# Preliminary measures of faithfulness in least-to-most prompting

**Mateusz Bagiński** **Jakub Nowak** **Lucie Philippon**[1]

**Jan Brauner, Esben Kran, Fazl Barez**

## Abstract

In our experiment, we scrutinize the role of post-hoc reasoning in the performance of large language models (LLMs), specifically the gpt-3.5-turbo model, when prompted using the least-to-most prompting (L2M) strategy. We examine this by observing whether the model alters its responses after previously solving one to five subproblems in two tasks: the AQuA dataset and the last letter task. Our findings suggest that the model does not engage in post-hoc reasoning, as its responses vary based on the number and nature of subproblems. The results contribute to the ongoing discourse on the efficacy of various prompting strategies in LLMs.

*Keywords: evaluations, prompting, AI safety*

## 1. Introduction

The performance of large language models on many tasks (especially difficult ones) is highly dependent on the prompting strategy. Much research effort is put into finding methods of effective elicitation of their capabilities. One such method is chain-of-thought (CoT; Wei et al., 2022), which consists of few-shot prompting the model with examples of problems or problems similar to the one we want it to solve, followed by exemplary answers that include sophistication chains of reasoning (hence the name of the technique). CoT was shown to improve model performance on many tasks (Wei et al., 2022).

However, it is not clear whether this performance improvement is due to the model relying on appropriate intermediate computations performed in the generated text (akin to a "scratchpad", e.g., Nye et al., 2021), rather than encouraging the model to "respond as an expert would".

Recently, Lanham et al. (2023) attempted to investigate why CoT seems to help with many tasks. In particular, they wanted to see whether Chain-of-Thought's helpfulness may not be related to its content but some other mechanism. They investigated three

[1] Authors listed in alphabetical order by last name.

hypotheses: **(1) post-hoc reasoning** - the model prompted to do Chain-of-Thought is more likely to produce the correct answers regardless of what is included in the CoT (as a "rationalization"); **(2) reservoir computation -** the model uses the additional tokens included in the CoT as additional positions, in which it can perform computations; **(3) encoded reasoning** - the model includes some information in the CoT, that is "encrypted", and not legible to human. Lanham et al. found no evidence for the latter two hypotheses and limited evidence for the post-hoc reasoning hypothesis, to the extent that varied depending on the model and the task.

Some alternatives to CoT have been proposed, and some of them perform better than CoT on some tasks. One of them is Tree of Thought (Yao et al., 2023), which is essentially an LLM-guided search process over a tree of alternative text completions. Another alternative, slightly closer to the "spirit" of the CoT, is least-to-most prompting (L2M; Zhou et al., 2023), which involves two few-shot learning stages. First, the model is shown how to decompose a complex/difficult task or problem into simpler/easier tasks or problems. These are then parsed in order to be used in the next stage, where the model answer them one after another, at each point seeing the current subproblem as well as the previous subproblems and the answers it has already given. At the end, it is given the final problem in the same way. Zhou et al. demonstrated that on many datasets, L2M performs much better than CoT.

We used L2M on two tasks designed to be decomposed into five steps (subproblems). We used a method similar to the one presented by Lanham et al. (2023) to measure the marginal gain in average accuracy achieved with each additional subproblem provided to the model.

## 2. Methods

We used the gpt-3.5-turbo-0613 model available through OpenAI's API (OpenAI, 2023). Our code is available on GitHub[2].

We evaluated the model's performance using least-to-most prompting on two tasks. The first one is the **AQuA dataset** (Ling et al., 2017). The second one is the last letter task used by Zhou et al. (2023) to present the advantages of least-to-most prompting over chain-of-thought.

### *2.1 AQuA dataset*

For this dataset, following the method of least-to-most prompting, we created an example decomposition prompt showing the model how to decompose a problem into subproblems. We ask the model to enclose the steps into <step#></step#> tags for easier extraction from the generated completion. This prompt was added before each of the sampled problems. After each of the subproblems, we added <step n=5> instead of <step n=3> to prompt the model into performing a 5-step decomposition instead.

A company produces 420 units of a particular computer component every month, at a

[2] https://github.com/Aelerinya/apart-eval-hackathon/

production cost to the company of $110 per component, and sells all of the components by the end of each month.
Q: What is the minimum selling price per component that will guarantee that the yearly profit (revenue from sales minus production costs) will be at least $626,400?
Options: A)226, B)230, C)240, D)260, E)280
A: To answer the question "What is the minimum selling price per component that will guarantee that the yearly profit (revenue from sales minus production costs) will be at least $626,400?", we need to know: <steps n=3><step1>How many units does the company produce yearly?</step1>, <step2>What would have to be the revenue per unit for the yearly revenue to be at least $626,400?</step2> and <step3>What would the price have to be to include the 110$ cost and the desired revenue?</step3></steps>

Each time we prompted the model with the sampled problem and 1–5 subproblems it already solved, we prepended the prompt with an example of the above prompt that was correctly solved using decomposition.

A company produces 420 units of a particular computer component every month, at a production cost to the company of $110 per component, and sells all of the components by the end of each month.
Q: How many units does the company produce yearly?
A: 12 * 420 = 5040
Q: What would have to be the revenue per unit for the yearly revenue to be at least $626,400?
A: 626400 / 5040 ~ 124.29
Q: What would the price have to be to include the 110$ cost and the desired revenue?
A: 124.29 + 110 = 234.29
Q: What is the minimum selling price per component that will guarantee that the yearly profit (revenue from sales minus production costs) will be at least $626,400?
Options: >>(A)226, >>(B)230, >>(C)240, >>(D)260, >>(E)280
A: >>(C)240

After prompting the model with the above prompt, the sample problem, and the first subproblem we take the model's answer to the subproblem and append it to the prompt. We then prompt the model with a final question and store its answer. Then we use the previous prompt (without the final question) and append the next subproblem to receive the model's answer to the subproblem and repeat the process for all subproblems reusing the model's answers and using the prompt after each of the five steps to get the intermediate final answer.

We also, separately, prompt the model with the problem, options, and "A: >>(" string to get its "base answer" with no L2M prompting applied and no context before the problem.

### *2.2 Last Letter Task*

*We take the last-letter-concatenation task (Wei et al., 2022). In this task, each input is a list of words, and the corresponding output is the concatenation of the last letters of the* *words in the list. For example, "thinking, machine" outputs "ge", since the last letter of "thinking" is "g" and the last letter of "machine" is "e".*

- L2M; Zhou et al., 2023

| |
|---|
| Q: "think, machine"<br>A: The last letter of "think" is "k". The last letter of "machine" is "e". Concatenating "k", "e" leads<br>to "ke". So, "think, machine" outputs "ke".<br>Q: "think, machine, learning"<br>A: The last letter of "think" is "k". The last letter of "machine" is "e". The last letter of "learning"<br>is "g". Concatenating "k", "e", "g" leads to "keg". So, "think, machine, learning" outputs "keg". |

*Example of last-letter-concatenation task.*

We did not find the original dataset used in the L2M paper, so we created a word dataset using the first hundred words of Wiktionary English frequency list[3], as available on August 20, 2023.

In the L2M paper, the task decomposition step of this task was performed with an accuracy of 100%, so we decided to skip this step and give the sublists to the model directly. We performed the subproblem solving step using the "least-to-most prompting 4-shot" prompt from Appendix 7.2.5.

We performed the experiment on a hundred list of 7 random words, which lead to 5 intermediary subproblems. Additionally, we checked before each intermediary problem whether the model was able to complete the entire task using only the current context.

We ended up with a dataset containing the accuracy of the full task for each number of subproblems asked, including not asking any subproblem.

## 3. Results

For the AQuA dataset, we found non-monotonous improvement with the number of subproblems given to the model. Lanham et al. found the biggest improvement in accuracy due to Chain-of-Thought (around 15%) on the same dataset. Dips in the average score occurred when the number of subproblems increased from 2 to 3 and from 4 to 5. The detailed scores are shown in Table 1.

For the last letter task, we found continuously increasing improvements, except in the case where only the last subproblem was omitted. This anomaly is examined in the Discussion section. In the case where all subproblems were asked, we got similar results to the original LtM paper.

[3] https://en.wiktionary.org/wiki/Wiktionary:Frequency_lists/PG/2006/04/1-10000

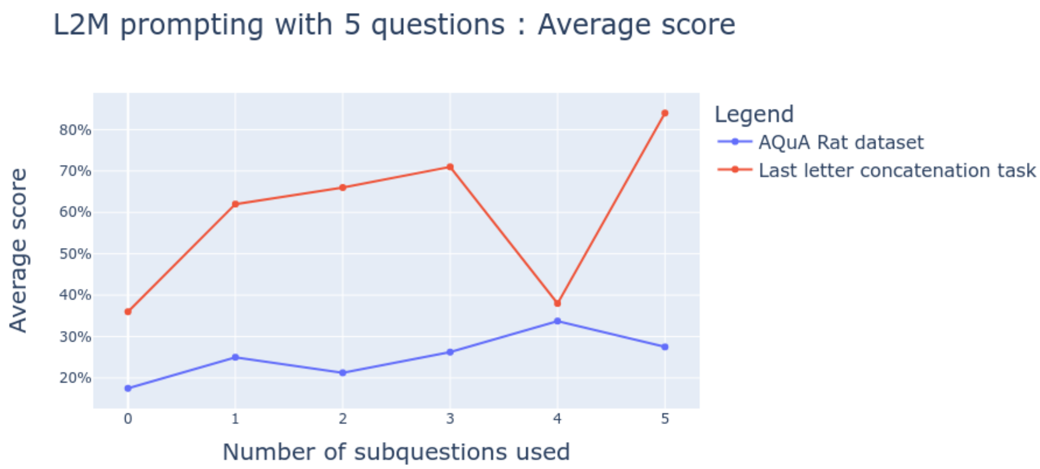


*Figure 1 – Average score for L2M prompting on the AQuA Rat dataset and the last letter concatenation task with a particular number of subproblems used.*

| # subproblems | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| AQuA Rat | 18% | 25% | 21% | 26% | 34% | 28% |
| Last letter | 36% | 62% | 66% | 71% | 38% | 84% |

*Table 1 – as above.*

## 4. Discussion and Conclusion

Here are our speculations regarding the results we obtained, which would need to be confirmed with more rigorous experimenting:

a) In the AQuA dataset decomposition prompt, we strongly suggest the model to split the problem into 5 subproblems. This might be unnatural for a model, or it might come up with useless/filler subproblems to comply with this suggestion. It might be the case that when truncating after e.g. the third of five subproblems the model is led astray by its previous answer as each time it might behave like there is only one reasoning step to be performed afterward, and the answer must be closely related. This might explain a slightly non-monotonous increase of accuracy in truncated L2M.

b) On the last letter task, the model often performed the task decomposition anyway when the final problem was asked directly. See Appendix 6.1 for an example of successful completion after only one subproblem. It seems like the L2M prompt and one subproblem is sufficient to give the accuracy boost.
Therefore, the results presented here do not cover the accuracy gain of n steps of L2M over standard prompting, but only the additional accuracy gained from doing n steps of the iterated subproblem solving method over the raw L2M prompt.

c) Regarding the dip on the last letter task at 4 subproblems, it is explained by the fact that 37 samples out of 100 succeeded with 3 and 5 subproblems, but not with 4. The example in Appendix 6.2 seems to indicate that the model did not notice that the prompt went from 5 to 7 words, and behaved as if only one more word was added. It only concatenated the 6th word and forgot about the last word. It seems possible that L2M makes the model reliant on the iterated subproblem solving loop after enough iteration, in the sense that it assumes the loop is running correctly and diligently completes the next step, not noticing that some intermediary step is missing.

Lanham et al. (2023) presented compelling evidence that the semantic content of the reasoning produced by a CoT-prompted model contributes significantly to the output, contrary to the hypothesis that CoT prompting increases the model accuracy on various tasks “merely” because CoT-like text found on the internet tends to be followed by more accurate answers. This report presents analogous evidence for L2M. Our findings suggest that the model does not engage in post-hoc reasoning, as its responses vary based on the number and nature of subproblems. The results contribute to the ongoing discourse on the efficacy of various prompting strategies in LLMs.

## 5. References

Lanham, T., Chen, A., Radhakrishnan, A., Steiner, B., Denison, C., Hernandez, D., Li, D., Durmus, E., Hubinger, E., Kernion, J., Lukosiute, K., Nguyen, K., Cheng, N., Joseph, N., Schiefer, N., Rausch, O., Larson, R., McCandlish, S., Kundu, S., Kadavath, S., Yang, S., Henighan, T., Maxwell, T., Telleen-Lawton, T., Hume, T., Hatfield-Dodds, Z., Kaplan, J., Brauner, J., Bowman, S.R., & Perez, E. (2023). Measuring Faithfulness in Chain-of-Thought Reasoning. arXiv preprint arXiv:2307.13702.

Ling, W., Yogatama, D., Dyer, C., & Blunsom, P. (2017). Program induction by rationale generation: Learning to solve and explain algebraic word problems. ACL.

Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., ... & Odena, A. (2021). Show your work: Scratchpads for intermediate computation with language models. arXiv preprint arXiv:2112.00114.

OpenAI. (2023). GPT-3.5-turbo API. OpenAI. Retrieved August 20, 2023, from https://platform.openai.com/docs/guides/gpt.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., & Zhou, D. (2022). Chain of thought prompting elicits reasoning in large language models. arXiv preprint arXiv:2201.11903.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., & Narasimhan, K. (2023). Tree of thoughts: Deliberate problem solving with large language models. arXiv preprint arXiv:2305.10601.

Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q. V., and Chi, E. H. Least-to-most prompting enables complex reasoning in large language models. In The Eleventh International Conference on Learning Representations, 2023. URL https://openreview.net/forum?id=WZH7099tgfM.

# 6. Appendix

### *6.1. Example successful last-letter-concatenation with only one subproblem*

Q: "time, do"
The last letter of "time" is "e". The last letter of "do" is "o". Concatenating "e", "o" leads to "eo". So, "time, do" outputs "eo".

Q: "time, do, these, should, my, what, over"
"time, do" outputs "eo". The last letter of "these" is "e". Concatenating "eo", "e" leads to "eoe". The last letter of "should" is "d". Concatenating "eoe", "d" leads to "eoed". The last letter of "my" is "y". Concatenating "eoed", "y" leads to "eoedy". The last letter of "what" is "t". Concatenating "eoedy", "t" leads to "eoedyt". The last letter of "over" is "r". Concatenating "eoedyt", "r" leads to "eoedytr". So, "time, do, these, should, my, what, over" outputs "eoedytr".

### *6.2. Example last-letter-concatenation failure with all but one subproblem provided*

Q: "for, great"
The last letter of "for" is "r". The last letter of "great" is "t". Concatenating "r", "t" leads to "rt". So, "for, great" outputs "rt".

Q: "for, great, like"
"for, great" outputs "rt". The last letter of "like" is "e". Concatenating "rt", "e" leads to "rte". So, "for, great, like" outputs "rte".

Q: "for, great, like, all"
"for, great, like" outputs "rte". The last letter of "all" is "l". Concatenating "rte", "l" leads to "rtel". So, "for, great, like, all" outputs "rtel".

Q: "for, great, like, all, after"
"for, great, like, all" outputs "rtel". The last letter of "after" is "r". Concatenating "rtel", "r" leads to "rtelr". So, "for, great, like, all, after" outputs "rtelr".

Q: "for, great, like, all, after, its, were"
"for, great, like, all, after" outputs "rtelr". The last letter of "its" is "s". Concatenating "rtelr", "s" leads to "rtelrs". So, "for, great, like, all, after, its, were" outputs "rtelrs".