EFCB suite:

$$\text{EFCB score} = \frac{1}{5}(\sum_{i=1}^{2} A_i + \sum_{i=1}^{2} \frac{B_i}{10} + (1 - C))$$

where A_i represents the two GMCQ tasks that use accuracy, B_i represents the two tasks that are scored on a maximum of 10 (Reverse-Qa and MPR-Gen), and C represents the hallucination rate.

3 Dataset

3.1 Choice of GitHub Repositories

We select six GitHub repositories that provide the pull request and code patches for our EFCB dataset. When selecting our choice of GitHub repositories, we made sure that these GitHub repositories of interest (i) had a high number of well-documented pull requests and coding contributions, (ii) were libraries that were as close as possible to a production-level tool and (iii) had a strong adoption rate in industry (using GitHub stars as an indicator).

As shown in Table 2, we choose six GitHub repositories spanning different use cases and programming languages: two of them related to social media infrastructure (mastodon's full stack codebase and Bluesky's Indigo service for its messaging protocol), two of them related to database systems (duckdb and chroma), and the remaining two related to tunneling services (tailscale and Cloudflare's cloudflared client).

3.2 EFCB Tasks

The EFCB suite has five tasks, which are defined as follows. For sake of brevity, the full question prompts are included in the appendix.

3.2.1 GMCQ-Easy

GitHub Multiple Choice Question-Easy (GMCQ-Easy) provides the model with a GitHub pull request title and description, and then lists four choices of code patches corresponding to actual code patches for different pull requests in the same repository. Next, the model must be able to choose the choice of the code patch that susccessfully closed the pull request.

3.2.2 GMCQ-Hard

GitHub Multiple Choice Question-Hard (GMCQ-Hard) is a "needle-in-the-haystack" variation of GMCQ-Easy. Given a GitHub pull request title and description, the model is then provided with 4 variations of the code path that closed that pull request. Three of these options have two "words" (i.e. variables) that are swapped, meaning that the option with no word swapping is the correct one.

3.2.3 MPR-Gen

For **Masked Pull-Request Generation** (MPR-Gen), the model is provided with the GitHub pull request's title, description and code path with a masked section. The model is then asked to generate the code corresponding to the masked section, which gpt-4.1-nano rates from 0 to 10 as an LLM-as-a-judge.

3.2.4 Reverse-QA

Reverse-Question & Answer (Reverse-QA) provides a model with a code patch from a pull request, and then asks the model to generate a title and description that corresponds to the code patch. A gpt-4.1-nano model then rates the similarity from a scale of 0 to 10 as an LLM-as-a-judge.

3.2.5 Reverse-QA-Hallu

Reverse-QA-Hallucination (Reverse-QA-Hallu) is a variation of Reverse-QA, where the LLM-as-

Task	Chroma	Cloudflare	DuckDB	Indigo	Mastodon	Tailscale	Total
GMCQ-Easy	196	148	215	176	239	187	1,161
GMCQ-Hard	196	148	215	176	239	187	1,161
MPR-Gen	160	82	174	141	163	148	868
Reverse-QA	196	148	215	176	239	187	1,161
Reverse-QA-Hallu	196	148	215	176	239	187	1,161
Total	-	-	-	-	-	-	5,512

Table 3: Question count for all five tasks in the EFCB.

Model	GMCQ-E	GMCQ-H	MPR-G	R-QA	R-QA-H	EFCB Score
Llama-3.3-70B	0.803	0.476	3.74	0.965	7.23	0.482
o3-mini	0.892	0.868	3.67	0.946	7.41	0.584

Table 4: Summary of EFCB Suite Results

a-judge is prompted to detect hallucinations as opposed to similarity. If the model's response contains any information that was not present in the original GitHub issue, the LLM-as-a-judge considers that to be an example of a hallucination.

4 Results

Overall, we note observe that o4-mini has a slight edge in terms of performance over Llama-3.3-70B, though there are cases where Llama-3.3-70B scores better. Furthermore, model performance is generally uniform across codebases and programming languages, which suggests that current state of the art models show sufficient generalization across different coding languages and use cases.

4.1 Code Understanding

GMCQ-Easy, GMCQ-Hard and Reverse-QA serve as tasks that evaluate a model's ability to understand code. Given the results tables, we note that both OpenAI's o4-mini and Llama-3.3-70B show similar performance across tasks, which suggests that current state of the art language models can have sufficiently uniform performance across different codebase types and programming languages without any noticeable biases. However, we note that model performance consistently dropped for Bluesky's Indigo repository, which suggests that either the documentation or coding style for Indigo may have certain characteristics which makes it more challenging for language models to complete. While model scores are generally high across these three tasks, we believe that we have not yet attained performance saturation yet: both GMCQ tasks have an evident ground truth, which would make attaining 100% accuracy an achievable outcome. Additionally, MPR-Gen scores average around 3.7 for Revesre-QA, with ample room to improve model performance.

4.2 Code Generation

Using MPR-Gen to evaluate code generation abilities, we observe that both o4-mini and Llama-3.3-70B perform strongly and almost uniformly across different repositories, averaging between 7.2 and 7.4 on a scale of 10. This further supports the idea (from the code understanding subsection) that current state of the art models perform uniformly across different coding languages and use cases. While the average scores between 7.2 and 7.4 are close to 10, there is still space for improvement, which suggests that this task is not saturated yet.

4.3 Hallucination Mitigation

For Reverse-QA-Hallu, we observe that hallucination rates are still very elevated and uniform across different repositories, averaging between 94% to 97%. One likely explanation is that language models tend to be verbose: given a code patch, language models may generate adjacent concepts and ideas that are not initially present in a human-authored pull request title and description. Accordingly, these results suggest that additional work should be directed to better define the scope of what constitutes a hallucination and to identify strategies that can further mitigate hallucinations for models working on code-related tasks.

Model Name	mastodon	indigo	cloudflared	duckdb	tailscale	chroma	unweighted average
llama-3.3-70b	7.48	6.9	6.88	7.18	7.5	7.45	7.23
o4-mini	7.5	7.35	7.49	7.29	7.73	7.11	7.41

Table 5: Results for MPR-Gen

Model Name	mastodon	indigo	cloudflared	duckdb	tailscale	chroma	unweighted average
llama-3.3-70b	0.912	0.699	0.845	0.8	0.759	0.801	0.803
o4-mini	0.975	0.869	0.872	0.893	0.824	0.918	0.892

Table 6: Results for GMCQ-Easy

5 Related Work

5.1 Language Model Benchmarks

Before the rise of transformer models, early work in language model evaluation used language translation (i.e. English to French), among other similar benchmarks, to evaluate model performance. (Sutskever et al., 2014; Bahdanau et al., 2016) The introduction of the transformer model coincided with an increase in industry standard benchmarks. A common type of benchmark used multiple choice questions as a standard format, with MMLU, WNLI and GPQA being notable examples of multiple choice benchmarks. (Hendrycks et al., 2021; Levesque et al., 2011; Rein et al., 2023) Recent work has also shown that certain language models are sufficiently reliable to be used to directly evaluate language model outputs: consequently, this configuration, known as "llm-as-a-judge", allows benchmarks to resemble human-like conversations without requiring a human judge in the loop, thereby making language model evaluations much more efficient and diverse. (Zheng et al., 2023)

5.2 Coding Benchmarks

With the rise of transformer models, one of the earliest coding-specific benchmarks which is still industry-standard is the HumanEval benchmark that asks a model to generate a Python function body, when given some stating code and a docstring with context and instructions. (Chen et al., 2021) Building on the idea of code generation as a task, SWE-bench introduces industry-like tasks by asking language models to create pull requests with generated code that can close real-world GitHub code issues. (Jimenez et al., 2024) Other recent developments include LiveCodeBench and LiveBench that focus on contamination-free coding tasks (tasks that are not part of a model's pre-training data) (Jain et al., 2024; White et al., 2025), and CRUXEval which aims to evaluate code understanding by prompting a model to generate a function's input based on the output, and to do the complementary task as well. (Gu et al., 2024).

5.3 Hallucination Benchmarks

Research in developing hallucination benchmarks is very recent, as hallucinations have only been extensively studied with the rise of transformer models. One notable hallucination evaluation dataset is HaluEval, which asks a language model to determine whether a generated response contains a model hallucination. (Li et al., 2023). Another approach involves training language models that specialize in hallucination detection, which can be used similarly as an llm-as-a-judge: one such example is the LYNX model, which is capable of detecting hallucinations better than state of the art models like GPT-4o and Claude-3-Sonnet. (Ravi et al., 2024).

6 Conclusion

This paper introduces the Environment-Free Coding Benchmarks suite that comprises 5,512 unique coding questions that are sourced from real-world pull requests originating from six GitHub repositories with an active contributor base. By comparing EFCB with previous coding benchmarks, we show that EFCB offers multiple advantages: (i) it does not require an external environment to run code examples, thereby simplifying the benchmark evaluation process; (ii) it evaluates multiple abilities related to coding tasks (code understanding, generation, and hallucination), as opposed to most coding benchmarks that evaluate a single aspect; and (iii) EFCB offers a much more diverse and larger question bank across multiple programming languages and different production-ready code use cases.

After evaluating o4-mini and Llama-3.3-70B

Model Name	mastodon	indigo	cloudflared	duckdb	tailscale	chroma	unweighted average
llama-3.3-70b	0.452	0.392	0.574	0.47	0.465	0.5	0.476
o4-mini	0.883	0.824	0.919	0.879	0.834	0.867	0.868

Table 7: Results for GMCQ-Hard

Model Name	mastodon	indigo	cloudflared	duckdb	tailscale	chroma	unweighted average
llama-3.3-70b	3.94	3.4	4.01	3.97	3.84	3.28	3.74
o4-mini	4.26	3.36	4.07	3.4	3.74	3.18	3.67

Table 8: Results for Reverse-QA

as our choice of state of the art models, we observe that current SOTA models demonstrate approximately uniform performance across different programming languages and use cases, suggesting that current SOTA language models are sufficiently good to be used in a wide variety of coding environments with a similar level of performance. Additionally, the EFCB suite has not yet attained benchmark saturation - notably with Reverse-QA-Hallu still having more than 94% fail cases on average - making it a desirable choice for a benchmark to evaluate language models coding abilities.

Directions for future work include (i) defining the scope of model hallucination and evaluating different hallucination mitigation strategies, (ii) improving model question and answer capabilities with respect to GitHub pull requests as a ground truth measurement, (iii) developing methods to verify code runnability at inference time and (iv) evaluating models on niche languages and programming use cases with smaller contributor activity.

7 Limitations

7.1 Choice of language models.

One limitation in our current work is the narrow scope of our choice of language models. Given budget constraints, we chose two models that we believe are representative of the current state of the art for coding models, though we invite researchers to evaluate other state of the art models to evaluate the reproducibility of our reported results.

7.2 Generation of bug-free code.

Given that the premise of EFCB is to run model evaluations without the need of a dedicated coding environment, we rely on a language model as a judge to grade model generated code. Even though llm-as-a-judge approaches have shown reliable empirical results (Zheng et al., 2023), this approach lacks the ability to directly run model outputs of generated code. As a result, while EFCB can give a holistic indication of a model's ability to generate code, additional tests may be needed to ascertain its coding abilities to generate runnable, bug-free code.

7.3 Implicit information in the ground truth pull requests.

Another limitation of our current work is that we use human-authored pull requests that were written for code releases, rather than for model evaluation. As a result, each pull request title and description may contain implicit references that may not be apparent to models (or human evaluators) who do not have domain-specific knowledge of that particular GitHub repository. Thus, models with verbose generation may generate adjacent content that is relevant to the specific task, though these adjacent generated contents may be flagged as mistakes by the EFCB suite's scoring methods. However, given that these pull requests are sourced from an industry setting with production-ready development in mind, our EFCB code does offer an industry-like environment to evaluate models, as opposed to a synthetic, clean environment which may not be representative of industry-like environments.

7.4 Assessing the risk of data leakage.

By definition, data leakage occurs when the ground truth to a set of questions is contained in a model's training data. To minimize the risk of data leakage, we select a set of closed pull requests in reverse chronological order (starting with the most recent pull requests). As of this delay, given that there is often a gap between a model's training data cutoff date and its release date, we note that the EFCB's selection of a "most-recent-first" subset of pull requests minimizes the risk of data leakage. However, we acknowledge that this method is not a fool-proof

Model Name	mastodon	indigo	cloudflared	duckdb	tailscale	chroma	unweighted average
llama-3.3-70b	0.941	0.983	0.959	0.967	0.984	0.954	0.965
o4-mini	0.946	0.949	0.966	0.944	0.957	0.913	0.946

Table 9: Results for Reverse-QA-Hallu

method to shield this evaluation from data leakage in future models. As a result, potential solutions for future benchmarks that build on EFCB include regularly release minor versions that solely include recent pull requests (past the training data cutoff date) or building a closed dataset that contains pull requests made by human annotators independently of the original open-source GitHub pull requests.

7.5 Limitations on using an LLM-as-a-judge approach.

Three of the five EFCB tasks (MPR-Gen, Reverse-QA, and Reverse-QA-Hallu) use GPT-4.1-nano to assign a score to the response of the target model. Given that past work by Zheng et al. has shown that LLM-as-a-judge evaluations are in agreement with human annotator judgments upward of 80% of the time, we believe that using GPT-4.1-nano as a "judge LLM" is generally reliable. Some benefits of using GPT-4.1-nano as a judge include (i) being able to use GPT-4.1-nano to directly compare a predicted response with a ground truth sample, (ii) its speed relative to having a group of human evaluators judge the results and (iii) potentially less variance in its responses relative to a human group of annotators. However, without a human judge in the loop, we acknowledge that there is a risk of some scores being inaccurate. Thus, we interpret the scores as a measure of the similarity between the predicted response and the ground truth, rather than a measure of an execution-based evaluation. While Zheng et al.'s work suggests that there exists a sufficient amount of correlation between an LLM-as-a-judge's decision and a human annotator's judgement, we recognize that more empirical work on EFCB-related evaluations is needed to further validate the use of LLMs-as-a-judge for scoring coding-based evaluations.

8 Ethics Statement

The data collection process is based on six open-sourced GitHub repositories, for which we made sure that our work remained in accordance with each GitHub repository's respective license. Our dataset only contains publicly available code patches and pull request details, and does not contain any user information nor any sensitive information. Our dataset is available on GitHub (see the Reproducibility Statement). Should any reviewers raise any concerns, we will actively work to address those concerns and document changes made.

9 Reproducibility Statement

All of the EFCB questions and prompts are accessible at the following GitHub repository: https://anonymous.4open.science/r/efcb-fork-8965. Additional code examples are available upon request. All the model evaluation was done using OpenAI's public API endpoint and Together AI's public API endpoint.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2016. Neural machine translation by jointly learning to align and translate. *Preprint*, arXiv:1409.0473.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I. Wang. 2024. Cruxeval: A benchmark for code reasoning, understanding and execution. *Preprint*, arXiv:2401.03065.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Preprint*, arXiv:2009.03300.

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

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik

Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? *Preprint*, arXiv:2310.06770.

Hector J. Levesque, Ernest Davis, and L. Morgenstern. 2011. The winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*.

Junyi Li, Xiaoxue Cheng, Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2023. HaluEval: A large-scale hallucination evaluation benchmark for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6449–6464, Singapore. Association for Computational Linguistics.

Selvan Sunitha Ravi, Bartosz Mielczarek, Anand Kannappan, Douwe Kiela, and Rebecca Qian. 2024. Lynx: An open source hallucination evaluation model. *Preprint*, arXiv:2407.08488.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. Gpqa: A graduate-level google-proof qa benchmark. *Preprint*, arXiv:2311.12022.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. *Preprint*, arXiv:1409.3215.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need. *Preprint*, arXiv:1706.03762.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent abilities of large language models. *Preprint*, arXiv:2206.07682.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddartha Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. 2025. Livebench: A challenging, contamination-limited llm benchmark. *Preprint*, arXiv:2406.19314.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

A Judging Prompts

We use OpenAI's gpt-4.1-nano as a LLM-as-a-judge.

A.1 MPR-Gen

Judging prompt:

> You are an impartial judge. Please evaluate the following ground truth and predicted responses, and output a score between 0 and 10, based on the similarity between the ground truth and predicted response.
>
> 0 means not similar, and 10 means exactly similar.
>
> Only return the score as a number. Do not return anything else.

A.2 Reverse-QA

Judging prompt:

> You are an impartial judge. Please evaluate the following ground truth and predicted responses, and output a score between 0 and 10, based on the similarity between the ground truth and predicted response.
>
> 0 means not similar, and 10 means exactly similar.
>
> Only return the score as a number. Do not return anything else.

A.3 Reverse-QA-Hallu

Judging prompt:

> You are an impartial judge. Given the following ground truth and predicted response, output a score of either 0 or 1.
>
> If the predicted response contains information not present in the ground truth, output 1. Otherwise, output 0.
>
> Only return the score as a number. Do not return anything else.

B Selection of GitHub Repositories

We select six GitHub repositories from a variety of use cases and programming languages, that are as close as possible to codebases that reflect production-ready development:

- 2 of them related to databases (chroma and duckdb)
- 2 of them related to tunneling/networking (tailscale and cloudflared)
- 2 of them related to social media platforms (mastodon and indigo)

B.1 Chroma

Chroma is an open-source vector database primarily implemented in Rust and Python, with 20.3k stars.

Accessible at:
https://github.com/chroma-core/chroma

B.2 Cloudflared

Cloudflared is a CLI client that tunnels traffic from a Cloudflare network to user-defined origins.

Cloudflared is primarily implemented in Go with 10.7k stars.

Accessible at:
https://github.com/cloudflare/cloudflared

B.3 DuckDB

DuckDB is an analytical datbase based on SQL, primarily implemented in C+= with 30.1k stars.

Accessible at:
https://github.com/duckdb/duckdb

B.4 Indigo

Indigo is Bluesky's source code for its AT-protocol services. Indigo is primarily implemented in Go, , with 1.1k stars.

Accessible at:
https://github.com/bluesky-social/indigo

B.5 Mastodon

The Mastodon repository contains the full-stack code for the open-source Mastodon blogging platform. It is primarily implemented in Ruby, with 48.4k stars.

Accessible at:
https://github.com/mastodon/mastodon

B.6 Tailscale

Tailscale is a CLI tool for tunneling and peer-to-peer use cases. It is primarily implemented in Go and has 23k stars.

Accessible at:
https://github.com/tailscale/tailscale

C Sample Questions

C.1 GMCQ-Easy/Hard

GMCQ-Easy and GMCQ-Hard have the same question structure.

The difference is that the code patches for GMCQ-Easy are from four different pull requests, while GMCQ-Hard has the same code patch, but three of the four have two words (i.e. variables) swapped.

System:

"The user will ask which of the following 4 choices (A, B, C or D) is the best solution to the following GitHub pull request. Even the correct answer may have some irrelevant code inside of it. Only respond with a single letter A, B, C or D in uppercase."

User:

"Task Description:

fixes tailscale/tailscale16082˚˚should be true by default on iOS and Android. These platforms don't have˚CLI and aren't impacted by the original issue that prompted this change.—

Choice A: [[code]]

Choice B: [[code]]

Choice C: [[code]]

Choice D: [[code]]"

Ground Truth:

"C"

C.2 MPR-Gen

System:

"You are given a GitHub pull request title, description, and code that contains masked lines. Generate the code diff by replacing the masked lines with code that solves the GitHub pull request. The masked lines are marked with [[MASK]] in the code diff. Generate the entire code diff by replacing the masked lines of code with the correct lines of code, while including all the previous code. Only generate the code diff, do not generate anything else."

User:

"Title: feature/capture: fix wireshark decoding and add new disco frame types: Fix the wireshark lua dissector to support 0 bit position and not throw modulo div by 0 errors.

Add new disco frame types to the decoder.

Updates tailscale/corp29036

Code Diff:

[Code diff that contains a [[MASK]] for sime lines]

Ground Truth:
[Code path for those masked lines of code]

C.3 Reverse-QA

System:
"Given the following code diff, generate a title and a description of a GitHub pull request that best represents this code diff.your answer in the following format:: [[Title of the GitHub Pull Request]]: [[Description of the GitHub Pull Request]]"

User:
"tailcfg/tailcfg.go@@ -2446,6 +2446,14 @@ const (// native tailnet. This is currently only sent to Hello, in its// peer node list.4 NodeCapability = ñative-ipv4¨++// NodeAttrRelayServer permits the node to act as an underlay UDP relay+// server. There are no expected values for this key in NodeCapMap.+NodeCapability = ërelay:serverë¨++// NodeAttrRelayClient permits the node to act as an underlay UDP relay+// client. There are no expected values for this key in NodeCapMap.+NodeCapability = ërelay:clientë¨)// SetDNSRequest is a request to add a DNS record.—"

Ground Truth:
"Title: tailcfg: add relay client and server NodeAttr's: Updates tailscale/corp27502"

C.4 Reverse-QA-Hallu

System:
"Given the following code diff, generate a title and a description of a GitHub pull request that best represents this code diff.your answer in the following format:: [[Title of the GitHub Pull Request]]: [[Description of the GitHub Pull Request]]"

User:
"go.toolchain.rev@@ -1 +1 @@- [commit hash] + [commit hash] —"

Ground Truth:
"Title: go.toolchain.rev: bump to 1.24.3: Updates https://github.com/tailscale/corp/issues/28916"