# Assessing LLMs' Capability to Understand TLA+

Houming Chen Daniel Hou Oh Jun Kweon

## 1 Introduction and Problem Statement

TLA+ [10] is a formal specification language used in system engineering to design and verify protocols. One obstacle in using TLA+ in an industrial setting is its notable learning curve, making it challenging for engineers to articulate TLA+ specifications for the protocols they designed.

For instance, an engineer who conceptualizes a protocol may want to express it in TLA+ code to validate the protocol using tools like TLC [24]. Nonetheless, errors may arise while attempting to represent the protocol in TLA+ code precisely. If the TLA+ code doesn't faithfully represent the intended protocol, then formal verification results become meaningless.

Therefore, we are interested in investigating how large language models (LLMs) can aid in TLA+ coding. Specifically, we aim to construct a benchmark to evaluate the capability of zero-shot LLMs in determining whether a piece of TLA+ code genuinely represents a described protocol.

## 2 Existing Works

Writing TLA+ is difficult, so there are existing efforts to simplify the process. PlusCal [12], a pseudocode language, offers a more straightforward writing experience and can be automatically converted into TLA+. There are also studies exploring the translation of C [16] and Java [13] code to TLA+.

On another note, large language models (LLMs) like ChatGPT and GPT-4 [19] have emerged as powerful tools for a wide range of programming tasks, including code generation [15], debugging assistance [21], and text-to-SQL conversion [20]. However, as far as we are aware, there has not been any research about applying LLMs to aid TLA+ writing.

## 3 Limitations in the Existing Research

While TLA+ [10] is powerful for formal verification, its high level of abstraction makes it challenging to be written. Moreover, as TLA+ is a specification language, its code cannot be compiled and executed. This complicates the debugging process and raises concerns about ensuring that the TLA+ code faithfully represents the protocol design.

PlusCal [12] offers a syntax closer to conventional programming languages, somewhat easing the difficulties of writing TLA+. Nonetheless, PlusCal remains relatively abstract and still cannot be executed. The translation of PlusCal to TLA+ also introduces further complexities for debugging.

Two studies have explored translating a small subset of C [16] and Java [13] code to TLA+. However, these subsets are too small for these studies to be applicable in real applications.

## 4 Insights

Motivated by the challenges of TLA+ coding and the remarkable capabilities of LLMs like ChatGPT, we initially explored using ChatGPT to translate programming languages like C++ into TLA+ or translate natural language protocol descriptions into TLA+. However, some simple tests on ChatGPT showed that it struggles to generate TLA+ code from scratch.

Therefore, we shifted our focus to a more manageable problem: assessing whether LLMs can comprehend TLA+ in the way that it can check the alignment between TLA+ code and its corresponding natural language description of a protocol.

If we discover that, through prompting engineering, advanced LLMs like GPT-4 can accurately determine if a TLA+ code segment truly represents a described protocol, then these LLMs could serve as a valuable tool for engineers. It would help them in ensuring their TLA+ code accurately mirrors their intended protocol design. On the other hand, if LLMs like GPT-4 do not perform well on this task, our benchmark remains valuable, providing insights that will guide the evolution and assessment of future LLMs.

## 5 Method and Timeline

Our timeline can be broken down into three phases: a research phase, a benchmark creation phase, and an evaluation phase.

1. Research Phase: Dedicate time to learning about TLA+ and TLA+ implementations of different protocols.
2. Benchmark Creation Phase: Create a dataset consisting of data pairs in the format $(N, T)$, where $N$ is a natural language description of a protocol, and $T$ is the corresponding TLA+ code. This dataset will include both "valid" pairs, where $T$ accurately represents $N$, and "invalid" pairs, where $T$ fails to accurately represent $N$.
3. Evaluation phase: Evaluate the ability of some zero-shot LLMs, like ChatGPT, to distinguish between "valid" and "invalid" pairs.

| Date | Milestones |
|---|---|
| Week of 10/02 | Complete proposal, learn TLA+ |
| Week of 10/09 | Get more familiar with TLA+ |
| Week of 10/16 | Implement Paxos in TLA+ |
| Week of 10/23 | Choose protocols, make mutations |
| Week of 10/30 | Discuss with instructors |
| Week of 11/06 | Get natural language descriptions of chosen protocols |
| Week of 11/13 | Finalize protocols and mutations |
| Week of 11/20 | Apply dataset/mutations on GPT |
| Week of 11/27 | Write up final report |

# Midterm Report

## 1 Overview

Our research mainly focuses on assessing the ability of LLMs to comprehend TLA+ in a zero-shot context. Originally, we planned to develop a benchmark to evaluate how well LLMs can determine whether a piece of TLA+ code accurately implements a given protocol. After some explorations and experiments, we found a viable and practical method to construct this benchmark. The basic idea of this benchmark involves adding logical "bugs" into the TLA+ specifications of various protocols and observing whether LLMs can detect these bugs in a zero-shot manner. Successful identification of these "bugs" by an LLM would indicate its capability to comprehend TLA+ protocol specifications to some extent. We name this benchmark as the Bug-Detection benchmark, the details of which will be elaborated later in this midterm report.

Additionally, we have incorporated another research objective into our project. During our discussion with Professor Chen during his office hours, he outlined an alternative benchmark for evaluating LLMs' comprehension of TLA+. This benchmark involves presenting an LLM with a TLA+ specification without disclosing the associated protocol. We would then check if the LLM can correctly identify the protocol described by the TLA+ specification. The ability of the LLM to correctly recognize the protocol would demonstrate its understanding of the TLA+ code to a certain degree. We have named this benchmark the Protocol-Identification benchmark and will describe it in detail later.

In the initial few weeks of our project, alongside dedicating time to learning basic TLA+ and familiarizing ourselves with well-known protocols, we started the development of these two benchmarks. Currently, we have preliminary versions of both benchmarks. Although in nascent stages, they are functional and already offer evaluations of the capability of LLMs in understanding TLA+ code. In this report, we:

1. Detail the design of both the Protocol-Identification and Bug-Detection benchmarks, which we utilize to assess the ability of LLMs to comprehend TLA+.
2. Present the evaluation results obtained from applying these benchmarks to both ChatGPT (GPT-3.5) and GPT-4, thereby gauging their performance.
3. Discuss the experiment results and describe our future plan.

## 2 The Protocol-Identification Benchmark

The general idea of this benchmark is to assess whether an LLM can identify the protocol a given TLA+ code snippet signifies. To accomplish this, we will provide an LLM with a TLA+ specification without revealing the corresponding protocol. We will then evaluate whether the LLM accurately identifies this protocol.

We further divide this benchmark into two sub-benchmarks: the Open-Ended Protocol-Identification benchmark and the Multiple Choice Protocol-Identification benchmark. In the Open-Ended benchmark, the LLM is asked to directly output the name of the protocol. In the Multiple Choice benchmark, an LLM is asked to select the correct answer from a provided list of options.

We now formally describe the Protocol-Identification benchmark. We first construct a set of protocols $P = \{p_1, p_2, ..., p_n\}$, where each $p_i$ is the name of a protocol. Then we find TLA+ specifications for these protocols. Let them be $s_1, s_2, ..., s_n$, where $s_i$ is a TLA+ specification for $p_i$.

For each $i$, we give the following two groups of LLM prompts to the LLM and check its response.

1. Open-Ended question prompt group

   System Prompt: "You are a TLA+ code reviewer. Your job is to review the TLA+ codes, and answer what protocol the code describes. "

   User Prompt: "Review the following code. What is the protocol the code describes?\n" + $s_i$.

2. Multiple Choice question prompt group

   System Prompt: "You are a TLA+ code reviewer. Your job is to review the TLA+ codes, and answer what protocol the code describes. "

   User Prompt: "Review the following code. What is the protocol the code describes? The only possible answers to this question are " + $p_1$ + ", " + $p_2$ + ", " + ... + $p_n$ + ".\n" + $s_i$

Finally, we assess the frequency with which the language model accurately identifies the protocol specified by the given TLA+ snippet. The model's score for the Open-Ended Protocol-Identification benchmark will be determined by its accuracy in recognizing the protocol for the Open-Ended question prompt group. Similarly, its score for the Multiple Choice Protocol-Identification benchmark will be based on its accuracy in identifying the protocol for the Multiple Choice question prompt group.

### 2.1 Selection of the Protocols

To construct the set of protocols $P$, we choose protocols based on the following three guidelines.

1. The TLA+ specification for the protocol should not be too long, so the prompt will fit in the input size limit for most LLM.
2. The protocol must be well-known prior to the cutoff date of the training dataset for the LLM to ensure that the model is well-informed about the protocol. This ensures that it is possible for the LLM to identify the protocol in a zero-shot manner.

For this midterm report, we have chosen eight protocols: two-phase commit (2PC) [23], Paxos [8], the Lamport's Bakery algorithm [7], Dijkstra's Mutex [6], Disk Paxos [9], Egalitarian Paxos [17], Fast Paxos [11], and Raft [18]. We use the TLA+ specifications for these protocols from [4].

## 2.2 Changing the Module and Variable Names

We noticed that in some TLA+ specifications from [4], some module names and variable names in the TLA+ code might directly contain the name of the protocol. This is undesirable as there is a potential for GPT to leverage these names to recognize protocols, rather than identifying the protocol by understanding the logic of the TLA+ code.

To address this issue, we have modified the module and variable names within these TLA+ specifications to some meaningless strings whenever the original module or variable name includes the protocol's name.

## 2.3 Experiment Results and Discussion

We evaluated ChatGPT (GPT3.5) and GPT-4 [19] with our Protocol-Identification Benchmark. The evaluation results are shown in Table 1. From the experiment results, we can see that both GPT 3.5 and GPT 4 are doing fairly well, as in each case the LLM has successfully identified the protocol in at least half of the test cases. This suggests that GPT-3.5 and GPT-4 possess at least a basic comprehension of the logic behind these TLA+ codes. Notably, GPT-4 outperforms GPT-3.5, achieving an almost perfect score on the multiple-choice prompts. This shows GPT-4's superior grasp of TLA+ compared to GPT-3.5. Nonetheless, neither model has attained a perfect score. Given that protocol identification is still a relatively easy task in the context of TLA+ comprehension, these findings demonstrate that there is still significant room for improvement before GPT can fully master TLA+.

Table 1: The experiment results of evaluating ChatGPT (GPT3.5) and GPT4 on our Protocol-Identification Benchmark. The 'Yes' in the table means the LLM has successfully identified the protocol, while 'No' means otherwise.

| Prompt Type | Open-Ended | | Multiple Choice | |
|---|---|---|---|---|
| GPT Version | GPT3.5 | GPT4 | GPT3.5 | GPT4 |
| 2PC | Yes | Yes | Yes | Yes |
| Bakery | Yes | Yes | No | Yes |
| Dijkstra's Mutex | No | No | Yes | No |
| Paxos | Yes | Yes | Yes | Yes |
| Disk Paxos | No | No | No | Yes |
| Egalitarian Paxos | No | Yes | No | Yes |
| Fast Paxos | No | Yes | Yes | Yes |
| Raft | Yes | Yes | Yes | Yes |
| Accuracy | 4/8 | 6/8 | 5/8 | 7/8 |

# 3 The Bug-Detection Benchmark

The Bug-Detection benchmark is another benchmark we developed to evaluate LLMs' understanding of TLA+. Its core objective is to determine whether LLMs can pinpoint errors in TLA+ code. To establish this benchmark, we integrated logical errors into the TLA+ specifications of various protocols to test the LLMs' error detection capabilities.

It's important to clarify that the bugs we've incorporated into the TLA+ codes are logical bugs, as opposed to more straightforward bugs such as typos or syntactic errors. Considering the potential complexity of bug detection, we currently only focus on small logical errors that alter the original code by typically one, and at most, a few lines of code. For instance, we may modify a single line in the TLA+ specification of a Two-Phase Commit (2PC) protocol, resulting in an incorrect 2PC that erroneously permits participants to commit, even if the coordinator has instructed them to abort.

Formally, we also construct a set of protocols $P = \{p_1, p_2, ..., p_n\}$. Then we find TLA+ specifications $s_1, s_2, ..., s_n$ for these protocols, where $s_i$ is a TLA+ specification for $p_i$. For each protocol $p_i$ and its TLA+ specification $s_i$, we create several modified version of $s_i$, which are $s_{i,1}, s_{i,2}, ..., s_{i,m}$. Each $s_{i,j}$ is very similar to $s_i$ and they usually differ by only a few lines of code. However, there are some critical logical differences between $s_{i,j}$ and $s_i$ so that $s_{i,j}$ does not accurately specify protocol $p_i$.

Then for each valid pair of $i$ and $j$, we give the following group of LLM prompts to the LLM and check its response.

- System Prompt: "You are a TLA+ code reviewer. "
- User Prompt: 'Review the following code. Then answer that whether it correctly implements the " + $p_i$ + " protocol. Answering YES or NO and then explain why.\n" + $s_{i,j}$

The LLM is expected to answer NO as we are giving it a buggy TLA+ specification. However, merely answering NO does not confirm that the LLM has accurately pinpointed the issue. We regard the bug as successfully identified only if the LLM not only responds with NO but also provides an accurate explanation for the errors in the TLA+ specification.

For this midterm report, we use three protocols - 2PC, Paxos, and Raft - for the Bug-Detection Benchmark. For each protocol, we created 5 buggy versions of the TLA+ specification. That is, in total, we have created 15 bugs, and detailed information about these bugs is shown in Appendix A.

## 3.1 Experiment Results and Discussion

We evaluated ChatGPT (GPT3.5) and GPT-4 [19] with our Bug-Detection Benchmark. The evaluation results are shown in Table 2. The experiments reveal that GPT-4 performs exceptionally well on this benchmark. A good performance on this benchmark proves again that GPT-4 is able to do some basic understanding of TLA+. However, on this benchmark,

GPT-3.5 exhibits a significantly weaker performance, suggesting it has a limited capability in this task. Notably, consistent with the prior experiment, the current experiment further corroborates GPT-4's superior understanding of TLA+ code over GPT-3.5.

Table 2: The experiment results of evaluating ChatGPT (GPT3.5) and GPT4 on our Bug-Detection Benchmark. Here $n/5$ means that the LLM has successfully identified the bugs in $n$ buggy versions.

| Protocol | GPT-3.5 Accuracy | GPT-4 Accuracy |
|---|---|---|
| 2PC | 3/5 = 60% | 5/5 = 100% |
| Paxos | 1/5 = 20% | 5/5 = 100% |
| Raft | 0/5 = 0% | 3/5 = 60% |

# 4 Limitations and Future Work

## 4.1 Adding More Protocols for the Benchmarks

At present, we only use eight protocols for the Protocol-Identification Benchmark and three protocols for the Bug-Detection Benchmark. We recognize that we are using too few protocols. To make our benchmark stronger and more comprehensive, we definitely need to add more protocols to our benchmark. However, a limiting factor is that it is very hard and time-consuming for us to learn new protocols and understand their TLA+ specifications. We will consult Professor Chen in the future to consider how to confront this challenge.

## 4.2 Adding More Bugs and Categorizing Benchmark Bugs into Varying Levels of Difficulty

For the comprehensiveness of the benchmark, we should also add more bugs per protocol. Ideally, we will have a large enough set of mutations per protocol such that we can even categorize them into varying levels of difficulty.

## 4.3 Evaluate More LLMs using Benchmarks

Currently, we are only using our benchmarks to evaluate ChatGPT (GPT3.5) and GPT4. We will try to evaluate more LLMs like Llama [22].

## 4.4 Other Possible Improvements for the Protocol-Identification Benchmark

For this benchmark, we modify the TLA+ code by altering module and variable names whenever they contain the protocol's name, see section 2.2. We are doing this to prevent LLMs from getting too much information from variable names while it is expected to comprehend the TLA+ code by logic. However, this approach may be insufficient, as other terms within the variable names might also provide clues to the LLMs. For instance, encountering variables named "votes," "candidates," or "electors" might lead the LLM to deduce that the protocol might be a consensus protocol.

# Final Report

## 1 Overview

Our midterm report already encompasses a fairly complete project. Consequently, throughout the latter half of the semester, our main focus has been on extending our existing work, rather than introducing entirely new benchmarks and experiments. We delve into some of the ideas outlined in section 4 from our midterm report. Additionally, we also have addressed some issues highlighted by Professor Chen in his feedback for our midterm report.

In this final report, we will first present several extensions we made to our existing work and make related discussions. The extensions include:

- We revisit the Protocol-Identification Benchmark, categorizing misidentifications into two tiers based on their severity.
- For the Bug-Detection Benchmark, we have formulated two sets of guidelines, each tailored to a specific category of bugs. We then redesign some of our original bugs and add some new bugs to our benchmark in alignment with these guidelines.
- We improve the Bug-Detection Benchmark by conducting multiple runs of each LLM for each bug. This approach introduces two new metrics: Run-wise Accuracy and Bug-wise Accuracy, offering a more detailed assessment of LLM performance.
- In addition to GPT-3.5 and GPT-4, we consider two other LLMs - PaLM [5] and Llama-2 [22]. We evaluate the four LLMs with the new Bug-Detection Benchmark.

Subsequently, we examine how our project compares to related works. We also compare the bug-detection abilities of LLMs in TLA+ specifications, as observed in our study, with their performance in detecting bugs in general programs, as reported in other studies.

Finally, we discuss the limitations of our study and conclude our findings.

## 2 Characterization of Missidentifications in the Protocol Identification Benchmark

Previously, we have used benchmarks to assess LLMs' abilities to identify TLA+ specs of well-known protocols. Following an analysis of the missidentifications made by LLMs, we categorized these errors into two types. *Minor Missidentifications* occur when a protocol is incorrectly identified by the LLM, but the inputted protocol and the identified protocol are logically related. Conversely, *Major Missidentifications* happen when the LLM incorrectly identifies a protocol, and there is no logical relation between the inputted protocol and the one identified by the LLM. For example, if the LLM identifies one Paxos-family protocol as another Paxos-family protocol, this should be considered a minor misidentification. We score a correct identification as 1, a minor misidentification as 0.5, and a major misidentification as 0. Then the performance and scores of the LLMs are shown in Table 3.

Table 3: The number of correct identifications, minor misidentifications, and major misidentifications, the raw accuracy, and the overall average score of ChatGPT (GPT3.5) and GPT-4 run on the open-ended and the multiple choice Protocol-Identification benchmark. In the table, we use ID as an abbreviation for identification.

| Prompt Type | Open-Ended | | Multiple Choice | |
|---|---|---|---|---|
| GPT Version | GPT3.5 | GPT4 | GPT3.5 | GPT4 |
| Correct ID | 4 | 6 | 5 | 7 |
| Minor MisID | 3 | 2 | 2 | 1 |
| Major MisID | 1 | 0 | 1 | 0 |
| Raw Accuracy | 4/8 | 6/8 | 5/8 | 7/8 |
| Average Score | 0.69 | 0.86 | 0.75 | 0.94 |

We believe that classifying misidentifications in this manner provides a better measure of LLMs' ability to identify TLA+ specs of protocols. Despite these changes, all discussions and conclusions regarding the performance of GPT-3.5 and GPT-4 on this benchmark, as presented in our mid-term report, are still applicable. This includes the key observation that GPT-4 significantly outperforms GPT3.5 in both the open-ended and the multiple-choice benchmark.

## 3 Standardize the Bug-Detection Benchmark

In the bug-detection benchmark, we introduce manually crafted bugs into TLA+ specifications of various protocols to assess the ability of LLMs in identifying these errors. Previously, our bug creation process was subjective and produced bugs of varied complexity. To standardize this process, we have now developed two sets of guidelines, each targeting a specific bug category:

- Class A Bug: Single Line Bug
  1. A Class A Bug involves a precise alteration in the correct code, characterized by either the modification of a single operator, the substitution of one variable for another, or the amendment of a solitary condition.
  2. Such a bug is confined to a singular line of code.
  3. It should result in a significant logical deviation from the original TLA+ specification, altering its fundamental logic.
- Class B Bug: Multi-line Bug
  1. A Class B Bug encompasses multiple alterations in the correct code, which could include redefining an

entire phase or step of the protocol or modifying several conditions.

2. A Class B Bug spans across multiple lines of code.
3. While involving multiple lines, a Class B Bug still represents a single, cohesive logical modification to the original TLA+ specification.

We then revise some of our original bugs and add new ones to our benchmark in alignment with these guidelines. We ensure that we have ten buggy versions of each TLA+ specification for every protocol, with five from Class A and five from Class B. We organize our buggy TLA+ specifications such that specification 1 through 5 each contains one Class A bug, while specification 6 through 10 each contains one Class B bug.

# 4 Evaluating More LLMs with the New Bug-Detection Benchmark

In the midterm report, we only test GPT-3.5 and GPT-4 [19] with the Bug-Detection Benchmark. In this report, we additionally evaluate PaLM [5] and Llama-2 [22] with the Bug-Detection Benchmark.

In the midterm report, we evaluated the performance of GPT-3.5 and GPT-4 using the OpenAI API using a personal API key and a Python script. To automate the process for PaLM and Llama-2, we extended our Python script to integrate APIs for both PaLM and Llama-2. We note that, while we used the official OpenAI API [2] for GPT and the official Google API [1] for PaLM, we were unable to find an official Llama-2 API. Instead, we use the Replicate API [3], which unofficially serves Llama-2 responses. You may find all of our scripts, protocols, buggy TLA+ codes, bug explanations, LLM responses, and more in this project's GitHub repository [1].

One limitation of the Replicate API is that it can only take a maximum of 4096 tokens as input. While this is sufficient for the 2PC and Paxos TLA+ codes, the Raft TLA+ codes exceeded this number, so we are unable to evaluate Llama-2 on the Raft TLA+ codes. We will mark results about Raft on Llama-2 as 'N/A' in the following sections.

## 4.1 Run LLMs On Each Buggy TLA+ Specification For Multiple Times

When running our benchmarks, we notice that for different trials of experiments, the same LLM will respond differently to the same input and prompts, resulting in variability in our benchmark measurements. To account for this non-determinism in LLMs' responses and to make the results more robust, we run each of the ten buggy TLA+ specifications for four times, resulting in a total of 40 runs per protocol for each LLM.

In this context, we are focusing on two types of accuracy:

- Run-wise accuracy: This measures the proportion of runs in which the LLM successfully identifies the bug. For instance, consider a scenario where we have two buggy TLA+ specifications for a single protocol, and we are evaluating the LLM's ability to detect these two bugs. If the bug 1 is explained correctly 0 out of 4 times and the bug 2 is explained correctly 3 out of 4 times, the total run-wise accuracy is $\frac{0+3}{4+4} = \frac{3}{8}$.
- Bug-wise accuracy: This reflects the proportion of bugs successfully identified by the LLMs. In this case, a bug is considered 'caught' by the LLM if it is correctly explained in at least one of the runs. Using the same example as above, if the bug 1 is explained correctly 0 out of 4 times and the bug 2 is explained correctly 3 out of 4 times, the total bug-wise accuracy is $\frac{0+1}{1+1} = \frac{1}{2}$. This is because the second bug is identified at least once, while the first bug is never identified, so the proportion of bugs successfully identified by the LLM is $\frac{1}{2}$.

## 4.2 Correct Explanations are as Important as Correct Results

In our midterm report, we emphasize that determining whether an LLM can successfully identify a bug requires more than merely assessing if the LLM deems the modified specification as buggy. It is also essential to examine the explanation provided by the LLM to ascertain whether it truly understands the nature of the bug. This subsection will delve deeper into this idea, leveraging our latest data.

If we simply check whether the LLMs are able to detect that the buggy code was incorrect, regardless of whether it was able to give a correct explanation of why it was incorrect. The results are shown in 4.

The exceptional performance of Llama-2, significantly outperforming other LLMs with almost perfect scores, might initially seem impressive. However, a closer and more thoughtful examination of these results uncovers serious issues. Consider a hypothetical scenario: if an LLM always labeled code as incorrect, it would achieve a 100% identification rate. But such a pattern doesn't necessarily indicate a deep understanding of the bugs. Even a simple Python script could be programmed to label all inputted TLA+ codes as incorrect. To truly assess whether LLMs' identification stems from a semantic comprehension of TLA+, we must also scrutinize the explanations they provide.

## 4.3 Experiment Results and Discussion

Based on the insight that identification alone is not representative of an LLM's ability to understand TLA+, we verify that the supporting explanations are also correct. Given four runs of ten buggy specifications across three protocols and four LLMs, this means that we have to manually review 480 responses (technically 440, since we are unable to run Raft on Llama-2). Given the large number of responses to manually

[1] https://github.com/ohjuny/LLM-TLA

review, a priority is to minimize the potential subjectivity of what is considered correct. As a result, we develop a simple set of guidelines for reviewing responses:

- If a response successfully identifies a buggy TLA+ specification as incorrect but gives multiple reasons, as long as these reasons contain the accurate explanation of the bug, we consider the response as correct.
- If a response contains more than three reasons, we only consider the first three reasons.

Tables 5 present the results of analyzing the explanations and the Run-wise accuracy. We notice that these results are significantly different from when we only evaluated identification. We notice that the performance of each model drops significantly. In particular, though, the performance of Llama-2 drops from 99% to 19%, making it no longer the best performing model. The best performing model is now GPT-4, which shows almost double the accuracy of the second and third best performers. We also note that GPT-3.5 drops from 34% to 0%. The decrease in accuracy across all models reflects the intuition that explanation is a significantly harder problem than identification, and the dramatic drop of GPT-3.5 and Llama-2 in particular reveal that it is possible for a model with little understanding of TLA+ to perform well on a purely identification-based benchmark.

In Table 6, we present the accuracy of models based on the complexity of the bugs. Surprisingly, the results do not show a clear trend of whether Class A bugs or Class B bugs are easier for LLMs to understand and correct. For each model, we observe a similar number of Class A and B bugs that are correctly identified and explained. This presents an opportunity to further categorize bugs based on more complex characteristics in the future.

Table 7 presents the bug-wise accuracy of each model. We note that bug-wise accuracy results in slightly inflated numbers in comparison to run-wise accuracy. This is expected given that bug-wise accuracy is more tolerant of incorrect explanations; only one out of four responses must be correctly explained for the bug to be deemed correctly explained. Despite the different notion of accuracy, though, the benchmark results in the same ordering of models; namely GPT-4, Llama-2, PaLM, GPT-3.5. This consistency gives greater confidence that the benchmark is distinguishing the models based on the models' understanding of TLA+ rather than a non-deterministic coincidence.

Table 4: Number of buggy TLA+ specs identified as incorrect out of 40 runs for each LLM and protocol and the Run-wise accuracy of each LLM

| | 2PC | Paxos | Raft | Run-wise Acc |
|---|---|---|---|---|
| GPT-3.5 | 1 | 0 | 40 | 41/120 = 34% |
| GPT-4 | 22 | 40 | 9 | 71/120 = 59% |
| PaLM | 0 | 40 | 40 | 80/120 = 67% |
| Llama-2 | 40 | 39 | N/A | 79/80 = 99% |

Table 5: Number of buggy TLA+ specs identified as incorrect with correct explanation and the Run-wise accuracy of each LLM. Each LLM had four runs on each buggy TLA+ specifications, which means 40 runs in total for each protocol.

| | 2PC | Paxos | Raft | Run-wise Acc |
|---|---|---|---|---|
| GPT-3.5 | 0 | 0 | 0 | 0/120 = 0% |
| GPT-4 | 8 | 29 | 5 | 42/120 = 35% |
| PaLM | 0 | 13 | 8 | 21/120 = 18% |
| Llama-2 | 0 | 15 | N/A | 15/80 = 19% |

Table 6: Number of buggy TLA+ specs identified as incorrect with correct explanation. Each LLM had four runs on each buggy TLA+ specifications, which means 20 runs of Class A bugs in total for each protocol and 20 runs of Class B bugs in total for each protocol.

| | GPT-3.5 | | GPT-4 | | PaLM | | Llama-2 | |
|---|---|---|---|---|---|---|---|---|
| Class | A | B | A | B | A | B | A | B |
| 2PC | 0 | 0 | 0 | 8 | 0 | 0 | 0 | 0 |
| Paxos | 0 | 0 | 19 | 10 | 8 | 5 | 9 | 6 |
| Raft | 0 | 0 | 3 | 2 | 4 | 4 | N/A | N/A |
| Total | 0/60 | 0/60 | 22/60 | 20/60 | 12/60 | 9/60 | 9/40 | 6/40 |

Table 7: Number of buggy TLA+ specs identified as incorrect with correct explanation in at least one out of four runs of each buggy TLA+ specs and the Bug-wise accuracy of each LLM.

| | 2PC | Paxos | Raft | Bug-wise Acc |
|---|---|---|---|---|
| GPT-3.5 | 0/10 | 0/10 | 0/10 | 0/30 = 0% |
| GPT-4 | 2/10 | 9/10 | 3/10 | 14/30 = 47% |
| PaLM | 0/10 | 4/10 | 2/10 | 6/30 = 20% |
| Llama-2 | 0/10 | 6/10 | N/A | 6/20 = 30% |

More detailed experiment results can be found in Appendix B and our GitHub repository https://github.com/ohjuny/LLM-TLA.

## 5 Related Works

### 5.1 General Bug Detection Benchmarks

The QuixBugs benchmark [14] is a compilation of 40 buggy programs, with each one presenting a buggy version of well-known algorithms like the Euclidean GCD algorithm, quicksort, and breadth-first search, etc. Each of these faulty implementations has a single-line logical bug [14]. The programs are translated into both Python and Java [14].

Contrary to QuixBugs' 40 distinct programs, each harboring one bug, our bug-detection benchmark focus on just three TLA+ specifications, adding 10 unique bugs into each. This

approach stems from the limited number of short, classical, and widely-used protocols, unlike the plethora of well-known algorithmic codes like GCD, quicksort, BFS, etc.

### 5.2 LLMs for General Bug Detection

Recent research [21] evaluated the zero-shot bug detection ability of ChatGPT (GPT-3.5) on QuixBugs [14]. In this study, ChatGPT was tasked with debugging each Python program in QuixBugs four times [21]. This procedure is similar to our approach, where we apply each Large Language Model (LLM) to every buggy TLA+ specification the same number of times. Consequently, it is both feasible and reasonable to compare the run-wise and bug-wise accuracy of ChatGPT in detecting bugs in general programs with the run-wise and bug-wise accuracy of various LLMs in identifying bugs in TLA+ specifications. The comparison is shown in Table 8.

Table 8: The comparison of the run-wise and bug-wise accuracy of ChatGPT (GPT-3.5) in detecting bugs in Python programs with the run-wise and bug-wise accuracy of various LLMs in identifying bugs in TLA+ specifications. The run-wise and bug-wise accuracy of ChatGPT (GPT-3.5) in detecting bugs in Python programs is calculated by the data in [21]. Other data are from this research, see section 4.3.

| Run-wise Acc | GPT-3.5 | GPT-4 | PaLM | Llama-2 |
|---|---|---|---|---|
| QuixBugs python program | 20% (32/160) | - | - | - |
| TLA+ Spec | 0% | 25% | 18% | 19% |

| Bug-wise Acc | GPT-3.5 | GPT-4 | PaLM | Llama-2 |
|---|---|---|---|---|
| QuixBugs python program | 47.5% (19/40) | - | - | - |
| TLA+ Spec | 0% | 47% | 20% | 30% |

The comparison reveals that while ChatGPT (GPT-3.5) demonstrates some proficiency in identifying bugs in short Python programs from QuixBugs, yet it struggles to do the same with TLA+ specifications. We think that there might be two possible reasons:

1. The Python programs in QuixBugs are implementations of relatively simple algorithms. Their logic is likely more straightforward compared to the complex logic of the three protocols we tested.

2. The programs in QuixBugs, which are not only based on famous and widely utilized algorithms but are also coded in Python, are likely more represented in the training data of LLMs.

Despite ChatGPT (GPT-3.5)'s inability to identify bugs in TLA+ specifications, the other three LLMs demonstrate some capacity to do so. This suggests that more advanced models might have a greater potential for debugging tasks, indicating a deeper understanding of the code.

## 6 Limitations and Future Work

Our research still has limitations. One limitation of our approach to measurements is the process of bug generation. The bug generation process involved manual adjustments to correct code, where each author was responsible for a single protocol. Thus, the difficulty of bugs for each protocol was left to each respective author's discretion, resulting in varying accuracy scores across different protocols. To counteract this, in the future, we can make the test sets for each protocol more consistent by involving all the authors to generate bugs for each protocol.

Another notable limitation is the benchmarks for explanation accuracy. When determining if an LLM's explanation for the protocol's bug was correct, each explanation was graded by the respective protocol's author. Because of this, the explanation accuracy for different protocols varied heavily. With more time, the accuracy variance can be mitigated by having each bug explanation be graded by all the authors.

An area of exploration that we are especially interested in is how each LLMs' responses would change if we had tried different prompts. Currently, our prompt is open-ended, accepting a wide range of answers. Due to the lack of time, we were only able to test our benchmarks against one prompt, but we are curious how our results would change if, for example, we asked for the LLMs to directly modify the TLA+ specs. or Our work can be expanded on by exploring how the behavior of the LLMs are affected with different prompts.

## 7 Conclusion

TLA+ is a difficult language to use and write for many developers. In this paper, we explored the capabilities of LLMs to understand TLA+ code. Specifically, we created and performed benchmarks tests on GPT-3.5, GPT-4, PaLM, and Llama-2. These benchmarks evaluate the LLMs' capabilities in identifying TLA+ specs of common protocols and identifying and explaining buggy TLA+ specs. While there were limitations to the approach in measuring their capabilities, our findings indicate that LLMs have a basic understanding of TLA+. While there is still much to explore in this field, we believe the development of LLMs can eventually lead to them being a suitable and powerful tool to aid developers in writing and debugging TLA+ code.

## 8 Acknowledgement

We thank Professor Chen for his help and guidance.

We identified one data point in Table 1 was wrong in our midterm report. Specifically, the performance of GPT3.5 in identifying Dijkstra's Mutex protocol in the Multiple Choice benchmark was incorrectly written as 'NO' when it should have been 'YES'. This error has been corrected. Despite this adjustment, the overall discussion and conclusions of this paper remain consistent and unaffected.

## References


[1] Google palm api. https://developers.generativeai.google/guide/palm_api_overview. Accessed: 2023-12-03.

[2] Openai gpt api. https://platform.openai.com/docs/guides/text-generation. Accessed: 2023-12-03.

[3] Replication llama-2 api. https://replicate.com/blog/run-llama-2-with-an-api. Accessed: 2023-12-03.

[4] Tla+ examples. https://github.com/tlaplus/Examples, 2023.

[5] CHOWDHERY, A., NARANG, S., DEVLIN, J., BOSMA, M., MISHRA, G., ROBERTS, A., BARHAM, P., CHUNG, H. W., SUTTON, C., GEHRMANN, S., SCHUH, P., SHI, K., TSVYASHCHENKO, S., MAYNEZ, J., RAO, A., BARNES, P., TAY, Y., SHAZEER, N., PRABHAKARAN, V., REIF, E., DU, N., HUTCHINSON, B., POPE, R., BRADBURY, J., AUSTIN, J., ISARD, M., GUR-ARI, G., YIN, P., DUKE, T., LEVSKAYA, A., GHEMAWAT, S., DEV, S., MICHALEWSKI, H., GARCIA, X., MISRA, V., ROBINSON, K., FEDUS, L., ZHOU, D., IPPOLITO, D., LUAN, D., LIM, H., ZOPH, B., SPIRIDONOV, A., SEPASSI, R., DOHAN, D., AGRAWAL, S., OMERNICK, M., DAI, A. M., PILLAI, T. S., PELLAT, M., LEWKOWYCZ, A., MOREIRA, E., CHILD, R., POLOZOV, O., LEE, K., ZHOU, Z., WANG, X., SAETA, B., DIAZ, M., FIRAT, O., CATASTA, M., WEI, J., MEIER-HELLSTERN, K., ECK, D., DEAN, J., PETROV, S., AND FIEDEL, N. Palm: Scaling language modeling with pathways, 2022.

[6] DIJKSTRA, E. W. Solution of a problem in concurrent programming control. *Commun. ACM 8*, 9 (sep 1965), 569.

[7] LAMPORT, L. A new solution of dijkstra's concurrent programming problem. *Communications of the ACM 17*, 8 (1974), 453–455.

[8] LAMPORT, L. The part-time parliament. *ACM Trans. Comput. Syst. 16*, 2 (may 1998), 133–169.

[9] LAMPORT, L. The part-time parliament. *ACM Trans. Comput. Syst. 16*, 2 (may 1998), 133–169.

[10] LAMPORT, L. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, June 2002.

[11] LAMPORT, L. Fast paxos. *Distrib. Comput. 19*, 2 (oct 2006), 79–103.

[12] LAMPORT, L. The pluscal algorithm language. In *International Colloquium on Theoretical Aspects of Computing* (2009), Springer, pp. 36–60.

[13] LAU, H., AND NESTMANN, U. Java goes tla+. In *2011 Fifth International Conference on Theoretical Aspects of Software Engineering* (2011), IEEE, pp. 117–124.

[14] LIN, D., KOPPEL, J., CHEN, A., AND SOLAR-LEZAMA, A. Quixbugs: A multi-lingual program repair benchmark set based on the quixey challenge. In *Proceedings Companion of the 2017 ACM SIGPLAN international conference on systems, programming, languages, and applications: software for humanity* (2017), pp. 55–56.

[15] 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).

[16] METHNI, A., LEMERRE, M., BEN HEDIA, B., HADDAD, S., AND BARKAOUI, K. Specifying and verifying concurrent c programs with tla+. In *Formal Techniques for Safety-Critical Systems: Third International Workshop, FTSCS 2014, Luxembourg, November 6-7, 2014. Revised Selected Papers 3* (2015), Springer, pp. 206–222.

[17] MORARU, I., ANDERSEN, D. G., AND KAMINSKY, M. There is more consensus in egalitarian parliaments. In *Proc. 24th ACM Symposium on Operating Systems Principles (SOSP)* (Farmington, PA, Nov. 2013).

[18] ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference* (USA, 2014), USENIX ATC'14, USENIX Association, p. 305–320.

[19] OPENAI, R. Gpt-4 technical report. *arXiv* (2023), 2303–08774.

[20] POURREZA, M., AND RAFIEI, D. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015* (2023).

[21] SOBANIA, D., BRIESCH, M., HANNA, C., AND PETKE, J. An analysis of the automatic bug fixing performance of chatgpt. *arXiv preprint arXiv:2301.08653* (2023).

[22] 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* (2023).

[23] WEISS, C. B., LYNCH, N. A., AND TUTTLE, M. R. Two-phase commit: A reliable distributed transaction protocol. *ACM Comput. Surv. 15*, 3 (Sept. 1983), 287–294.

[24] YU, Y., MANOLIOS, P., AND LAMPORT, L. Model checking tla+ specifications. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods* (1999), Springer, pp. 54–66.

# A Appendix A

Table 9: A list of bugs we created for the Bug-Detection Benchmark. (Note that this table is for the midterm report. Details and explanations of the bugs for the final version can be found in our Github Repo https://github.com/ohjuny/LLM-TLA)

| Protocol | The Created Bug |
|---|---|
| 2PC | Does not send abort message when aborting. |
| 2PC | Changed canCommit from "for all" to "there exists". |
| 2PC | Send abort message instead of commit message when trying to commit. |
| 2PC | Messages are not being sent when aborting in TMAbort. |
| 2PC | Removed commit and abort representions in the transaction manager. |
| Paxos | Removed check for current ballot is equal to new ballot in phase 1b. |
| Paxos | RM state will be commit even if it should be abort. |
| Paxos | Removed RMprepare in next state action statement, should never reach prepare stage. |
| Paxos | Maximum statement does not correctly return maximum. |
| Paxos | No messages are sent, send(m) always returns true. |
| Raft | Does not change to Candidate after timing out. |
| Raft | Become a follower instead of a leader. |
| Raft | Become a leader when not reaching 1/2 votes. |
| Raft | Voting as a follower instead of candidate. |
| Raft | Do nothing when vote is granted. |

# B Appendix B

Table 10: Number of buggy 2PC TLA+ specifications identified as incorrect. Each LLM had four runs on each buggy 2PC TLA+ specifications.

| | GPT-3.5 | GPT-4 | PaLM | Llama-2 | Total |
|---|---|---|---|---|---|
| Bug. 1 | 0/4 | 1/4 | 0/4 | 4/4 | 5/16 |
| Bug. 2 | 0/4 | 3/4 | 0/4 | 4/4 | 7/16 |
| Bug. 3 | 0/4 | 4/4 | 0/4 | 4/4 | 8/16 |
| Bug. 4 | 0/4 | 3/4 | 0/4 | 4/4 | 7/16 |
| Bug. 5 | 1/4 | 1/4 | 0/4 | 4/4 | 6/16 |
| Bug. 6 | 0/4 | 4/4 | 0/4 | 4/4 | 8/16 |
| Bug. 7 | 0/4 | 2/4 | 0/4 | 4/4 | 6/16 |
| Bug. 8 | 0/4 | 4/4 | 0/4 | 4/4 | 8/16 |
| Bug. 9 | 0/4 | 0/4 | 0/4 | 4/4 | 4/16 |
| Bug. 10 | 0/4 | 0/4 | 0/4 | 4/4 | 4/16 |
| Total | 1/40 | 22/40 | 0/40 | 40/40 | |

Table 11: Number of buggy Paxos TLA+ specifications identified as incorrect. Each LLM had four runs on each buggy Paxos TLA+ specifications.

| | GPT-3.5 | GPT-4 | PaLM | Llama-2 | Total |
|---|---|---|---|---|---|
| Bug. 1 | 0/4 | 4/4 | 4/4 | 4/4 | 12/16 |
| Bug. 2 | 0/4 | 4/4 | 4/4 | 4/4 | 12/16 |
| Bug. 3 | 0/4 | 4/4 | 4/4 | 4/4 | 12/16 |
| Bug. 4 | 0/4 | 4/4 | 4/4 | 4/4 | 12/16 |
| Bug. 5 | 0/4 | 4/4 | 4/4 | 4/4 | 12/16 |
| Bug. 6 | 0/4 | 4/4 | 4/4 | 4/4 | 12/16 |
| Bug. 7 | 0/4 | 4/4 | 4/4 | 4/4 | 12/16 |
| Bug. 8 | 0/4 | 4/4 | 4/4 | 4/4 | 12/16 |
| Bug. 9 | 0/4 | 4/4 | 4/4 | 4/4 | 12/16 |
| Bug. 10 | 0/4 | 4/4 | 4/4 | 3/4 | 12/16 |
| Total | 0/40 | 40/40 | 40/40 | 39/40 | |

Table 12: Number of buggy Raft TLA+ specifications identified as incorrect. Each LLM had four runs on each buggy Raft TLA+ specifications.

| | GPT-3.5 | GPT-4 | PaLM | Llama-2 | Total |
|---|---|---|---|---|---|
| Bug. 1 | 4/4 | 0/4 | 4/4 | N/A | 8/12 |
| Bug. 2 | 4/4 | 2/4 | 4/4 | N/A | 10/12 |
| Bug. 3 | 4/4 | 1/4 | 4/4 | N/A | 9/12 |
| Bug. 4 | 4/4 | 0/4 | 4/4 | N/A | 8/12 |
| Bug. 5 | 4/4 | 1/4 | 4/4 | N/A | 9/12 |
| Bug. 6 | 4/4 | 0/4 | 4/4 | N/A | 8/12 |
| Bug. 7 | 4/4 | 3/4 | 4/4 | N/A | 11/12 |
| Bug. 8 | 4/4 | 0/4 | 4/4 | N/A | 8/12 |
| Bug. 9 | 4/4 | 1/4 | 4/4 | N/A | 9/12 |
| Bug. 10 | 4/4 | 1/4 | 4/4 | N/A | 9/12 |
| Total | 40/40 | 9/40 | 40/40 | N/A | |

Table 13: Number of buggy 2PC TLA+ specs identified as incorrect with correct explanations. Each LLM had four runs on each buggy 2PC TLA+ specifications.

| | GPT-3.5 | GPT-4 | PaLM | Llama-2 | Total |
|---|---|---|---|---|---|
| Bug. 1 | 0/0 | 0/1 | 0/0 | 0/4 | 0/5 |
| Bug. 2 | 0/0 | 0/3 | 0/0 | 0/4 | 0/7 |
| Bug. 3 | 0/0 | 0/4 | 0/0 | 0/4 | 0/8 |
| Bug. 4 | 0/0 | 0/3 | 0/0 | 0/4 | 0/7 |
| Bug. 5 | 0/1 | 0/1 | 0/0 | 0/4 | 0/6 |
| Bug. 6 | 0/0 | 4/4 | 0/0 | 0/4 | 4/8 |
| Bug. 7 | 0/0 | 0/2 | 0/0 | 0/4 | 0/6 |
| Bug. 8 | 0/0 | 4/4 | 0/0 | 0/4 | 4/8 |
| Bug. 9 | 0/0 | 0/0 | 0/0 | 0/4 | 0/4 |
| Bug. 10 | 0/0 | 0/0 | 0/0 | 0/4 | 0/4 |
| Total | 0/1 | 8/22 | 0/0 | 0/40 | |

Table 14: Number of buggy Paxos TLA+ specs identified as incorrect with correct explanations. Each LLM had four runs on each buggy Paxos TLA+ specifications.

| | GPT-3.5 | GPT-4 | PaLM | Llama-2 | Total |
|---|---|---|---|---|---|
| Bug. 1 | 0/0 | 4/4 | 0/4 | 4/4 | 4/12 |
| Bug. 2 | 0/0 | 4/4 | 0/4 | 1/4 | 0/12 |
| Bug. 3 | 0/0 | 4/4 | 4/4 | 0/4 | 8/12 |
| Bug. 4 | 0/0 | 4/4 | 4/4 | 4/4 | 7/12 |
| Bug. 5 | 0/0 | 3/4 | 0/4 | 0/4 | 6/12 |
| Bug. 6 | 0/0 | 0/4 | 0/4 | 0/4 | 5/12 |
| Bug. 7 | 0/0 | 1/4 | 0/4 | 0/4 | 6/12 |
| Bug. 8 | 0/0 | 4/4 | 1/4 | 4/4 | 7/12 |
| Bug. 9 | 0/0 | 2/4 | 4/4 | 1/4 | 4/12 |
| Bug. 10 | 0/0 | 3/4 | 0/4 | 1/3 | 4/11 |
| Total | 0/0 | 29/40 | 13/40 | 15/39 | |

Table 15: Number of buggy Raft TLA+ specs identified as incorrect with correct explanations. Each LLM had four runs on each buggy Raft TLA+ specifications.

| | GPT-3.5 | GPT-4 | PaLM | Llama-2 | Total |
|---|---|---|---|---|---|
| Bug. 1 | 0/4 | 0/0 | 0/4 | N/A | 0/8 |
| Bug. 2 | 0/4 | 2/2 | 0/4 | N/A | 2/10 |
| Bug. 3 | 0/4 | 1/1 | 4/4 | N/A | 5/9 |
| Bug. 4 | 0/4 | 0/0 | 0/4 | N/A | 0/8 |
| Bug. 5 | 0/4 | 0/1 | 0/4 | N/A | 0/9 |
| Bug. 6 | 0/4 | 0/0 | 0/4 | N/A | 0/8 |
| Bug. 7 | 0/4 | 2/3 | 0/4 | N/A | 2/11 |
| Bug. 8 | 0/4 | 0/0 | 0/4 | N/A | 0/8 |
| Bug. 9 | 0/4 | 0/1 | 0/4 | N/A | 0/9 |
| Bug. 10 | 0/4 | 0/1 | 4/4 | N/A | 4/9 |
| Total | 0/40 | 5/9 | 8/40 | N/A | |