Large Language Models Exhibit Limited Reasoning Ability on Coding Problems

Anonymous ACL submission

Abstract

Claims that large language models (LLMs) have complex reasoning ability have stirred broad interests, and controversies, of academics and non-academics alike. A popular basis for such claims comes from LLMs' ability to solve coding problems, which involves understanding the problem statement and providing code that solves the problem. Although such abilities are remarkable feats worth praising, we argue that they come from memorization rather than reasoning. We first show that LLMs' problem-solving ability degrades with increased recency of the problem, likely due to the reduced amount of training data for more recent problems, regardless of the problem difficulty labeled by human experts. Additionally, we show that an LLM often fails to solve the problem when presented with reworded but equivalent problem statements, further suggesting their limited reasoning ability.

1 Introduction

The development of large language models (LLMs) has substantially advanced natural language processing, enabling various applications across multiple domains. Their remarkable performance on diverse tasks has sparked significant research interest in their reasoning abilities. While LLMs are often claimed to possess reasoning abilities, it still remains a controversy (Huang and Chang, 2023; Mondorf and Plank, 2024). Although some previous studies provide evidence that LLMs are able to reason (Shi et al., 2022; Suzgun et al., 2023; Lampinen et al., 2024; Saparov and He, 2023), others show that their ability to reason is limited (Arkoudas, 2023; Yang et al., 2023; Kambhampati et al., 2024; McCoy et al., 2023; Valmeekam et al., 2023; Razeghi et al., 2022). Some studies show evidence for both the true ability to reason and reliance on shallow heuristics (Prabhakar et al., 2024). Thus, it remains unclear whether the behavior of these systems is based on true reasoning or superficial heuristics such as memorization or recognition of surface-level patterns.

In this study, we evaluate LLMs' reasoning ability on solving coding problems, which inherently require reasoning skills. Code generation, where models assist with generating and completing code, is one of the most popular application of LLMs (Jiang et al., 2024), with the promises of AI-generated code improving coding efficiency. While previous research has shown that LLMs can generate code, often more efficiently than humans (Coignion et al., 2024), it has rarely been investigated whether this ability stems from genuine reasoning or merely from retrieving patterns from training data. Coding problems provide an effective test case for this distinction, as they require reasoning skills, yet many widely used problem sets are likely to have been included in training data. This allows us to assess whether their performance reflects true reasoning or pattern recognition. If LLMs exhibit lower accuracy on prompts that are unlikely to have appeared in their training data, while performing well on otherwise similar problems, this would suggest that their performance is primarily dependent on memory retrieval rather than reasoning.

To test this, we conducted two analyses. First, we investigated the change in model performance as a function of the time point at which the problem was entered into the coding problem database. If models perform better with older problems, which are more likely included in the training data, compared to more recent problems with similar difficulty, the finding would suggest that the model tend to recite from memory rather than use reasoning skills. Second, we investigated the effect of prompt perturbation, by replacing 1-3 keywords with their synonyms and testing whether the model performance changes. If model performance on code generation is susceptible to word-level perturbation in the prompts, as often reported in previous studies

(Wang et al., 2023; Qiang et al., 2024; Zhu et al., 2024; Zhuo et al., 2023), the finding would suggest that models perform well only on word sequences they have encountered during training, rather than applying reasoning skills to solve problems. We also examine the effect of word frequency on performance under perturbation. Our results from both analyses demonstrate that models respond differently to questions that are essentially the same but phrased differently, suggesting that LLMs have limited reasoning ability.

2 Methods

2.1 Dataset Specification

The evaluation data set consists of manually collected random samples of coding problems from a popular repository called LeetCode[1]. Each LeetCode problem has a set of associated attributes: Problem ID, difficulty, test cases, # of submissions, and # of acceptances. Problem IDs are numerical values assigned automatically by LeetCode, indicating the order in which problems were added to the database. A higher ID corresponds to a more recently added problem. The difficulty level is one of *easy, medium, hard* and is a manually labeled identifier that provides an estimate of the difficulty of each problem. Test cases are used to verify the correctness of the provided solution and consist of input-output pairs that the solution must process correctly, producing the expected output for each given input. All test cases must pass in order for the solution to be correct or accepted. # of submissions is the number of attempted solutions submitted by the users, and # of acceptances are the subset of such submissions that pass all the test cases. A sample of the problem and test cases is presented in Appendix A.

The data set collected for the present study consisted of 30 problems for each difficulty level, resulting in a total of 90 problems.

2.2 Model Specifications

The set of models evaluated is (ordered in the decreasing number of model parameters): GPT-4o (OpenAI et al., 2024), Gemini 1.5 Pro (Team et al., 2024), GPT-3.5 (OpenAI et al., 2024), Llama 3 70B (Grattafiori et al., 2024), and StarCoder 2 15B (Li et al., 2023). This selection ensures coverage of key model variables such as the model size, whether the model is open- or closed-source, and whether the model is vanilla or fine-tuned. For example, GPT-4o and Gemini 1.5 Pro are closed-source models, while Llama 3 and StarCoder 2 are open-source. GPT-4o, Gemini 1.5 Pro, and Llama 3 are vanilla models optimized for generic question answering, and StarCoder 2 is a model fine-tuned specifically for code generation.

[1]https://leetcode.com

2.3 Prompt Perturbation Analysis

We examined how modifying certain words within a prompt affects the output quality of large language models. For this, we used only GPT-4o, as it demonstrated the best performance among all models outlined above (see Section 3). We selected a subset of problems (N=25) for which GPT-4o passed all test cases and modified the prompts without altering their meaning. To ensure that the model received essentially the same prompt, we replaced specific keywords with their closest synonyms.

We extracted a set of keywords in each prompt using keyBERT (Grootendorst, 2020), a model that provides a list of keywords, each with a score that measures how important the word is in the prompt. KeyBERT utilizes BERT, a model that generates vectors from text, to extract document embeddings and word embeddings, and provides a set of relevant words based on cosine similarity measures. Using keyBERT, we performed three levels of keyword-based analysis, replacing the highest scoring, the top two, and the top three words with their synonyms, in order to examine changes in model performance as a function of the number of keywords replaced. If a keyword (e.g., *stacking*) appeared in another form (e.g., *stacked*), all instances were replaced accordingly (e.g., *piling*, *piled*), but the total replacement count was still considered as one. That is, we examined the effect of the number of *unique* keywords replaced. An example of perturbations is presented in Appendix B.

Synonyms for the selected keywords were obtained from the WordNet database (Princeton University, 2010) using the nltk package (Bird et al., 2009). From the list of synonyms suggested by WordNet, we selected the one that minimally altered the meaning of the original prompt. In addition, we obtained the word frequencies of the original keywords and their synonyms from SUBTLEXus (Brysbaert and New, 2009) to. If a word does not appear on the SUBTLEXus frequency list, the frequency was regarded as zero.

2.4 Evaluation Metric

Each of the model performance is solely evaluated on whether or not each output generated passes all of the test cases. For a given model m, problem p and query q, correctness of the model and problem pair for the given query is defined as:

$$C_{m,p}(q) = \begin{cases} 1, & m\text{'s solution for } p \text{ generated} \\ & \text{from } q \text{ passes all test cases} \\ 0, & \text{Otherwise.} \end{cases}$$

Given $C_{m,p}(q)$, the accuracy A is defined as

$$A_m(q) = \frac{\sum_{p \in P} C_{m,p}(q)}{|P|}$$

where accuracy for a given model m for a set of problem P varies for input query q.

3 Results & Discussion

3.1 Recency effect on accuracy

We first evaluated how the problem solving performance of the five LLMs outlined in 2.2 changes as a function of the time point at which a problem was entered into LeetCode. Figure 1 presents correctness of the five models' outputs as a function of Problem ID, divided by difficulty. First, the accuracy of output was highest among the Easy problems (0.82), intermediate among the Medium (0.59), and lowest among the Hard ones (0.33). Of the five models, GPT-4o exhibited the highest accuracy (0.78). Within each difficulty, we see an effect of problem ID such that the model tends to struggle more often with higher-numbered (more recent) problems. This pattern is observed consistently across all models. The result shows that the models perform better with older problems, which are more likely to have been included in the training data, suggesting that the problem solving ability is likely based on memorization rather than true reasoning.

3.2 Prompt Perturbation

Next, we evaluated changes in model performance under prompt perturbation. As mentioned in Section 2.3, we only considered GPT-4o's outputs for this analysis, as GPT-4o exhibited the highest overall problem solving performance in 3.1.

Figure 1 presents the correctness of GPT-4o outputs as a result of prompt perturbation. By design, accuracy in the Original Prompt condition is 1, as we included only the prompts that the model correctly solved in the analysis. Critically, replacing a single keyword with a synonym already led to a drop in accuracy (0.72), which declined further as more keywords were replaced (0.64 when two words were replaced; 0.56 when three were replaced). Despite the small sample size, this pattern was consistent across all difficulty levels—accuracy never increased but either remained the same or decreased when more words were replaced. Specifically, for Easy problems, the number of correct solutions changed from 7 to 6 to 6 across the 1-, 2-, and 3-word replacements; for Medium problems, from 6 to 5 to 5; and for Hard problems, from 5 to 5 to 3.

In the 1-word replacement condition, we also examined how the relative frequency between the original word and its synonym affected model performance. Among the 25 problems analyzed, one had identical frequencies for both words and was excluded from the analysis. For the remaining 24 problems, the new word (synonym) had a higher frequency in 15 cases and a lower frequency in 9 cases, compared to the original word. When the new word was more frequent (N=15), the model correctly solved 11 problems and failed on 4 (0.73 accuracy). When the new word had a lower frequency (N=9), the model correctly solved 6 problems and failed on 3 (0.67 accuracy). Therefore, a decrease in word frequency in the new prompt leads to a greater drop in accuracy compared to an increase in frequency, although this finding should be interpreted with caution due to the small sample size.

3.3 Summary & Discussion

In summary, LLMs perform better on older LeetCode problems, likely due to memorization from training data, and exhibit decreasing accuracy on more recent problems and when a small portion of the prompts are perturbed, suggesting a reliance on surface-level patterns rather than true reasoning. We also found that the accuracy decrease is more noticeable when the new prompt contains rarer words. Our findings are consistent with those of previous studies (Gonen et al., 2023; Razeghi et al., 2022) showing that the more frequently the prompt appears in the training data, the more familiar the model is with it, leading to improved model performance. Similarly, numerous studies have reported a close relationship between pretrain-

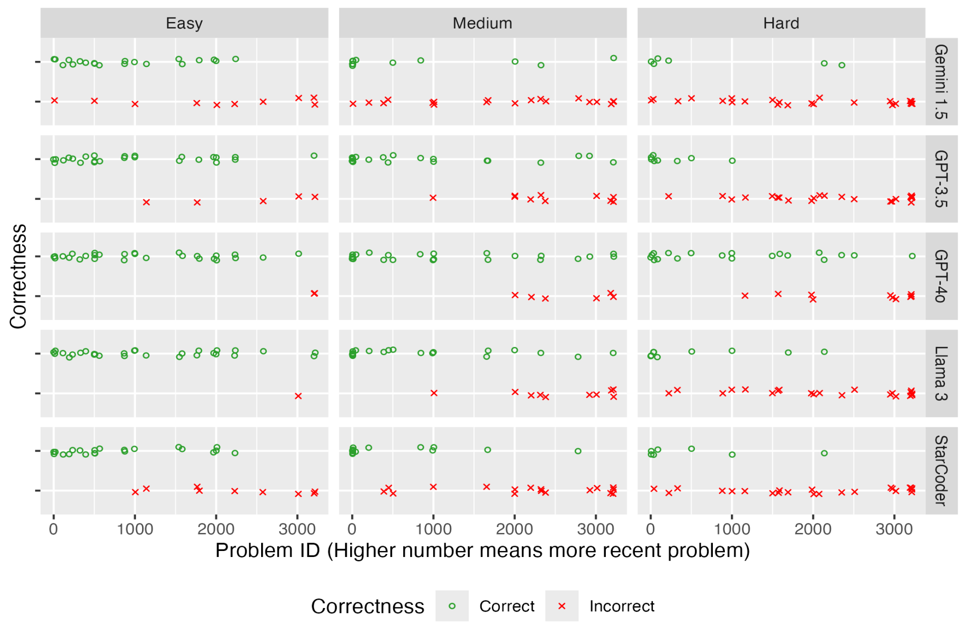

Figure 1: Correctness of model output as a function of Problem ID, grouped by difficulty level and LLM type

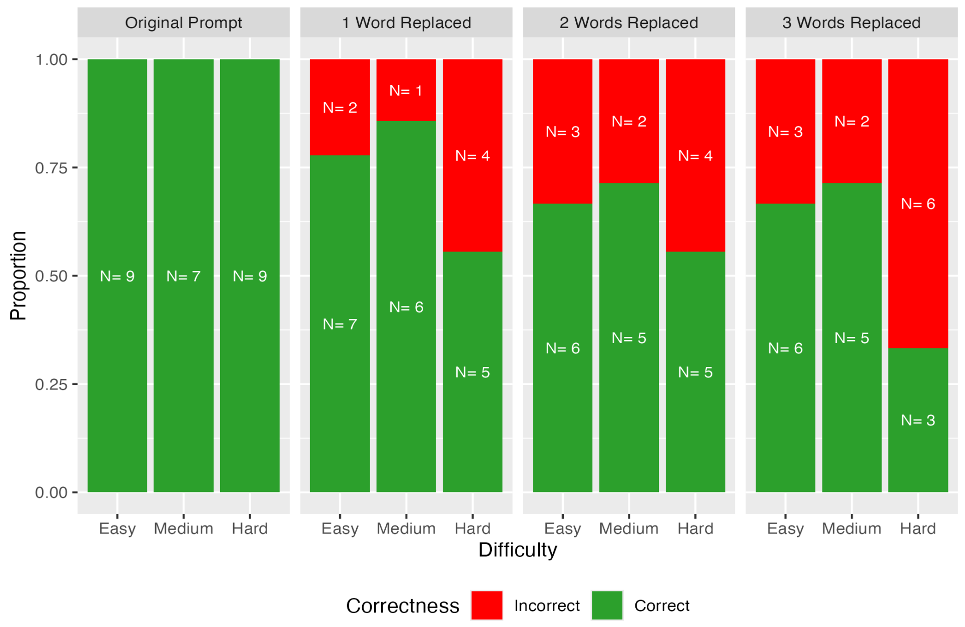

Figure 2: Model accuracy $A_m(q)$ for m = GPT-4o and varying queries q, grouped by the number of unique keywords replaced and difficulty level

ing data and task performance (e.g. Elazar et al. 2023; Kandpal et al. 2023).

4 Conclusion

We argue that LLMs are, by design, autoregressive word sequence predictors and that there is nothing inherent in them that enables reasoning. LLMs can only simulate reasoning by learning statistical patterns in large amounts of data and applying learned patterns in a way that appears logical. This work provides compelling evidence to this argument, which shows that LLMs fail to solve the simplest problems with subtle changes to how the problem is worded. We believe that this work will motivate future research to improve LLMs' problem solving ability via data augmentation using prompt perturbation and improved LLM architectures.

Limitations

One of the main limitations of this work is that the experiment is performed only on English query and result pairs. Although the experimental methods are not limited to English, models trained with other languages may or may not perform better than models trained mostly on English problems. Another limitation to discuss is the limited set of models evaluated. Although this work evaluates the reasoning capabilities of many popular LLMs, it does not provide coverage of many available LLMs, mainly due to the ever increasing number of LLMs coming into the market. In addition, the work does not evaluate specialized reasoning models that employ Chain-of-Thought (CoT) reasoning. We argue that solving LeetCode problems may not be greatly influenced by CoT due to the nature of LeetCode problems asking for a one-step solution, such claim is a worthwhile investigation for future work. The last limitation is the low number of samples evaluated, which is mainly due to the manual nature of data collection and the cost of evaluation. We plan to address this limitation by obtaining and evaluating a larger set of data with additional funding.

Ethics Statement

We do not anticipate an immediate ethical or societal impact resulting from our work. However, we acknowledge that LeetCode is a widely used tool in the industry to assess interview candidates' programming aptitude. Thus, if the content and the result of this work is maliciously used, it can potentially lead to plagiarism on programming interviews. In addition, it is well known that LLMs hallucinate, i.e., generate invalid answers that seems correct. Thus, code generation via the approaches mentioned in this paper can also result in hallucinations, which may lead to unforeseen issues if the code is used directly in real-world applications.

References

Konstantine Arkoudas. 2023. Gpt-4 can't reason.

Steven Bird, Edward Loper, and Ewan Klein. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.

M. Brysbaert and B New. 2009. Moving beyond Kučera and Francis: A critical evaluation of current word frequency norms and the introduction of a new and improved word frequency measure for American English. *Behavior Research Methods*, 41:977–990.

Tristan Coignion, Clément Quinton, and Romain Rouvoy. 2024. A performance study of llm-generated code on leetcode. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, EASE 2024, page 79–89. ACM.

Yanai Elazar, Nora Kassner, Shauli Ravfogel, Amir Feder, Abhilasha Ravichander, Marius Mosbach, Yonatan Belinkov, Hinrich Schütze, and Yoav Goldberg. 2023. Measuring causal effects of data statistics on language model's 'factual' predictions.

Hila Gonen, Srini Iyer, Terra Blevins, Noah Smith, and Luke Zettlemoyer. 2023. Demystifying prompts in language models via perplexity estimation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10136–10148, Singapore. Association for Computational Linguistics.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. 2024. The llama 3 herd of models.

Maarten Grootendorst. 2020. Keybert: Minimal keyword extraction with bert.

Jie Huang and Kevin Chen-Chuan Chang. 2023. Towards reasoning in large language models: A survey.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. 2024. Llms can't plan, but can help planning in llm-modulo frameworks.

Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. 2023. Large language models struggle to learn long-tail knowledge.

Andrew K Lampinen, Ishita Dasgupta, Stephanie C Y Chan, Hannah R Sheahan, Antonia Creswell, Dharshan Kumaran, James L McClelland, and Felix Hill. 2024. Language models, like humans, show content effects on reasoning tasks. *PNAS Nexus*, 3(7):pgae233.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, et al. 2023. Starcoder: may the source be with you!

R. Thomas McCoy, Shunyu Yao, Dan Friedman, Matthew Hardy, and Thomas L. Griffiths. 2023. Embers of autoregression: Understanding large language models through the problem they are trained to solve.

Philipp Mondorf and Barbara Plank. 2024. Beyond accuracy: Evaluating the reasoning behavior of large language models - a survey. In *First Conference on Language Modeling*.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, et al. 2024. Gpt-4 technical report.

Akshara Prabhakar, Thomas L. Griffiths, and R. Thomas McCoy. 2024. Deciphering the factors influencing the efficacy of chain-of-thought: Probability, memorization, and noisy reasoning.

Princeton University. 2010. About WordNet.

Yao Qiang, Subhrangshu Nandi, Ninareh Mehrabi, Greg Ver Steeg, Anoop Kumar, Anna Rumshisky, and Aram Galstyan. 2024. Prompt perturbation consistency learning for robust language models. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 1357–1370, St. Julian's, Malta. Association for Computational Linguistics.

Yasaman Razeghi, Raja Sekhar Reddy Mekala, Robert L Logan Iv, Matt Gardner, and Sameer Singh. 2022. Snoopy: An online interface for exploring the effect of pretraining term frequencies on few-shot LM performance. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 389–395, Abu Dhabi, UAE. Association for Computational Linguistics.

Abulhair Saparov and He He. 2023. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *The Eleventh International Conference on Learning Representations*.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. 2022. Language models are multilingual chain-of-thought reasoners.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. 2023. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, Toronto, Canada. Association for Computational Linguistics.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Haoyu Wang, Guozheng Ma, Cong Yu, Ning Gui, Linrui Zhang, Zhiqi Huang, Suwei Ma, Yongzhe Chang, Sen Zhang, Li Shen, Xueqian Wang, Peilin Zhao, and Dacheng Tao. 2023. Are large language models really robust to word-level perturbations?

Zhun Yang, Adam Ishay, and Joohyung Lee. 2023. Coupling large language models with logic programming for robust and general reasoning from text. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 5186–5219, Toronto, Canada. Association for Computational Linguistics.

Kaijie Zhu, Jindong Wang, Jiaheng Zhou, Zichen Wang, Hao Chen, Yidong Wang, Linyi Yang, Wei Ye, Yue Zhang, Neil Gong, and Xing Xie. 2024. Promptrobust: Towards evaluating the robustness of large language models on adversarial prompts. In *Proceedings of the 1st ACM Workshop on Large AI Systems and Models with Privacy and Safety Analysis*, LAMPS '24, page 57–68, New York, NY, USA. Association for Computing Machinery.

Terry Yue Zhuo, Zhuang Li, Yujin Huang, Fatemeh Shiri, Weiqing Wang, Gholamreza Haffari, and Yuan-Fang Li. 2023. On robustness of prompt-based semantic parsing with large pre-trained language model: An empirical study on codex. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 1090–1102, Dubrovnik, Croatia. Association for Computational Linguistics.

A Problem Structure

Each LeetCode problem has three parts: problem description, a set of constraints and code snippet to guide the answer generation. Here is a sample problem.

Problem Description:

Special Positions in a Binary Matrix. Given an $m \times n$ binary matrix `mat`, return the number of special positions in `mat`.

A position `(i, j)` is called special if `mat[i][j] == 1` and all other elements in row `i` and column `j` are `0` (rows and columns are 0-indexed).

Constraints:

```
m == mat.length
n == mat[i].length
1 <= m, n <= 100
mat[i][j] is either 0 or 1.
```

Code Snippet:

```
def numSpecial(mat):
    """
    :type mat: List[List[int]]
    :rtype: int
    """
```

Given such problem template, the prompt involves prepending the following statement:

> *Can you write python 2 code to solve this problem using the code snippet below:*

To test whether the generated code is correct, each problem also has a set of test cases with an ID that specify a set of inputs and expected output.

ID	Input	Output
1	`[[[1,0,0],[0,0,1],[1,0,0]]]`	1
2	`[[[1,0,0],[0,1,0],[0,0,1]]]`	3
3	`[[[1,1,0],[0,0,0],[0,0,0]]]`	0

If the generated code is correct, the code should produce the correct output for all inputs in a reasonable amount of time.

B Prompt Perturbation Example

An example of prompt perturbation is as follows. Consider the following original prompt:

> Can you write python 2 code to solve this problem using the code snippet below:
>
> Maximum **Height** by **Stacking Cuboids**
>
> Given `n` **cuboids** where the dimensions of the ith **cuboid** is `cuboids[i]=[width_i,length_i,height_i]` (0-indexed). Choose a subset of **cuboids** and place them on each other.
>
> You can place **cuboid** `i` on **cuboid** `j` if `width_i<=width_j` and `length_i<=length_j` and `height_i<=height_j`. You can rearrange any **cuboid**'s dimensions by rotating it to put it on another **cuboid**.
>
> Return *the maximum **height** of the **stacked cuboids***.
>
> *Constraints:*
>
> ```
> n == cuboids.length
> 1 <= n <= 100
> 1 <= w_i, l_i, h_i <= 100
> ```
>
> *Code Snippet:*
>
> ```
> def maxHeight(cuboids):
>     """
>     :type cuboids: List[List[int]]
>     :rtype: int
>     """
> ```

KeyBert model identified the words 'cuboid, height, stacked' as the keywords, and the synonyms of these words are selected as 'box, tallness, piled'. The selected words are emphasized in the example prompt. These words, along with other forms of the same words, such as 'cuboids' and 'stacking', are replaced with respective synonyms and used as queries to the LLM of interest.