Verifying Reasoning Ability of Large Language Models via SAT Problems

Ken Shi
Department of Computer Science
University of Toronto
kenshi@cs.toronto.edu

Abstract

The evaluation of Large Language Models (LLMs) regarding their inductive reasoning capabilities is a trending yet complex topic within the research community. Developing standards for this domain poses challenges due to the broad spectrum of distinct reasoning tasks encompassed by induction. This project aims at verifying the inductive abilities of LLMs through the lens of SAT solving, a fundamental and thoroughly investigated reasoning problem. This project sets up a framework for assessing LLMs' performance in SAT solving through designing a pipeline of SAT problem evaluation by LLMs. This project performs a few simple experiments on SAT solving with LLMs and draws a few counter-intuitive conclusion about LLM's reasoning ability from the angle of SAT solving.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable proficiency across a variety of tasks in Natural Language Processing (NLP). The advancement of LLMs seems to have knocked down most of the well-established challenges defined in the early age of Artificial Intelligence [1], including search, pattern-recognition, planning and learning. However, whether we have solved the task of induction is still up for debate. This is largely because of the extensive range of tasks classified under induction. Inductive behaviors in an LLM can be observed through its ability of question-answering, natural language understanding, arithmetic reasoning, symbolic reasoning, etc. Nevertheless, the community has yet to agree on a unified view of evaluating inducibility.

Among those tasks that may contribute to concluding LLMs' ability of reasoning, the Boolean Satisfiability (SAT) problem stands out as a fundamental yet potentially effective task for evaluating a model's ability of automated reasoning. Since most of the state-of-the-art SAT solvers are deterministic, they can easily scale up with the size of the queries. However, one of the weaknesses LLMs have demonstrated was long-context handling. Thus, SAT is one of the few challenges yet to be fully addressed by LLMs.

This project aims to take on verification of LLMs' inductive ability through the angle of SAT solving. Specifically, the project will attempt to build a pipeline that enables conversational LLMs to solve SAT problems and evaluate their performance with a few implementation of traditional SAT solvers.

Through the experiments, this project draws a few conclusions on the general trend of LLM's ability on solving SAT problems as its complexity increases. An counter-intuitive finding in this project showed that line-of-thought prompting will in fact reduce the recall of the response, which hinders the performance of LLM SAT solving.

2 Related Work

There has been several studies that have aimed to evaluate LLMs with specific inductive tasks. Yuan et al, 2023 [2] evaluates a series of LLMs on solving mathematical problems. Qian et al, 2022 [3] explores some potential limitations of LLMs in arithmetic and symbolic reasoning by constructing a series of challenging examples. Dziri et al, 2023 [4] evaluates transformer based LLMs with a series of induction tasks including multi-digit multiplication, logic grid puzzles, and a classic dynamic programming problem. These evaluation schemes and metric are closely related to this project. Lewkowycz at al, 2022 [5] presented Minerva, a PaLM based LLM that is trained on mathematical problems. Similar approaches can be used to finetune accessible LLMs when needed.

This project will also incorporate Chain-of-thought Prompting [6] in addition to feeding SAT problems plainly. Imani et al, 2023 [7] presented MathPrompter that uses Chain-of-thought Prompting for mathematical problems. This project will draw inspiration from its methodology in constructing the evaluation pipeline.

3 Method Description and Justification

3.1 Pipeline Description

The pipeline used for evaluation and analysis is constructed by combining the following components: A SAT Generator, a Prompt Generator, a Response Collector and a Response Evaluator.

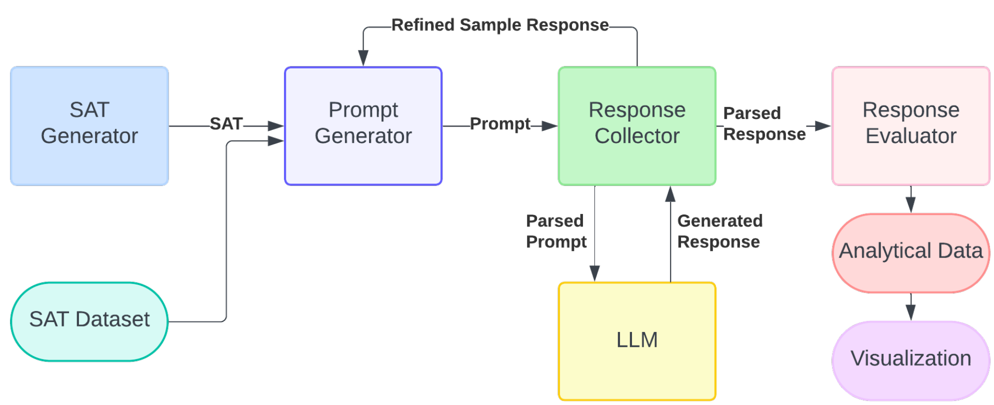

Figure 1: A high-level demonstration of the pipeline used for evaluations.

3.1.1 SAT Generator

This component generates a set of 3-SAT problems of n variables and k clauses. The generator uses a simple logic that randomly picks variables and assign random literals to them, and flips the literals to ensure that at least one literal aligns with the true assignment. This component is replaced with a simple parser when an existing dataset is used to provide SAT problems.

3.1.2 Prompt Generator

This component generates prompts of different styles. It fetches different settings (e.g. only gives single solution, use line-of-thoughts, use DIMACS format, etc) and constructs prompt with minimum paraphrasing for variations. For more complicated prompts, sample responses parsed by response collector is also fetched into prompt constructions.

3.1.3 Response Collector

This component sends the generated prompt to the selected LLM's API, collects their response and parses the response to a unified format. It is observed that without strict format restrictions, GPT models tend to use a varieties of response format. The response collector has been refined to adapt nearly all the possible solution format it produces with hundreds of case studies conducted. In addition to further evaluation, those responses will also be fed into prompt generators for more complicated prompts.

3.1.4 Response Evaluator

This component further parses the collected response to obtain at least one assignment produced by the LLM and evaluates whether it is a valid assignment for the given SAT problem. A few variations of the modules are written to adapt to the different settings and produce appropriate visualizations.

3.2 Validity of Reasoning Verification w/ SAT

SAT problems are one of the most well defined search based reasoning tasks in Computer Science. It is often ignored in the reasoning ability evaluations. However, there are a few advantages in using SAT problem to verify LLM's reasoning abilities.

3.2.1 Quantifiable Difficulty

A main advantage SAT problems have is that its complexity is easily quantifiable. In Dziri et al., 2023 [4], they defined "depth" that describes the level of difficulties for logic puzzles. Conveniently, in SAT problem, the complexity / difficulty of a problem is easily defined by the amount of variables (n), the amount of clauses (k) and the length of the clauses (m).

3.2.2 Data Leakage Handling

Because of the fact that many LLM's training data is not publicly disclosed, there are often concerns about evaluation done on data that is used for training. However, since SAT problems can be easily generated, it is guaranteed to be safe from having leaked into LLM's training data.

3.2.3 Out of Domain Problem Construction

In Chain-of-Thought Prompting, an Out-of-Domain (OOD) problem is referring to problems that goes out of training corpus' grammatical patterns. For example, if we consider finding the first letter of each word in a two-word name as in-domain, then finding the first letter of each word in a four-word name would be considered as OOD. In SAT, we can simply raise the amount of variables or clauses to carry the task from in-domain to OOD.

4 Evaluation

In this project, two popular LLMs developed by OpenAI, GPT-3.5-Turbo and GPT-4 will be used as the LLM component in the pipeline. The two models will be accessed via API calls instead of using chatGPT interfaces, as we speculate the latter would solve SAT problems via program synthesis instead of the traditional language generation.

4.1 GPT Models on SATLIB

The first experiment attempts to test the chosen LLM on solving existing SAT libraries. The choice of dataset is SATLIB [8], which is one of the most well established SAT problem dataset available. Through this experiment, we can get a taste of how SAT problems are currently handled by LLMs.

Surprisingly, even the simplest subset of SATLIB, uf20-91, both models failed to solve any cases among the 1000 problem instances in this subset. The prompt specifically asked the model to return an assignment that would satisfy the given instance. Although the recall of a proper assignment was 100%, the solve rate with those assignments was stunningly zero.

This clearly indicates that SATLIB was not used directly in training. In addition, this experiment sets an upper bound for later experiments that the amount of variable n should be bounded by 20 and the amount of clauses should be roughly bounded by the same proportion in which 91 instance is to all the possible 3-SAT clause combinations of a given n.

4.2 GPT Models on Generated SAT

Since the previous experiment failed completely, this set of experiments will look at SAT problems of much smaller scale. Specifically, the amount of variable n is chosen to be 3, 5, 10 respectively.

This set of experiments will look at two general categories of prompts, "standard" and "complex", which refers to (1) simple prompting asking for single assignment only and (2) more complex prompting that encourages the model to provide line-of-thoughts, respectively. This set of experiments will focus on comparing the two LLMs on each of those categories of prompt.

4.2.1 uf3

For $n = 3$, k is selected from range 2 to 10. The experimental result is summarized by figure 2.

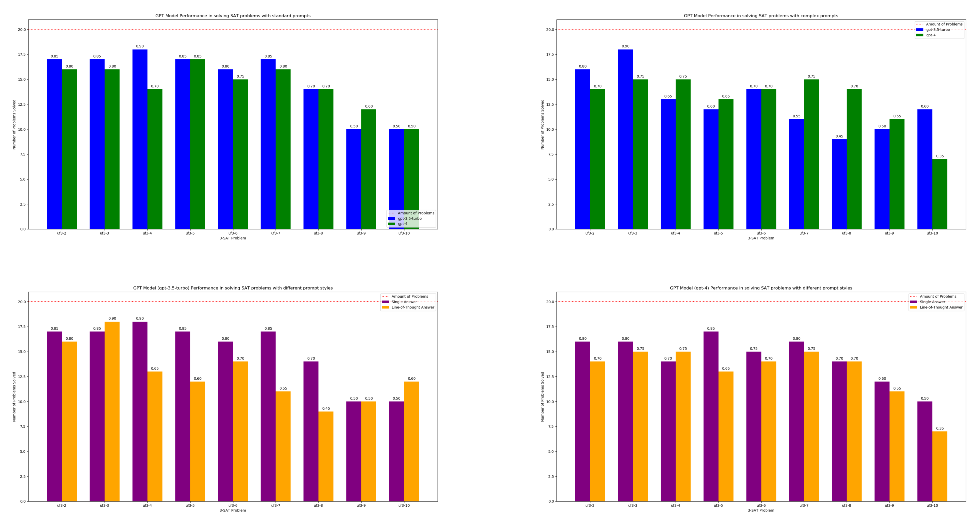

Figure 2: A set of bar charts of GPT models on uf3 SAT problems with standard prompting (upper left) / complex prompting (upper right) and different prompting style on uf3 SAT problems with gpt-3.5-turbo (lower left) / gpt-4 (lower right). The value above each bar indicates the solve rate of each setting. You can find figures of higher resolution in Appendix. (Figure 8, 9, 10, 11)

In general, when $n = 3$, both model maintained a relatively high solve rate for simple cases. There is a weak trend of decreased solve rate as k goes up.

Within GPT-3.5-Turbo, the two general prompting styles do not reveal a superior one over the other. However, within GPT-4, forcing the model to provide a single answer seems to be more favorable than line-of-thought approaches.

In comparison, although GPT-4 showed a better performance with line-of-thoughts prompt, GPT-3.5-Turbo seems to be the better performing model over all for this set of problems.

4.2.2 uf5

For $n = 5$, k is selected from range 12 to 60 with a gap of two for a larger coverage of clause distributions. The experimental result is summarized by figure 3.

When $n = 5$, the overall solve rate dropped significantly, barely going over 50%. An generally decreasing trend can again be observed as k goes up. However, with a wider span of k, there seems to be a rift at around $k = 44$ to $k = 48$. This suggests a potential optimal k in the sweet spot of conditions and constraints enforced by clauses that raises difficulty to maximum.

This time the relationship between the two prompting styles seems to be much more tangled and polarized. There was no general trend of one superior than the other. Between different k values, the solve rate can also be significantly different.

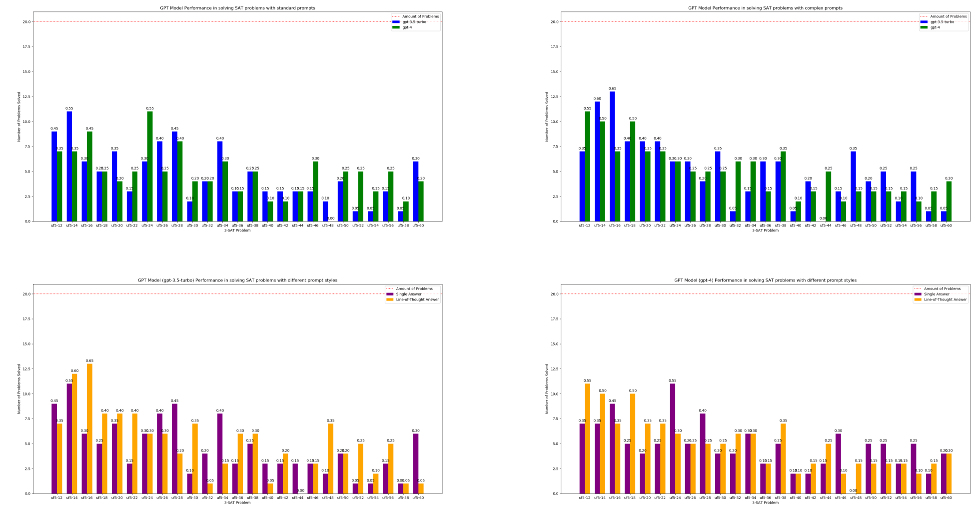

Figure 3: A set of bar charts of GPT models on uf5 SAT problems with standard prompting (upper left) / complex prompting (upper right) and different prompting style on uf5 SAT problems with gpt-3.5-turbo (lower left) / gpt-4 (lower right). The value above each bar indicates the solve rate of each setting. You can find figures of higher resolution in Appendix. (Figure 12, 13, 14, 15)

Similar relationship can be observed between GPT-3.5-Turbo and GPT-4. However, the between-model difference seems to be much smaller than betweem-prompt-style difference. This is potentially a sign of unified performances across models as the problem complexity increase.

4.2.3 uf10

For $n = 10$, k is selected from range 40 to 100 with a gap of five for a larger but sparser coverage of clause distributions. The experimental result is summarized by figure 4.

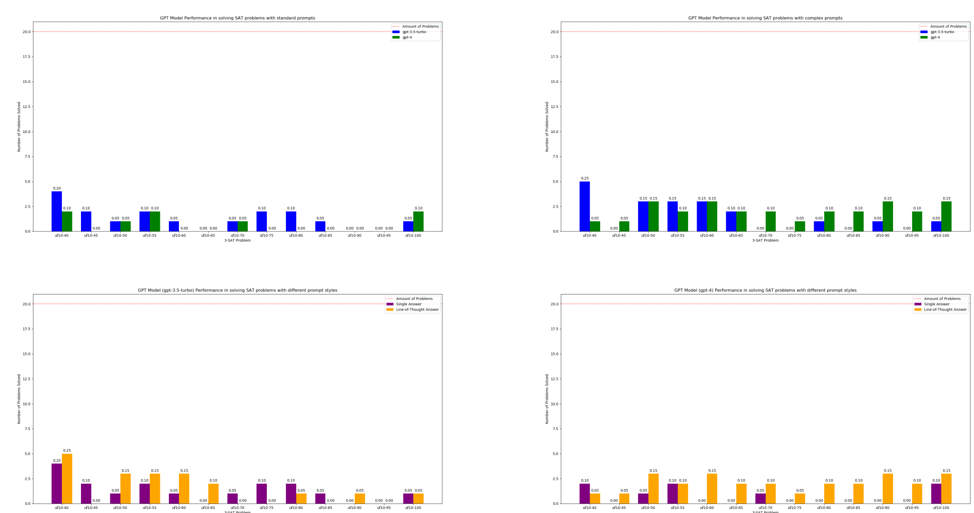

Figure 4: A set of bar charts of GPT models on uf10 SAT problems with standard prompting (upper left) / complex prompting (upper right) and different prompting style on uf10 SAT problems with gpt-3.5-turbo (lower left) / gpt-4 (lower right). The value above each bar indicates the solve rate of each setting. You can find figures of higher resolution in Appendix. (Figure 16, 17, 18, 19)

The solve rate approaches zero as we increase the amount of variable to $n = 10$. There are still about 5% to 10% of instances (one or two cases) getting solved, which might have been a little better than random guess.

This time we can see a clear boost in using line-of-thought prompts, which helped mitigating zero solve rate for GPT-4. However, this boost seems to be less effective for GPT-3.5-Turbo. With this difference, GPT-4 models with line-of-thought promptings seems be the better setting going into larger n.

4.3 Additional Quantitative Assessment

From previous set of experiments on uf3, uf5 and uf10, the results showed a general decreasing trend in solve rate as the complexity of SAT problem gets harder. However, the results seemed rather chaotic for making a choice of model / prompting style. After assessing the prompts and response manually, we came up with a hypothesis of why line-of-thoughts prompting, having more tokens and followed the chain-of-thoughts observation in earlier work, did not demonstrate a better performance. **That is, line-of-thoughts prompting could potentially lead the model into inconclusive state or allow the model to turn down the query user made.**

To support this hypothesis, We evaluated the recall of all the response collected. That is, whether the model actually provides a valid assignment or not (regardless of whether it is a correct assignment). For demonstration purpose, figure 5 is the recall for GPT-3.5-Turbo v.s. GPT-4 on uf10 as an example:

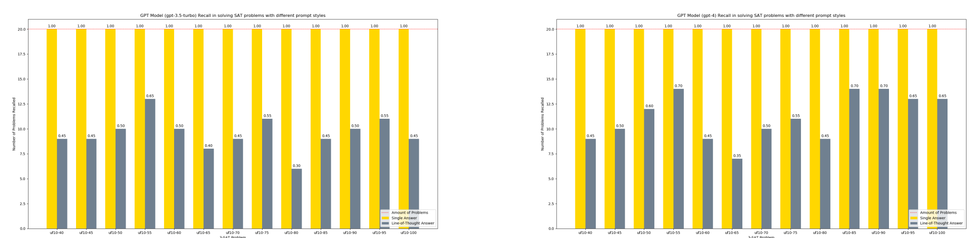

Figure 5: A sample pair of bar charts of recalls of different prompting style on uf10 SAT problems with gpt-3.5-turbo (left) / gpt-4 (right). The value above each bar indicates the recall of each setting. You can find figures for all SAT problems sets of higher resolution in Appendix. (Figure 20, 21, 22, 23, 24, 25)

As indicated by the graph, prompts that demands only outputting a single answer actually have perfect recall as the model will attempt to deliver an assignment regardless of their precision. In contrast, the recall for line-of-thoughts prompts are only around 50%, meaning that around half of the response did not even give a valid answer for evaluation to begin with.

This is also the case for uf3 and uf5, although the former has a slightly higher recall for line-of-thoughts prompts. This observation explains why it was hard to draw conclusions from results of different prompting styles.

4.4 Qualitative Analysis

To ensure the recall does not come from parsing errors, a large portion of the responses were manually assessed in addition to the components in the pipeline. The unrecalled cases can be summarized into two different categories: (1) An explanation of how to approach the problem (some with codes as well) is given, but no answers are calculated; (2) A plain rejection of the request, stating the complexity of the problem and explaining why it was not the right task for the model.

The former is mostly observed in uf3 and uf5, in which the model seems to “eventually give up” on producing a final solution. An example of that can be found in figure 6.

The latter is mostly observed in uf10 and uf5 with larger k, in which the model recognizes the complexity of the problem and states that it is better solved with SAT solver. An example of this case can be found in figure 7.

```
To solve this SAT problem encoded in DIMACS format, we need to find a satisfying assignment of Boolean values (true/false) for the variables that satisfies all the clauses.

First, let's analyze the DIMACS format:
- There are 5 variables (1, 2, 3, 4, 5).
- There are 40 clauses.
- Each line represents a clause, with a 0 at the end.

Now, let's convert the clauses to a more readable format:
1. (2 OR NOT 1 OR NOT 3)
2. (NOT 4 OR NOT 1 OR 2)
3. (1 OR NOT 3 OR NOT 4)
4. (NOT 2 OR NOT 3 OR NOT 5)
5. (2 OR NOT 5 OR NOT 4)
...
And so on for the remaining clauses.

Now, we need to find a satisfying assignment that makes all these clauses true. We can use a SAT solver to do this efficiently.

Encoding the problem into a SAT solver will involve setting up the constraints based on the clauses given. The SAT solver will then search for a valid assignment that satisfies all
the constraints.

answer: SATISFIABLE
```

Figure 6: An example response from uf5 that provides line-of-thoughts, but reaches no valuable conclusion.

```
The task requires us to solve a SAT problem. The instances of SAT problems given are encoded in DIMACS format. This is a popular format for representing SAT problem instances.

To solve a SAT problem we could use a SAT solver (like MiniSAT, or Cryptominisat), which uses the DPLL algorithm or its improvements (like clause learning). Here, as on SAT problem
has been given, we are assuming the use of a hypothetical or actual SAT solver to solve the problem.

The process that a typical SAT solver follows is:

1. Convert all problems into Conjunctive Normal Form (CNF). But here, the problem is already given in CNF format.

2. Here, the clauses to be satisfied are all the lines except for the first one, which simply tells us there are 10 variables and 85 clauses. A literal is a variable or its
negation. A clause is a disjunction of literals. A problem is in conjunctive normal form if the conjunction of the clauses is what is to be satisfied.

3. The SAT solver makes an assignment, and then checks if it leads to any clause being false. If there's any such clause, it's called a conflict clause.

4. The solver then iteratively revises its assignments to resolve these conflicts. It uses the principle of resolution which says that given any two clauses, one containing a
certain literal and another containing its negation, we can infer a clause that contains the rest of the literals from both clauses.

5. The solver keeps track of these resolutions in a data structure called the 'implication graph.'

6. When a conflict occurs, it uses the implication graph to identify which assignments to change. The assignments that led directly to the conflict form a 'cut' in the graph, some
of those assignments must be changed. This cut is a 'conflict clause.'

7. As the solver makes and revises assignments, it learns new conflict clauses, which help it prune the search tree, it avoids making the same mistakes again. This process is
called 'clause learning.'

8. The solver employs a decision heuristic to determine which variable should be assigned when there is no forced assignment. One common such heuristic is the VSIDS heuristic,
where it keeps a score for each variable and chooses the variable with the highest score. This score is initially the same for all variables and goes up whenever an assignment to a
variable leads to a conflict.

9. It also uses methods to prevent the clause database from growing too large, by periodically deleting certain less useful learnt clauses.

10. The solver continues this process until all clauses are satisfied (in which case the problem is SAT), or until it finds that no assignment satisfies all clauses (in which case
the problem is UNSAT).

The SAT solver would now apply these principles and strategies to produce the solution.

Since we don't have the computational power here to solve it directly, we cannot provide the exact solution to the SAT problem here.

answer: To solve the SAT problem provided, you need to use a SAT solver capable of handling this level of complexity, such as MiniSAT or CryptoMiniSat. Manual solution for problems
of this size is impractical if not impossible.
```

Figure 7: An example response from uf10 that basically refuses to provide an assignment on its end.

It is notable that these two behaviors, although similar in response, but are quite different fundamentally. This is also different from behaviors through chatGPT's interface as that would likely produce answers from running the synthesized code. This is one of the main reason why the experiment is conducted through the API rather than publicly accessible chat interfaces.

5 Limitations and Future Work

5.1 Limitations

5.1.1 Scalability of investigation

There is one particular issue about the way how response collector were constructed to parse response: It is difficult to carry it to larger scale. By inspecting the responses collected, it is surprising that the model is quite creative in how the answer is delivered. It has more than 5 ways of disclosing an arbitrary assignment in different format. Many of those are realized and covered as the response collector is coming together, but this is apparently an unreliable design factor when trying to apply the same pipeline to problems of slightly different format.

5.2 Future Work

There are many planned experiments not covered in this report due to time and resource constraints. Many of the planned experiment will be disclosed in this section.

5.2.1 More in-depth quantification of task difficulties

In the current stage of the project, The difficulty of a SAT problem instance is defined by the amount of variables (n), the amount of clauses (k) and the length of the clauses (m). However, the difficulty are often not necessarily uniform across different SAT instances. There is a potential in describing the semantic difficulty of a SAT instance by, for example, number of conflicts that are needed by a modern CDCL-based solver. That way, less complex but "harder" problems can be used in the experiment.

5.2.2 More challenging SAT Generation

In this project, SAT instances are currently generated by a very simple scheme that ensures the generated instance is satisfiable. However, there are many works such as G2SAT [9] that aims to generate more "legitimate" SAT instances. This is also something worth looking at when trying to boost the difficulty of the question in contrast to simply adding more variables and clauses.

5.2.3 More variations in prompt engineering

In this project, only two general types of prompts are covered. However, there are many other ways of creating more informative and more complex prompt, such as Chain-of-Thought Prompting [6], OOD prompting (e.g. provide uf5 examples prior to doing uf10), encoding the SAT problem with natural language (e.g. give each variable a semantically meaningful object and turn the problem into a word problem).

5.2.4 More conventional LLMs

Due to time and resource limit, only two models from the GPT family is being assessed. However, there are many competitive LLMs such as LLaMa, PaLM, Claude etc. Those models are also more "conventional" decoder-based model, for which prompting techniques like Chain-of-Thought are already proven to be quite effective. Using the same pipeline on those model will significantly improve the generalizability of all the conclusions.

5.2.5 More experiment on few-shot promptings

One important aspects of LLMs that I regret to exclude from the scope of the project is to examine the effect of doing few-shot prompting and collect multiple response from the model. Currently the experiment is done in zero shot, meaning that each query is handled with a fresh session without context. Moreover, only the first response is collected without ever having it regenerate. This decision significantly reduced the complexity of the experiments, but also left out many potential in drawing meaningful conclusions.

6 Conclusion

The report proposes a pipeline for verifying reasoning ability of LLMs through SAT problems. This report also presented a few early findings of applying the pipeline to two popular LLMs, GPT-3.5-Turbo and GPT-4. The experiments found that examined LLMs will collapse on solving SAT problems of 10 to 20 variables. The popular line-of-thought prompting could potentially hinder the performance of SAT solving as it no longer guarantee to produce a valid assignment. Although the experiments were significantly constrained by time and resources, a guideline of future work is outlined and will be investigated at a future time.

References

[1] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[2] Zheng Yuan, Hongyi Yuan, Chuanqi Tan, Wei Wang, and Songfang Huang. How well do large language models perform in arithmetic tasks?, 2023.

[3] Jing Qian, Hong Wang, Zekun Li, Shiyang Li, and Xifeng Yan. Limitations of language models in arithmetic and symbolic induction, 2022.

[4] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality, 2023.

[5] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models, 2022.

[6] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903, 2022.

[7] Shima Imani, Liang Du, and Harsh Shrivastava. Mathprompter: Mathematical reasoning using large language models, 2023.

[8] Holger Hoos and Thomas Stützle. *SATLIB: An online resource for research on SAT*, pages 283–292. 04 2000.

[9] Jiaxuan You, Haoze Wu, Clark W. Barrett, Raghuram Ramanujan, and Jure Leskovec. G2SAT: learning to generate SAT formulas. *CoRR*, abs/1910.13445, 2019.

A Appendix

A.1 Relevant Codes for The Project

The code used in this project can mostly be found in `https://github.com/Ken-Shi/SAT4LLM`.

A.2 High Resolution Figures of Different Experiments

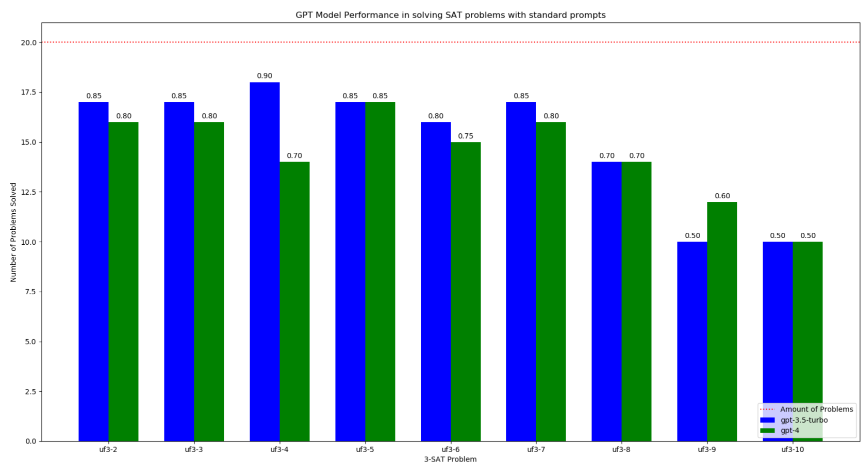

Figure 8: A bar chart of GPT models on uf3 SAT problems with standard prompting. The blue bars represent gpt-3.5-turbo and the green bars represent gpt-4. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the solve rate of each setting.

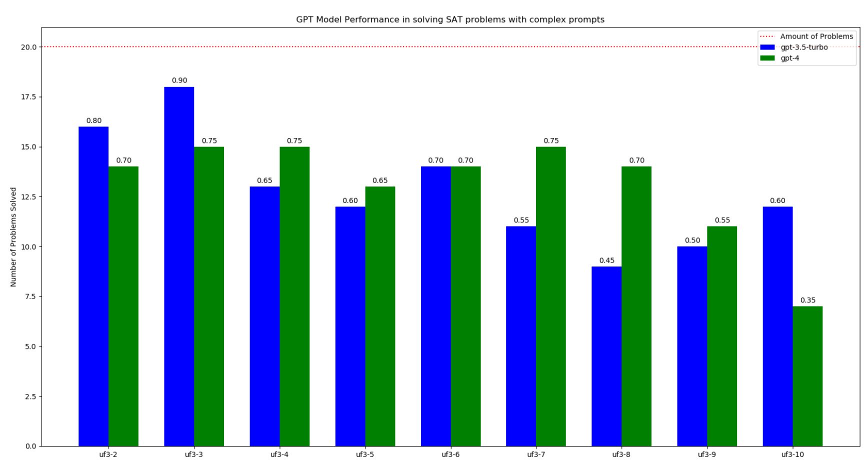

Figure 9: A bar chart of GPT models on uf3 SAT problems with complex prompting. The blue bars represent gpt-3.5-turbo and the green bars represent gpt-4. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the solve rate of each setting.

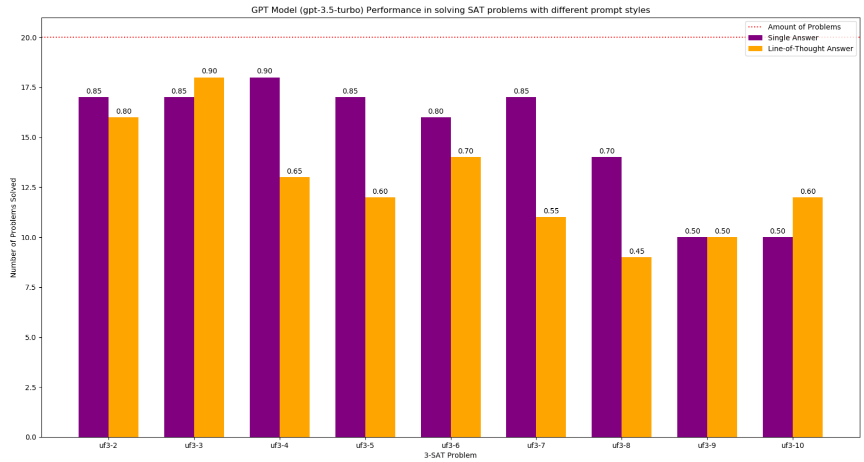

Figure 10: A bar chart of different prompting styles on uf3 SAT problems with gpt-3.5-turbo. The purple bars represent single answer prompts and the orange bars represent line-of-thoughts prompts. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the solve rate of each setting.

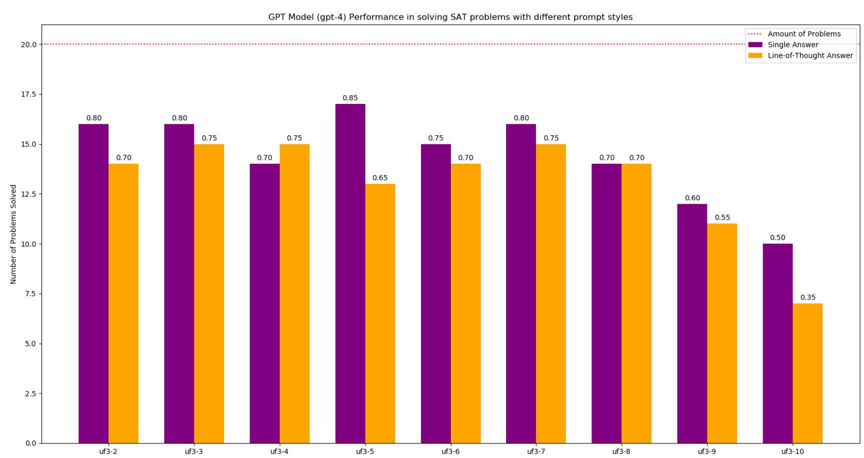

Figure 11: A bar chart of different prompting styles on uf3 SAT problems with gpt-4. The purple bars represent single answer prompts and the orange bars represent line-of-thoughts prompts. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the solve rate of each setting.

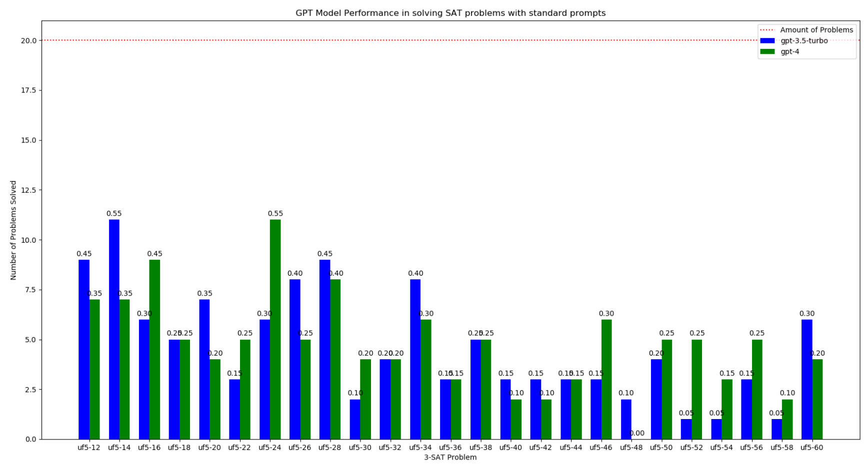

Figure 12: A bar chart of GPT models on uf5 SAT problems with standard prompting. The blue bars represent gpt-3.5-turbo and the green bars represent gpt-4. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the solve rate of each setting.

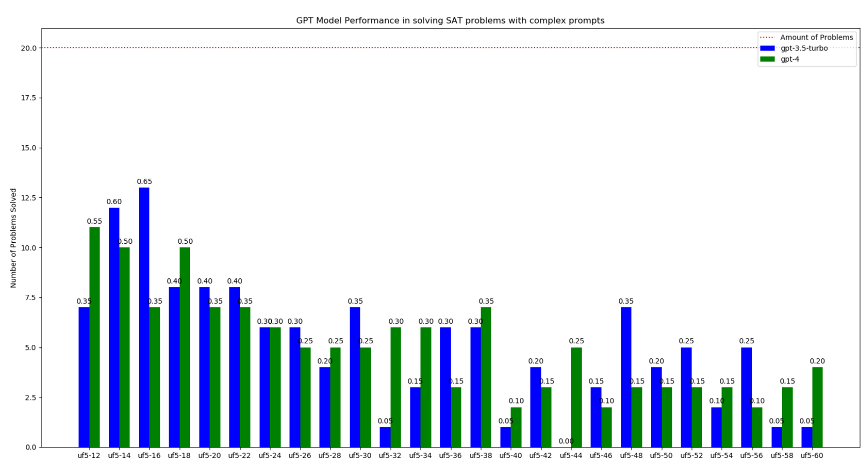

Figure 13: A bar chart of GPT models on uf5 SAT problems with complex prompting. The blue bars represent gpt-3.5-turbo and the green bars represent gpt-4. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the solve rate of each setting.

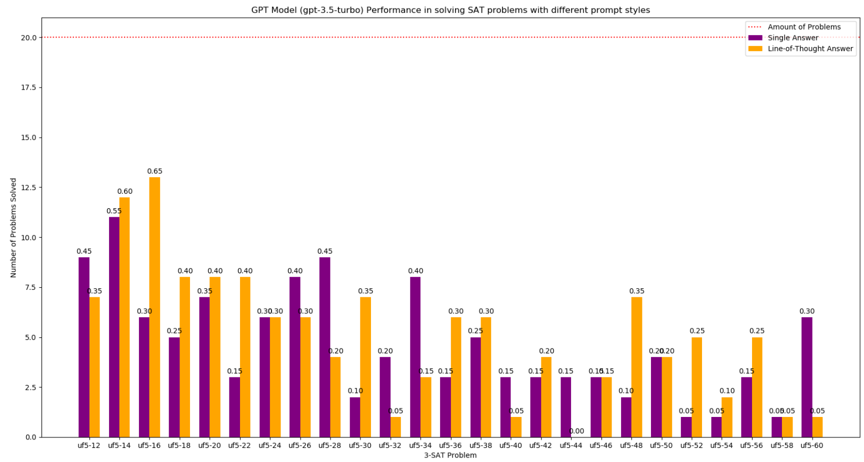

Figure 14: A bar chart of different prompting styles on uf5 SAT problems with gpt-3.5-turbo. The purple bars represent single answer prompts and the orange bars represent line-of-thoughts prompts. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the solve rate of each setting.

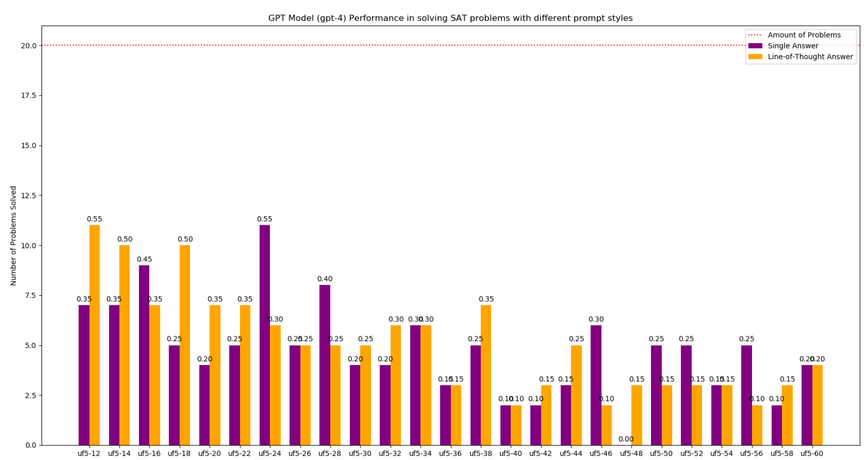

Figure 15: A bar chart of different prompting styles on uf5 SAT problems with gpt-4. The purple bars represent single answer prompts and the orange bars represent line-of-thoughts prompts. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the solve rate of each setting.

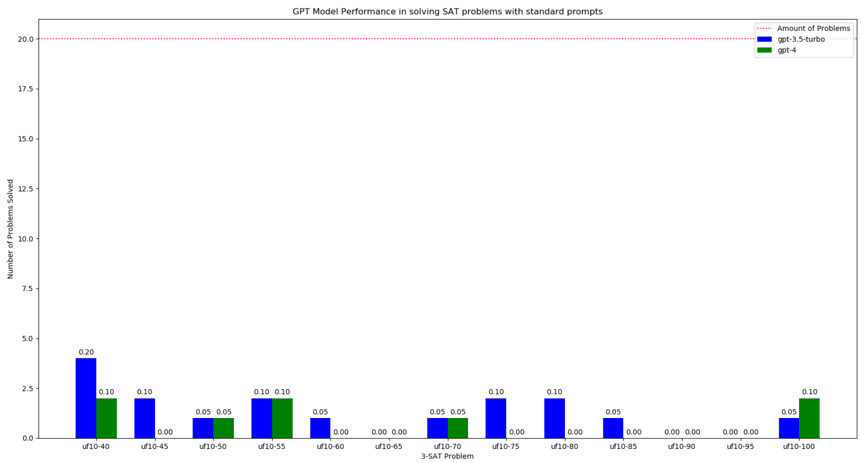

Figure 16: A bar chart of GPT models on uf10 SAT problems with standard prompting. The blue bars represent gpt-3.5-turbo and the green bars represent gpt-4. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the solve rate of each setting.

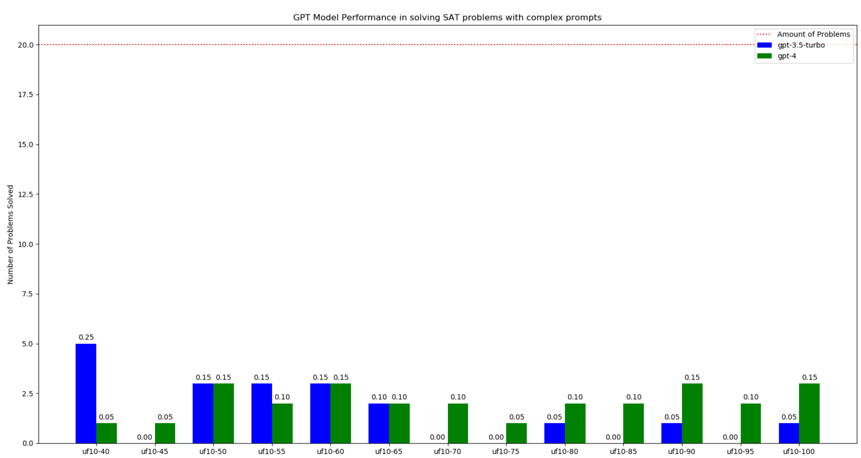

Figure 17: A bar chart of GPT models on uf10 SAT problems with complex prompting. The blue bars represent gpt-3.5-turbo and the green bars represent gpt-4. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the solve rate of each setting.

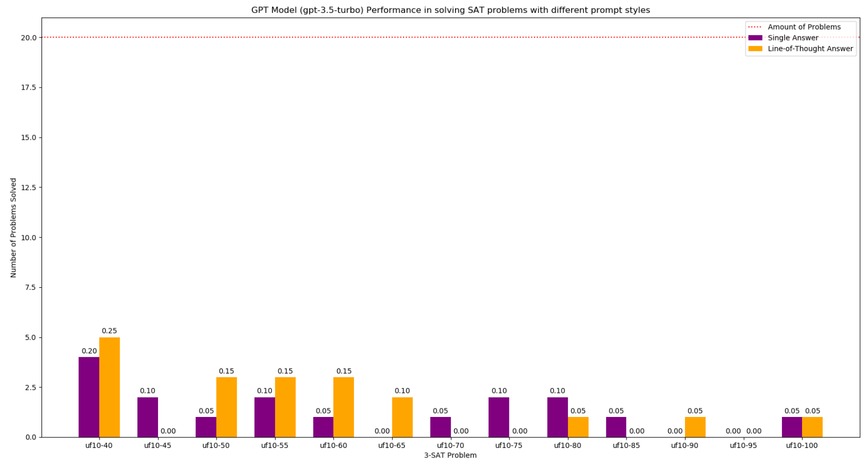

Figure 18: A bar chart of different prompting styles on uf10 SAT problems with gpt-3.5-turbo. The purple bars represent single answer prompts and the orange bars represent line-of-thoughts prompts. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the solve rate of each setting.

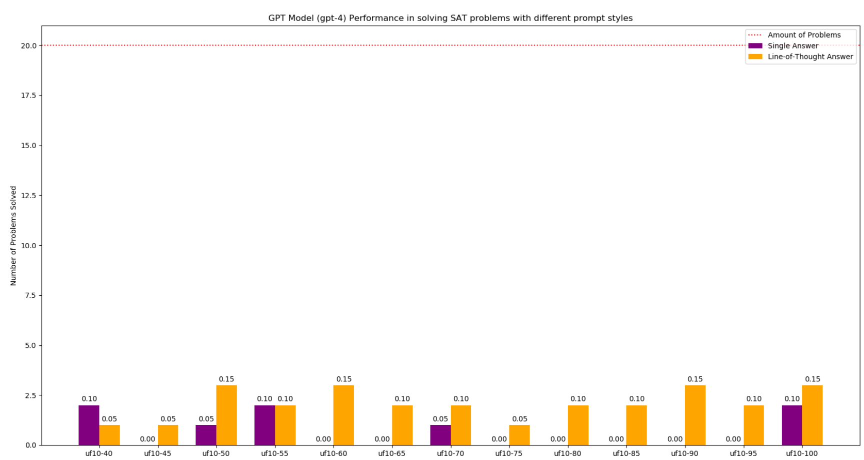

Figure 19: A bar chart of different prompting styles on uf10 SAT problems with gpt-4. The purple bars represent single answer prompts and the orange bars represent line-of-thoughts prompts. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the solve rate of each setting.

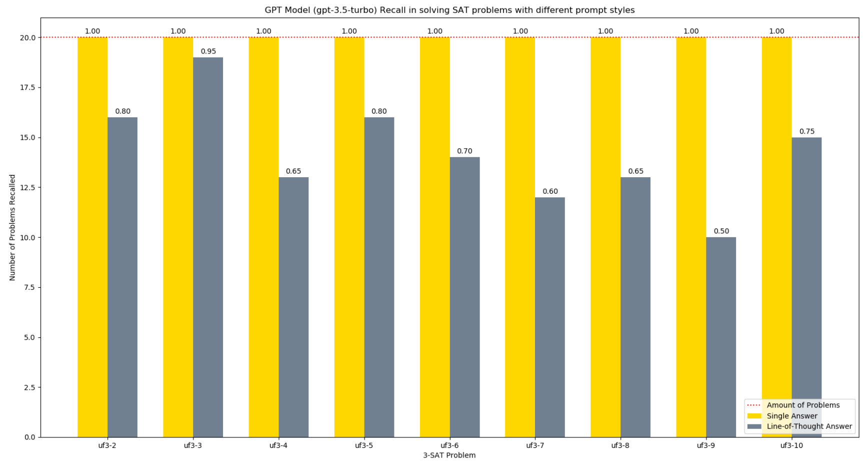

Figure 20: A bar chart for recalls of different prompting styles on uf3 SAT problems with gpt-3.5-turbo. The yellow bars represent single answer prompts and the grey bars represent line-of-thoughts prompts. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the recall rate of each setting.

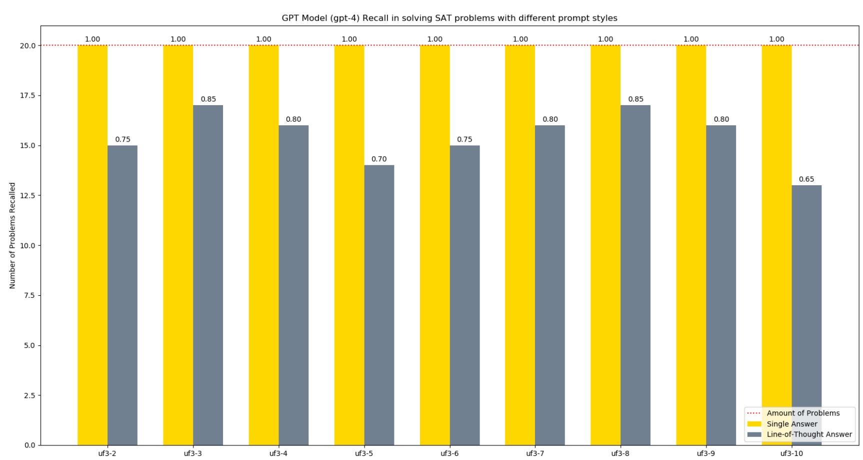

Figure 21: A bar chart for recalls of different prompting styles on uf3 SAT problems with gpt-4. The yellow bars represent single answer prompts and the grey bars represent line-of-thoughts prompts. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the recall rate of each setting.

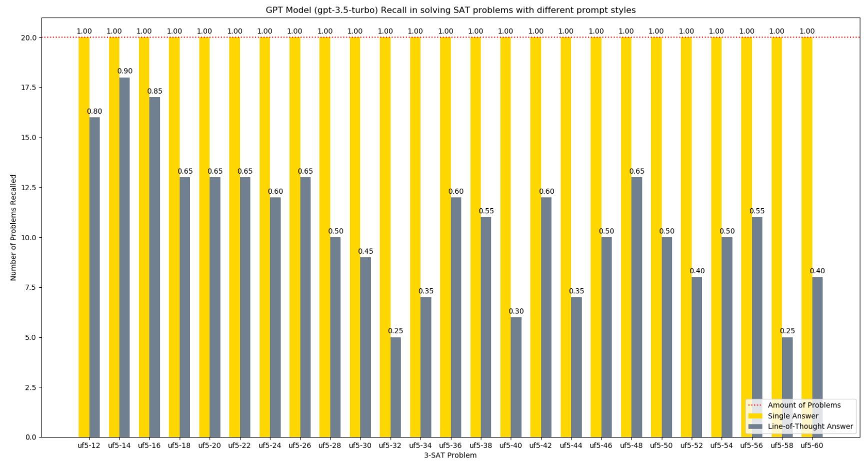

Figure 22: A bar chart for recalls of different prompting styles on uf5 SAT problems with gpt-3.5-turbo. The yellow bars represent single answer prompts and the grey bars represent line-of-thoughts prompts. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the recall rate of each setting.

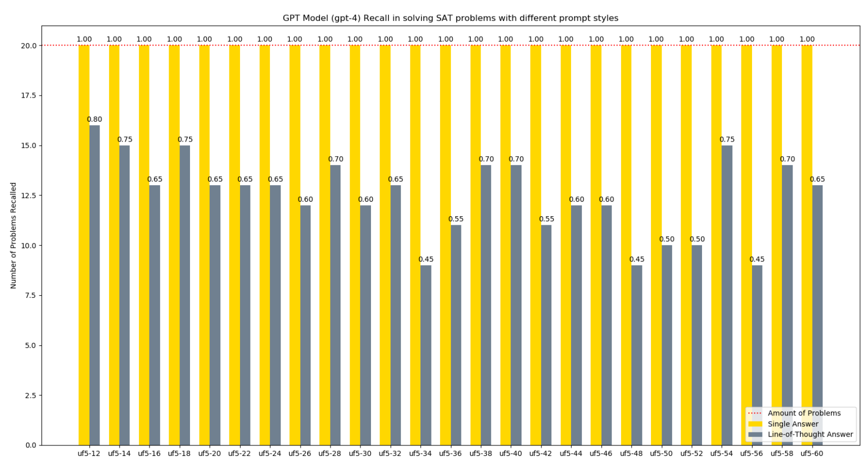

Figure 23: A bar chart for recalls of different prompting styles on uf5 SAT problems with gpt-4. The yellow bars represent single answer prompts and the grey bars represent line-of-thoughts prompts. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the recall rate of each setting.

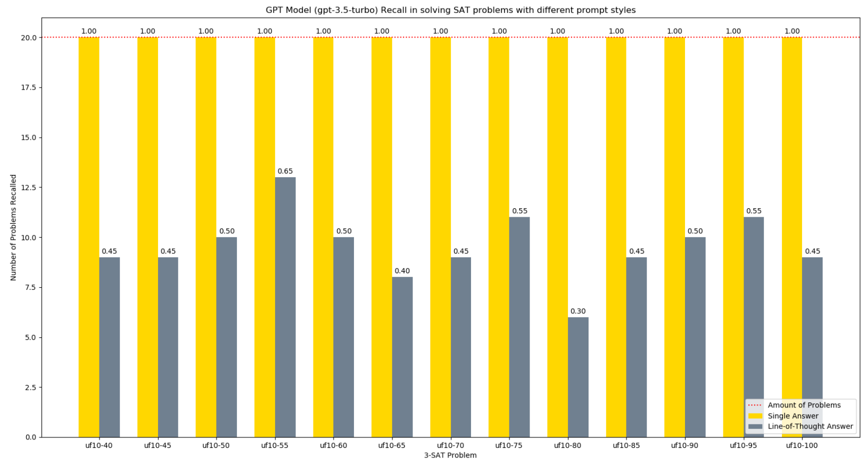

Figure 24: A bar chart for recalls of different prompting styles on uf10 SAT problems with gpt-3.5-turbo. The yellow bars represent single answer prompts and the grey bars represent line-of-thoughts prompts. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the recall rate of each setting.

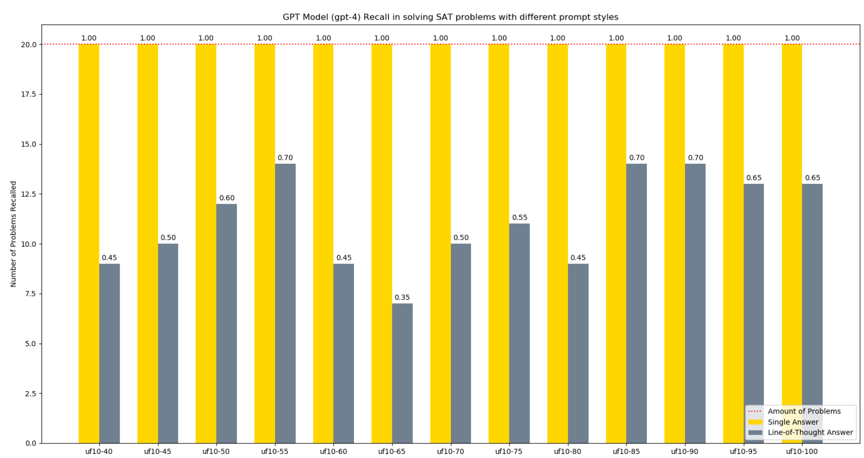

Figure 25: A bar chart for recalls of different prompting styles on uf10 SAT problems with gpt-4. The yellow bars represent single answer prompts and the grey bars represent line-of-thoughts prompts. The red dotted line indicates the total amount of problems tested. The value above each bar indicates the recall rate of each setting.