Evaluating AI Generated Code

Patrick Adebiaye
University of North Texas
Denton, Texas, USA
patrickadebiaye@my.unt.edu

Ahmad Edaibat
University of North Texas
Denton, Texas, USA
ahmadedaibat@my.unt.edu

Dillon Edington
University of North Texas
Denton, Texas, USA
dillonedington@my.unt.edu

Giancarlo Ramirez
University of North Texas
Denton, Texas, USA
GiancarloRamirez@my.unt.edu

I. Introduction

Artificial intelligence refers to the ability of a digital computer to perform the same or similar tasks to those usually done by human beings. AI technology has massively advanced in recent years with technologies like OpenAI. In this study we will be addressing certain characteristics which differentiates AI models to other ones in comparison. Despite the prevalent use of AI, there are little formal studies evaluating their coding prowess. Understanding these differences is essential for developers to choose the most suitable model for their specific needs.

Elon Musk has emphasized the importance of computing power and access to data in evaluating AI capabilities, suggesting that a good measure of an AI's strength is its ability to discuss complex technologies such as battery advancements. Our paper will bring many features into light leading to accessing what AI model is best suited for certain prompts/problems. This is vital for both researchers and the public who often rely on AI tools. In addition, we hope to gain insight on how AI weaknesses/strengths that weren't initially obvious were concluded upon.

In this study, the role that software testing will serve will be in creating the different prompts that we will feed into our AI along with any deliberate ways in which we do so. Expanding on this, we will be testing across varied outputs across many AI models. In this paper, we will test both whether a problem can be solved by an AI model explicitly and whether it can detect a wrong approach to a problem.

Accessibility in digital environments refers to designing and developing products, services, and content that can be perceived, understood, and operated by a wide range of users, regardless of their physical, intellectual, or cognitive abilities. Despite the widespread accessibility adoption in web applications over the years, studies have shown that most web applications still present many accessibility. We will be accessing this aspect in using ChatGPT which has a mobile and web interface.

II. Background

These will include deep learning, neural network, and machine learning models. During this study, software testing will help us gain some perspective on the performance of these models. The following factors we will be exploring access to information access to information refers to how much data a given AI model can retrieve.

This will differ between AI models and lead to different responses to the same prompts across AI models. We will additionally will be exploring the consistency in output in certain prompts, designed to be recreated by such models. We will be exploring coding aspects of such applications also, specifically with some coding and non-coding problems. Coding problems will have criteria that is measured uniquely.

In addition, access to information can also refer to AI models' interfaces being designed in a way where a user can easily use it effectively. The relevancy of information is also crucial to this aspect of AI models, specifically that information is accurate and is updated continuously. Finally access to information can also refer to how data is used, oftentimes data from an user, ensuring that given prompts with sensitive information are not accessible to other users.

Expanding upon more, some AI models focus on information generation where as others might focus on information retrieval. Models trained to be based on information generation might often be outdated if not properly maintained. AI models based on information retrieval can have higher accuracy. In more specificity, some AI models might use deep learning methods like neural networks succeed in classifying large data that might not correlate to each other. These models learn from patterns given data and making these models good for tasks like natural language processing.

AI models might answer the same prompt in different ways as ML models use tokens that map to each item in the output. This is important to note when investigating AI models as we will be using a criteria that should not be affected by the variance in outputs to prompts.

Our study aims to address some of the uncertainty in AI generated outputs and hopefully in doing so, suggesting what aspects are to be accounted for when

using AI and what can be improved upon within AI models.

III. RelatedWork

ChatGPT had one notable difference in that it does not have the ability to read a current projects entire workspace/files [2] unlike GitHub Copilot and Amazon CodeWhisperer. This is incredibly important to note as copilot can be more accurate, as ChatGPT often especially if working with complex code will forget its context.

During this study we will also use dummy function names to assess how large language models perform under such prompting [2]. From this study we learned that using tools like ChatGPT require active prompting and proper explanation from the user and this in respect to word prompts or to code itself[2]. LLM's specifically ChatGPT is noticeably more sensitive to wording than LLM's like Claude's sonnet model.

During this paper, it was found (ChatGPT o4 mini), was the best at predictions but differences in this regard when compared to the other ChatGPT models were not significant. This context is worth noting as we will be reviewing o4 mini specifically in our study. It's stated that "Moreover, whilst ChatGPT can ingest documents that mix text and images, this comparison suggests that it is not powerful enough to leverage this capability for quality score prediction"[4]. This is important to note as a person trying recreate images of capabilities defined by code only visible by visual components.

Additionally hallucinating in outputs is common within models, which is when a model confidently makes inferences wrongly. It's important to understand how machine language model architecture is implemented. Machine models are trained on data often through web scraping, due to the vast amount of data it's possible for a model to infer wrongly. A model at least large language models do not know intrinsically an answer but are instead trained on how to infer an answer. This can refer to referencing a function that doesn't exist or even replacing a function that you implemented already within the prompting. These errors can heavily influence the time spent debugging and trying to understand what parts of generated code are meaningful and even usable.

To understand how context affects outputs the following is stated "Following the above, ChatGPT 4o could be encouraged to hallucinate by entering partial titles. For example, the query "Score the following journal article: An experiment to ..." generated a fictional article title (ChatGPT 4o: "Assuming the journal article is titled "An Experiment to Improve Teaching Methods in Higher Education":") and a report evaluating it at 3* (ChatGPT 4: "This article could be rated 3* because it is internationally excellent in terms of originality, significance, and rigor but might fall short of the highest standards needed for a 4* rating due to limited global adoption or transformation impact."). Thus, entering short titles has additional risks"[4].

Beyond generalizing large language models, the following is also stated specifically a field of study and using llm's within it, " General Science. Further improvements are needed in the application of LLMs in the field of chemistry. Castro Nascimento and Pimentel [18] presented five straightforward tasks from various subareas of chemistry to assess ChatGPT's comprehension of the subject, with accuracy ranging from 25 to 100This specifically shows the volatility of models and the given context in which validity differs from problem to problem.

To further explore the uses of LLM's the following is also stated, " In software engineering tasks, ChatGPT generally performs well and provides detailed responses, often surpassing both human expert output and SOTA output. However, for certain tasks such as code vulnerability detection and information retrieval-based test prioritization, the current version of ChatGPT fails to provide accurate answers, rendering it unsuitable for these specific tasks [181]"[4]. It's incredibly important to use AI as a tool and not as sole proprietor of your work as vulnerabilities, such as ec2 keys can be exposed ad used maliciously. In this comes an important point, when working with data intensive tasks or any tasks, understanding both principles of safe code and what code of generative AI performs.

In terms of formal measures we did not explore but are somewhat implicitly defined by our rubric the following give meaningful background, and it's stated that "Accuracy is a measure of how correct a model is on a given task. The concept of accuracy may vary in different scenarios and is dependent on the specific task and problem definition. It can be measured using various metrics such as Exact Match, F1 score, and ROUGE score."[4]. These measures are incredibly important and measurable when training AI, and combined with automated testing could provide much needed context as to accuracy and possible routes to improving AI in the future. In order to understand these measures the following will give short definitions:

- **Exact Match (EM)** is a metric used to evaluate whether the model's output in text generation tasks precisely matches the reference answer. In question answering tasks, if the model's generated answer is an exact match with the manually provided answer, the EM is 1; otherwise, it is 0.
- **F1 score** is a metric for evaluating the performance of binary classification models, combining the model's precision and recall. The formula for calculation is as follows: F 1 = 2×Precision×Recall

Precision+Recall .

- **ROUGE** is primarily employed to assess the performance of tasks such as text summarization and machine translation, involving considerations of overlap and matching between texts. [4]

Different model versions do make significant differences on the ability for AI to solve coding problems, as stated "Chat-GPT exhibits a strong capability for arithmetic reasoning by outperforming GPT-3.5 in the majority of tasks." [4]. Our paper will aim to evaluate context handling by providing abstract prompting where implementation is not fed into models, as this study states that models have much difficulty with abstract reasoning. Despite the metrics stated above manual testing and the feedback that humans can give is still essential as stated by the following "Human evaluation is a way to evaluate the quality and accuracy of model-generated results through human participation. Compared with automatic evaluation, manual evaluation is closer to the actual application scenario and can provide more comprehensive and accurate feedback."

IV. StudyDesign

To best compare skill at generating code among currently utilized LLMs, we composed 8 prompts and judged the response according to a the rubric below:

TABLE I
RUBRIC FOR EVALUATING CODE GENERATION QUALITY ACROSS VARIOUS CRITERIA.

Criterion	Poor (1)	Middling (3)	Excellent (5)
Compilation	Doesn't run at all	Modifications needed	Flawless
Optimal Solution	Obtuse solution	Acceptable solution	Optimal solution
Conciseness	<50% of lines essential	~60% essential	>80% essential
Self Explanation	Code mostly unexplained	Some unclear parts	Fully explained
Memory	Forgets core functionality	Some renaming/redeclaration	Fully consistent with prompt

Our prompting was purposely varied, within our test cases where we used some practical problems such as physics calculations and some code cases.

A. Manual Testing Process

To start, after collecting all the prompts, the team members began inputting these into the AI models and manually tracking the outputs. The human touch was crucial for assessing qualitative factors, such as practical usefulness, creativity, and accuracy. Our rubric evaluation required us to carefully examine every answer, paying attention to nuances beyond what automated tests would measure, such as expression, contextual appropriateness, and the applicability of the outputs.

After the AI generated responses, we evaluated each one with a pre-defined, thorough rubric. These criteria were clearly and objectively defined, with considerations such as clarity, creativity, coherence, accuracy, relevance, and quality of the overall response. The rubric provided a standardized framework for consistent and fair assessment, enabling direct comparisons between different AI models and ensuring transparency in the evaluation process.

Ratings, along with qualitative observations and comments, were carefully recorded. The extensive documentation was beneficial for allowing full review and analysis of AI performance. This results collection phase provided detailed insights that helped identify clear patterns, strengths, and weaknesses of each AI model, as well as areas needing improvement.

B. Advantages of Manual Testing

There were several reasons why we chose manual testing over automated testing. First, manual testing is best at capturing subjective, qualitative properties like creativity, coherence, and relevance—features that automated tests struggle to measure accurately.

Additionally, the complex nature of interpreting AI outputs required human judges who could grasp contextual subtleties and closely analyze responses in suitable situational contexts. One key benefit of manual testing was the ability to provide rich qualitative feedback, immersing the team in the nuances of the AI model's capabilities, thereby promoting actionable recommendations for improvement.

The end result was a thorough and context-specific evaluation of the AI models that could not have been achieved through automated testing alone. By conducting manual interactions followed by a detailed rubric-based analysis of results, we gained a deep understanding of the AI models' performance, strengths, and limitations. This approach significantly enhanced our insights, providing strong data for future iterations and supporting our broader objective of holistically evaluating AI capabilities and possible improvements.

C. Models

We selected a range of models that represent the current strength of the AI market. The following four models represent this array:

- OpenAI's ChatGPT o-3
- DeepSeek's DeepSeekR1
- Google's Gemini 2.0 Flash
- Anthropic's Claude 3.7 Sonnet

D. Prompts

To holistically judge each model on its ability to generate code, we chose a range of prompts that we believe represent a full spectrum of tasks that users might present to LLMs. LLMs were given one attempt at each prompt with no corrections or regenerations.

E. Prompt List

We developed the following working list of prompts to evaluate the models across different programming tasks and scenarios:

- **Visual Programming:** Generate code for animating a short video with continuous image rendering
- **Algorithm Implementation:** Create a functional bubble sort algorithm
- **Simulation Development:** Build a simple physics simulation
- **Code Comprehension:** Comment and explain existing uncommented code
- **Web Development:** Create specific HTML/CSS components from scratch
- **Debugging:** Identify and fix segmentation faults and out-of-bounds errors
- **Accessibility Enhancement:** Implement usability/accessibility features
- **DOM Modification:** Modify an existing HTML page to add screen reader functionality
- **Data Structures:** Implement linked lists, double linked lists, and binary trees in C++
- **Esoteric Languages:** Solve problems using less common programming languages
- **Beginner Assistance:** Format and organize poorly structured code for readability

These prompts were specifically selected to encompass a broad range of programming challenges that represent real-world usage scenarios. By covering diverse aspects of software development, from algorithmic tasks to accessibility features, we aim to thoroughly evaluate each model's capabilities across the spectrum of programming tasks.

Additionally, we explored a set of non-coding prompts, these involved the following:

- **Physics Questions** - Accuracy and Conceptual Understanding
- **Visual Consistency** - Image Storytelling with a Necklace in Anime
- **Usability Comparison** - ChatGPT App vs Browser Version

Each test case was evaluated using a standardized 5-point rubric for the following performance metrics:

- **Accuracy / Consistency**: Whether the model produced a correct or stable result.
- **Clarity / Explanation**: The depth and helpfulness of the model's internal explanation (when applicable).
- **Feature Support**: How well the model maintained or utilized features relevant to the task (e.g., visual continuity or interactive UI elements).
- **Correction/Recovery**: How effectively the model could recover or improve performance upon follow-up.
- **Overall Rating**: General effectiveness and user satisfaction in that task.

Testing was conducted by prompting the models with identical or equivalent instructions, measuring both initial output quality and any corrections needed upon revision. Results were documented and compared across models to determine robustness, consistency, and clarity.

V. Result-Discussion

Throughout our observations within our study we found out that for coding problems compilation except in edge cases, and this was found in some cases for Sonnet 3.7 and ChatGPT 03-mini. For example, we found that ChatGPT does tends to avoid using statements that simplify logic. In more specificity when asked to make a heap sort solution, ChatGPT did not provide early termination.

Our research paper tackles the ambiguity that many student that use AI face. In educating anyone, not just computer science students on how AI works, will elevate the prompts that a person gives and hence better results are likely to be generated.

In addition, the researching of AI models within this paper, will help anyone who is building their own AI model, projects related to machine learning, and related to deep learning. Despite not having direct knowledge of how the inner models work, we hope to unveil characteristics that a student constructing smart intelligence can keep in mind. Our paper could also provide a basis for those wanting surface level engagement with AI topics. This material could be refined into AI introduction workshops at a college.

In this study, we evaluated the performance of multiple AI models across three distinct tasks. Each task was designed to assess the model's ability to either solve domain-specific problems, generate consistent media, or support usability across platforms. The primary models tested were ChatGPT-4o, Gemini 1.5, Claude 3.7 Sonnet, and ChatGPT (legacy GPT-4).

In order to show the observations stated above I will be providing an exact test case prompt for one of the evaluations that we performed. Within this explanation we will use, the Clojure or other wise stated "rare language prompt". In this case specifically, we prompted "Make a webcrawler in clojure", within ChatGPT (mini-o4), Claude Sonnet 3.7, and Gemini 2.0 Flash.

During this we found, differences in the way that ChatGPT majorly differed in output compared to Claude 3.7 Sonnet's model, specifically ChatGPT did not account for state handlers or handling of individual webcrawling urls (to avoid reprocessing). In addition, a

bug which would possibly provide bugs within execution was found where the code below, would not stop running once a given url's depth has been exhausted. In providing context, I will provide a description of what the expectations for the webcrawler's functionality was expected.

The following steps represent the problem's solving scope:

- Starts from a given URL
- Removes scripts from URL and extracts plain text
- Queue's up new links (as in referring to new links found during exploration)
- All handling of these steps should be done concurrently Uses shared queue and shared state atom (to keep track of explored URLS)
- Terminates successfully (through a predefined timer or a certain condition)
- Outputs the URL queue after stoppage occurs and/or counter of URLS crawled (any signifier along these lines)

ChatGPT's excution stop code:

```
(if (or ( >= depth 0) ...),
```

provided the following faulty logic, as second condition is made to be non-existent. This is due to majority of the program depths being ¿= 0

Correct logical code:

```
(if (and (>= depth 0)
(not (contains? visited start-url)))
  ...)
```

Actively checks for not being visited using state atom and for depth ¿ 0 opposed to just depth itself, being extremely important for traversing URLS and avoiding faulty execution.

In addition to these findings, ChatGPT was found to lack consistent and much needed logical comments, especially inline function comments. For example the following code (simply extracts URLS and extracts words from its HTML elements), for someone not too familiar with clojure is hard to follow especially with no comments:

```
(defn process
  [{:keys [url content]}]
  (try
(let [html (enlive/html-resource
(java.io.StringReader. content))]
{::t #'handle-results
    :url url
    :links (links-from url html)
    :words (reduce (fn [m word]
    (update-in m [word] (fnil inc 0)))
            {}
          (words-from html))})
    (finally (run *agent*))))
```

To further explore the found tendencies of our selected AI models, we will also expand on our HTML 5 webpage prompt. In this prompt we aimed for our result to imitate the page we prompted (an simple user interface). During this prompting, we found that ChatGPT did not break down react components into reusable ones, an extremely important principle of react development. In addition the navigation bar components was missing. In analyzing this result, a big worry for developers trying to recreate webpages with ChatGPT, might case much overload and debugging especially due to the vast amounts of achieving the same goals in React. In this it's important to acknowledge the maintenance and possible refactoring that directly implementing this model's code might have.

In this specific case, ChatGPT was also found to lack in explaining specific implementation choices/purposes, similar to the clojure webcrawler prompt. Gemini's 2.5 model used similar approaches to how a developer would use React, as it used components, hooks, and component composition in order to reconstruct a given webpage as stated before.

This model faced similar unexplained code specifically code which wasn't an overview of a chunk of code. Claude's 3.7 Sonnet model All models did surprisingly respect all naming conventions within the initial prompting, something to consider, due to generative AI's tendency to lose context. It's worth noting that this is possibly due to the prompt not being exhaustive in logic or size.

One of our other coding prompts involved prompting models to make a heap sort solution in c++. During this we found that ChatGPT did not use std libraries specifically, heavily increasing the complexity/needed code. Specifically, the commenting needed as the logic behind how and why nodes are switched. Furthermore, no context such as the overview of heap sort itself is given within the output. This is extremely important when analyzing code generation especially when not providing library context. ChatGPT in this instance also struggled with commenting code as needed accordingly. Specifically, unmaintainable code is exacerbated by this problem and is something to keep in mind. ChatGPT specifically did provide the needed test cases and main function.

Gemini 2.0 Flash was model found to to contain excessive comments within parameter comments which were not necessary and implicitly explained by the name/instantiation. In this output some variables were re-declared unessential could be solve by encapsulating code more efficiently. Explanation within this model

were used in a way that anyone could understand heavy contrasting to ChatGPT's output. Sonnet specifically had similar output to other models, but had better commenting specifically in line commenting while keep conciseness within code output.

In addition to these code generation problems, we conducted problems which cannot be evaluated on our first rubric. We specifically conducted physics problems specifically, with electricity and magnetism problems. While working on electricity and magnetism problems—particularly involving magnetic fields in solenoids and the Right-Hand Rule—ChatGPT-4o initially gave incorrect answers. In one case, it missed a factor of 2 in the magnetic field equation and misapplied the right-hand rule, leading to an incorrect direction of the magnetic field inside the solenoid.

After switching models from ChatGPT-4o to GPT-4 o1, the answer was corrected and aligned with the expected result. Notably, Gemini 1.5 answered the same questions correctly without the need for a follow-up or correction. This indicates that Gemini was more consistent in understanding vector direction and physical laws, while ChatGPT-4o showed a pattern of visualization errors in 3D-oriented physics concepts.

Secondly we tested these type of abstract non coding problems by prompting an sequence of an image containing a necklace in order to analyze detail retainment, visual consistency, and narrative alignment. In this test, a short story-based image series was generated where a character wears a necklace throughout an anime-style sequence. ChatGPT-4o struggled to maintain the visual presence of the necklace across frames, sometimes removing it, changing its design, or shifting its position. This breaks immersion in storytelling and negatively impacts continuity.

On the other hand, Gemini 1.5 retained the necklace in all images, correctly aligning with the narrative context. Claude Sonnet did a decent job but made slight inconsistencies in color and detail. This test highlights a critical limitation in ChatGPT-4o's visual memory and object permanence, which could impact applications in animation, comics, or visual design tools.

Thirdly, we analyzed the usability between the browser version of ChatGPT and its mobile counter part. When comparing ChatGPT's browser interface vs. the mobile app, I found the browser version significantly more user-friendly for extended tasks. It offers better layout control, easier code editing, and more flexibility for viewing long threads or switching contexts. The browser also supports features like copy-pasting formatted tables, tabs, and integration with other research tools.

Conversely, the ChatGPT mobile app feels more limiting for serious work. The inability to multitask or view content side-by-side (such as reading from a PDF while chatting) reduces productivity. While convenient for short prompts or quick queries, it falls short for anything requiring deep interaction or debugging, especially when using coding features or referencing long messages.

The following figures summarize the results for our non-coding test cases:

TABLE II
COMPARISON OF AI MODELS' PERFORMANCE ON PHYSICS QUESTIONS, EVALUATING ACCURACY, CONCEPTUAL CLARITY, AND ABILITY TO CORRECT MISTAKES.

Model	Accuracy	Conceptual Clarity	Correction After Prompt Change
ChatGPT-4o	2/5	Medium (3/5)	Improved after switching to GPT-4
GPT-4o1	4/5	High (4/5)	Corrected Right-Hand Rule mistake
Gemini 1.5	5/5	High (4/5)	Correct answer similar to GPT-4o1

TABLE III
COMPARISON OF CHATGPT APP AND BROWSER VERSION ON INTERFACE RESPONSIVENESS, LAYOUT CLARITY, MULTITASKING ABILITY, AND OVERALL USABILITY.

Platform	Interface Responsiveness	Layout Clarity
ChatGPT Browser	High (5/5)	Clear (5/5)
ChatGPT App (Mobile)	Medium (3/5)	Limited (3/5)

VI. CONCLUSION

To summarize, we found that ChatGPT struggled in logical clarity, comment use, and visual memory in image generation. Gemini's 1.5 model outperformed all other models in reasoning and logical non-coding problem solving tasks. Claude sonnet, performed similarly to ChatGPT on all fronts but commenting where it performed slightly better. Additionally, we concluded that ChatGPT's interface within its web counterpart is apparent.

Firstly our study helps students find and even might help them conduct their own observations on what models to use depending on the task at hand. In terms, of models and what could possibly be improved upon, we believe is context given in an output where someone whom might not have background can understand an AI's output (through commenting). Out study, also emphasizes what possible time overhead relying on code generation solely might have when building a project, even though we explored intermediate problems for most test cases.

One of the takeaways to take from our research paper its that extremely important to plan out specifications of a solution, especially if it's code. Doing so can make

your prompting specific and therefore the output for an LLM, easier to debug if needed or modify to fit your needs.

LLM's like Claude sonnet might be better suited for users that ar not familiar with a particular coding language or have knowledge within a certain field respective to a problem. Claude sonnet was found to be significantly more forgiving in this aspect of answer generation. In terms of handling errors, ChatGPT and Gemini were able to recover where as Claude needed more in depth clarification.

LLM's in the future will need to improve upon less contextualized problems to really help solving ambagious tasks, especially due to the nature of coding problems. In terms of LLMS, they do well in solving individual tasks such as building heap sort but struggle within those that require multiple working parts.

These models should improve especially as technology becomes the epicenter for companies and managing the increasing amounts of global data and its uses.

Models in the future will most likely be better at explaining code and the steps towards the output more than now. This aspect is most likely the most limiting as often, even when asking an AI model to trace back how it concluded something, this can be incredibly inaccurate.

Additionally, it's worth mentioning that AI usage and its effectiveness is increased by the modularity of a program, as stated above AI models are very accurate when context is present which is easier to do when implementing specific modules/aspects of software opposed to an entire code base.

Adding upon this, AI models are very useful when debugging given programs, in terms of debugging and finding errors by yourself one can provide the context and significantly decrease the time that it takes to debug by aid via AI.

To finalize, our study did have some limitations, specifically in a real research project for example an industry backed one, I believe setting up data pipes and then performing automatic testing on certain prompts could provide very valuable insight. Specifically, this would allow any research team to further formalize coding results further.

REFERENCES

[1] Elon Musk's insights on AI capabilities and the importance of computing power and data access. Available at: https://www.reuters.com/technology/artificial-intelligence/elon-musk-says-grok-3-final-stages-outperforming-all-chatbots-2025-02-13

[2] Yetistiren, Burak, et al. "Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT." *arXiv preprint* arXiv:2304.10778, 2023.

[3] Wang, Ruotong, et al. "Investigating and Designing for Trust in AI-powered Code Generation Tools." In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, 2024.

[4] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. 2024. A Survey on Evaluation of Large Language Models. ACM Trans. Intell. Syst. Technol. 15, 3, Article 39 (June 2024), 45 pages. https://doi.org/10.1145/3641289