DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. Reference herein to any social initiative (including but not limited to Diversity, Equity, and Inclusion (DEI); Community Benefits Plans (CBP); Justice 40; etc.) is made by the Author independent of any current requirement by the United States Government and does not constitute or imply endorsement, recommendation, or support by the United States Government or any agency thereof.

ORNL/TM- 2025/3753
PUB ID 229735

Prompt Phrase Ordering Using Large Language Models in HPC: Evaluating Prompt Sensitivity

Noah Thomason
Hilda B. Klasky

August 2024

OAK RIDGE National Laboratory

ORNL IS MANAGED BY UT-BATTELLE LLC FOR THE US DEPARTMENT OF ENERGY

ORNL/TM- 2025/3753

Computational Sciences & Engineering Division

PROMPT PHRASE ORDERING USING LARGE LANGUAGE MODELS IN HPC: EVALUATING PROMPT SENSITIVITY

Noah Tomasson
Hilda B. Klasky

August 2024

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831
managed by
UT-BATTELLE LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

ABSTRACT

Large language models (LLMs) have demonstrated effective performance in domain-specific tasks, often requiring a well-designed prompt to guide their responses. However, optimizing the right prompt is challenging due to prompt sensitivity—the phenomenon where small changes in the prompt can lead to significant variations in performance. In this study, we evaluate prompt performance by examining all permutations of independent phrases to investigate prompt sensitivity and robustness. We used two datasets: the GSM8k dataset, which assesses mathematical reasoning, and a custom template prompt for summarizing database metadata. Our goal was to evaluate the performance across all permutations of a sequence of prompt phrases. The study was conducted using the llama3-instruct- 7B model hosted on Ollama, with computations parallelized in a high-performance computing environment. By comparing the average index of phrases in the best and worst-performing prompts, we found that the order of independent phrases within a prompt significantly impacts LLM performance. Additionally, we used Hamming distance to assess changes between phrase orderings, concluding that prompt modifications can dramatically affect scores, often by almost random chance. These findings support existing research on prompt sensitivity. We discuss the challenges of prompt optimization, noting that altering phrases in a successful prompt does not always result in another successful prompt.

Index Terms—prompt engineering, prompt sensitivity, prompt ordering, prompt optimization, large language models, LLM performance, HPC, Hamming distance

1. INTRODUCTION

Large language models (LLMs) have demonstrated impressive performance across multiple domains. Prompt engineering is a valuable method for enhancing LLM effectiveness after training by providing additional input context for the model to draw inferences from [22]. While certain approaches, such as fine-tuning or retraining a model with different data, can improve LLM performance, these methods are often infeasible— particularly when using a closed-source model via an Application Programming Interface (API) service—and can also be computationally expensive. Prompt engineering can help 'guide' the model through a specific task. Several prompting paradigms have been tested, with some of the more widespread techniques including zero-shot vs. few-shot prompting [3], role-prompting [25], and chain of thought prompting [24]. Typically, these techniques are employed by a human to craft an optimal prompt for a given task.

Creating the perfect prompt can be challenging, time consuming, and resource-intensive for humans. To address this, work has been done to develop automatic prompt optimizers [18][11]. Some methods involve optimizing the prompt in discrete word space, as seen in [27] and [16], while others, like soft prompts, focus on optimizing the prompt within the underlying LLM embedding space, such as [26] and [7]. Although the latter can use gradients, it is often difficult or impossible to perform optimization on a closed-source model using their provided APIs. Additionally, projecting the optimized embedding into a human-readable sentence presents challenges in terms of coherency and performance preservation [1], so soft prompts are typically left in the embedding space. Discrete prompts, however, do not share this problem and can be directly optimized in a manner similar to human manual optimization.

However, optimizing a prompt in discrete space is often challenging for both machines and humans, frequently requiring extensive computation and time. One reason for this difficulty is the phenomenon known as 'prompt sensitivity,' where a model's prompt can be highly sensitive to small, unintuitive changes in human natural language [19]. These unexpected and hard-to-quantify changes make it difficult to predict how a certain modification will affect the prompt, or whether it will do so consistently. Small aspects of the prompt, such as capitalization, spaces, and delimiters, can significantly alter performance

on a task [20]. Additionally, synonyms, different grammatical tenses, and the order of exemplars in few-shot prompting paradigms can further impact performance [9].

Additionally, we found that most studies investigating ordering focus solely on the maximum difference in performance to conclude that order matters, which may overlook patterns in how this variation occurs. There is a need for more studies that investigate the effects of phrase ordering within the prompt, with a focus on understanding how the sequence of phrases within the prompt impacts performance. Therefore, in this study, we conducted two experiments to examine prompt ordering in two different domains. We:

1) Examine the ordering of a set of prompt phrases by generating all possible permutations, as well as permutations of subsets, in a mathematical reasoning task.
2) Examine the ordering of a set of instructions provided to an LLM for a summarization task by generating all possible permutations of the instructions based on a template prompt.
3) For both experiments, investigate if the position of a specific phrase in a prompt matter as well as how certain changes to the prompt affect the difference in performance (i.e., the prompt's sensitivity).

Moreover, one of the large problems of testing the ordering of a prompt is that it can be computationally expensive, as if we test all possible permutations of a list of elements in a prompt, it scales by a factorial amount. While this is not a problem if we use a closed-source API and pay larger fees, this can be an issue when using open-source APIs. Thus, we additionally introduce an easy-to-use high-performance computing layout to test LLM inference. We include an easy to setup and open-source script that demonstrates the concept with our experiments.

2. EXPERIMENTS

2.1 HPC ARCHITECTURE

In our study, we used local LLM models hosted on a high- performance computing architecture, specifically the Frontier [15] supercomputer at Oak Ridge National Laboratory. A brief overview of the architecture of the Frontier supercomputer is that it includes 8 functional Graphics Processing Units (GPUs), organized as 4 Advanced Micro Devices (AMD) MI250x with 2 Graphics Compute Dies (GCDs) each. As these are AMD GPUs, we used the Radeon Open Compute platform (ROCm) 5.7.1 and ROCm 6.0.0 [17] for communication to the GPU. We used a llama3-instruct-7B [21] model hosted on Ollama [14] for all experiments.

To explore simple parallelism, the data for evaluation was split among N nodes, which would theoretically asymptotically decrease runtime by around 1/N. The data was additionally split among 8 different Ollama servers within a node, one for each GPU. The general architecture is visualized in Fig. 1.

To set up the Ollama server, we needed to compile a custom binary without a Compute Unified Device Architecture (CUDA)-checking function that causes a segfault on load. This was patched in version 0.1.45. For working parallel servers, we also needed to compile a binary that fixes a bug that sets the wrong GPU IDs when automatically scanning for available GPUs, which was patched in version 0.3.1. We additionally include a patch in our code for the latter bug, as this was only fixed recently. We ultimately used version 0.3.1 for our experiments. Moreover, we needed to compile the GNU Compiler Collection (gcc) version 13 and create a module file to use on High-Performance Computing (HPC) architecture, but further testing indicates that this seems to have been patched out, or a one-off error.

The reasoning for using Ollama was that it is an easily usable server with built-in model-pulling and API features. We believe this improves experimental accessibility while still allowing for the use of multiple models and model parameters.

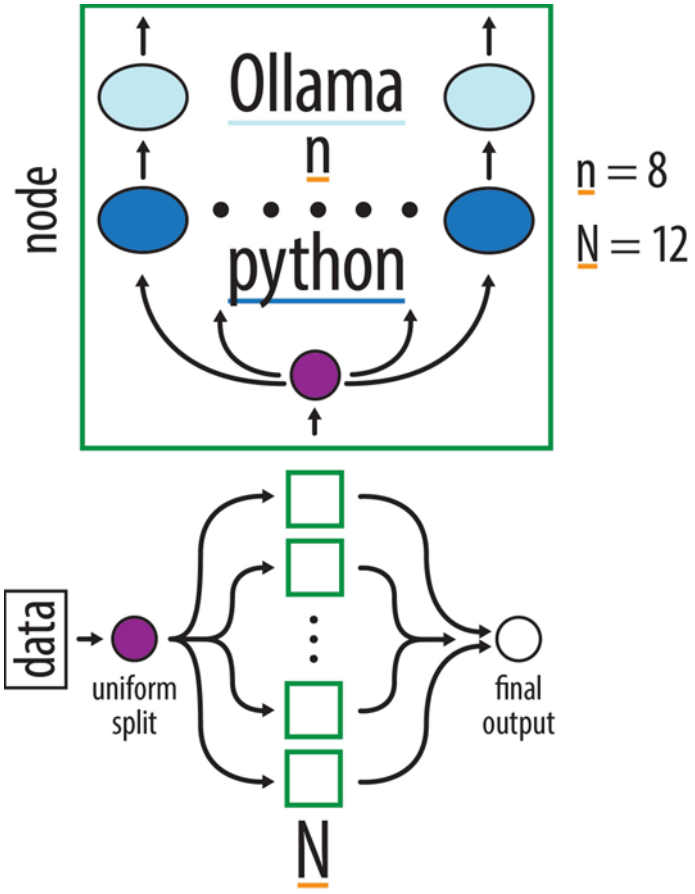

Figure 1. Architecture diagram of the HPC inference setup. (a) Intra-node setup, showing Ollama servers and Python inference and communication (upper half of the image). (b) Inter-node setup, enabling direct scalability (lower half of the image).

2.2 COMPLETE PROMPT PERMUTATION

2.2.1 Data and Methods

In our first experiment, we used a dataset containing question-answer (QA) tasks with various question and answer pairs. Specifically, we utilized the GSM8k dataset [5], a collection of grade-school mathematical word problems designed to test the mathematical reasoning of large language models (LLMs). Each problem includes a question used to generate an LLM response and a corresponding answer used to compare against the LLM's response.

To evaluate prompt sensitivity, we create permutations of a set of *n* prompt phrases combined with an empty location to insert the question. Each prompt phrase is typically a sentence, although it can be any other type of string. Formally, we define the set as

$$S = \{p_1, p_2, \ldots, p_n\} \cup \{q\}, \quad (1)$$

where p_i represents the prompt phrases and q is the question.

We then generate all k-permutations of S for $0 \leq k \leq n + 1$, filtering out those permutations that do not contain the question, as these do not represent a valid format for the task.

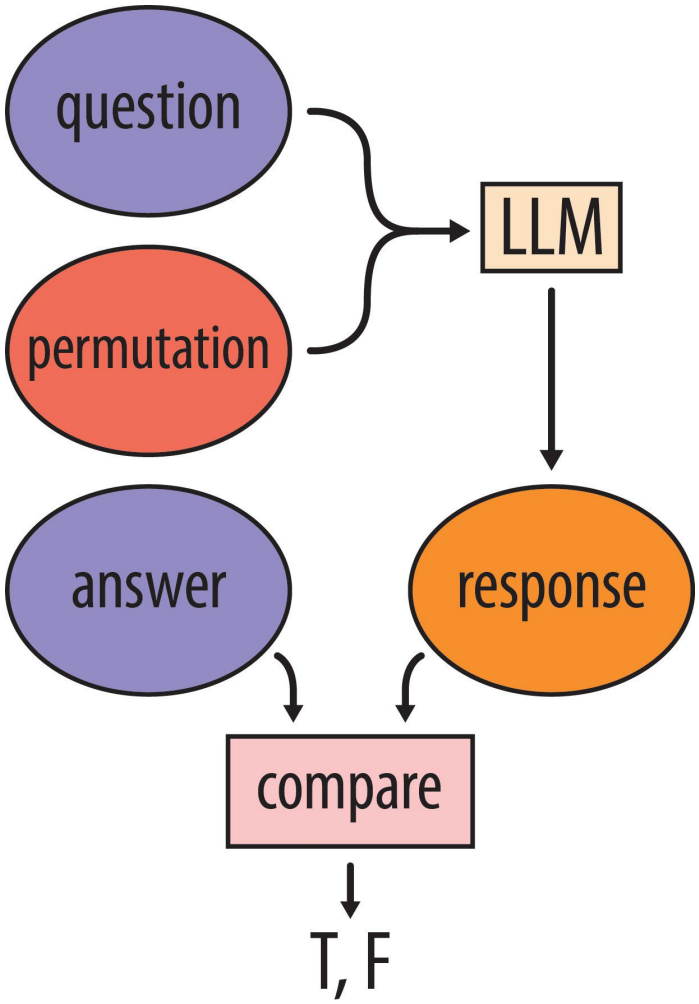

Figure 2. Architecture diagram of the GSM Complete Prompt Permutation experiment. (a) Generation of all prompt permutations (upper half of the image). (b) Evaluation of each permutation across GSM8k (lower half of the image).

The total number of valid prompt permutations can be directly calculated using the following equation:

$$f(n) = \sum_{i=0}^{n+1} [P(n+1, i) - P(n, i)] \quad (2)$$

where $P(n, i)$ denotes the number of permutations of i elements from a set of n elements.

In our experiment, we use $f(4) = 261$. The specific four prompt phrases we used, along with their respective reference sources, are provided in Appendix A.

Each of the permutations is then tested against the entirety of the GSM8k testing database of length 1319 to create an average accuracy score. When a permutation is tested, the empty location for the question is filled in with a question from GSM8k, an LLM response is generated, and then the response is compared against the ground-truth corresponding answer included in GSM8k. In our study, we compared the LLM-generated answer to the ground truth using a simple string comparison to check if the shortened answer string is in the LLM-generated answer. We note that this is a possible limitation, as it is possible that this can give a false positive. For example, the LLM could generate the number 24 in its reasoning chain but incorrectly answer 8. The current validation, given that the answer is 24, would count this as correct. In practice, this seems to be rare but should be noted. The average accuracy of a permutation across all question- answer pairs in GSM8k then corresponds to the accuracy of that specific permutation. The full architecture is visualized in Fig. 2.

To investigate if the order of the phrases in the prompt really mattered, we plotted each phrase's average position in a permutation for the 20 best and worst performing prompts in Fig. 3. As we are creating permutations of varying lengths here, we decided to normalize this index to improve comparability by remapping the space of the index to the set [0, 1], where 0 represents the very beginning of the list and 1 represents the end of the list. The top 20 best and worst 20 prompts, sorted by their score on the task, were then plotted against each other to see if they had a noticeable difference. If the prompt permutations were assigned random scores, we would expect the average prompt position to be at the exact center, at 0.5. This is because there are an equal number of elements in a list of permutations of a set for each position, and so for any random sample of the population, it is expected that there is a roughly equal number of elements in each position (thus the average being the middle, 0.5). Therefore, we can say that a prompt phrase with an average position significantly different from 0.5 "prefers" to be closer to the beginning or end of the sentence for the respective category, as higher or lower scores tend to be assigned to the permutation when the position is higher or lower. In other words, the position of a phrase in a prompt can influence its performance, and certain phrases might "prefer" to be closer to the beginning or end of the sentence to achieve better results.

2.2.2 Results

From Fig. 3, we can see that both the locations of the prompts: "SOLUTION" and "Take a deep breath..." are statistically the same as the expected value, so we can conclude that these prompt phrases do not prefer to be closer to the beginning or end of the sentence on average. However, the longer phrase starting with "Let's first understand. . . " appears to perform slightly better closer to the end of the sentence and worse at the beginning. This could be because the LLM prefers a coherent sentence at the end of the prompt instead of being interrupted by another short phrase or single word. On the other hand, "Let's think step by step" seems to perform better at the beginning. This could be because it serves as an introductory phrase that then gets into the longer, more descriptive phrases. The place to insert the question seems to perform both better and worse at the beginning of the sentence. This seems abnormal, but it could be because there are two different clusters of data exhibiting different properties. This points to the content of the phrases after the question being important in determining a prompt's performance.

We then investigated the relationship between the length of the permutation in terms of the number of phrases and its performance. As we take k-permutations of S for multiple values of k, we are able to compare the different lengths generated. From Fig. 4, we can see that as the permutation length increases, the average accuracy decreases, and the standard deviation gets larger. We note that there is a greater population in a specific k-permutation as k increases, but this should cause a smaller standard deviation. This could be indicative of the ordering of the prompt phrases becoming more unstable as the length of the prompt phrases increases, where the prompt is more sensitive to the order of its contents and the

relations between them. There is a greater chance of conflicts between two components as the system of prompts becomes more complex.

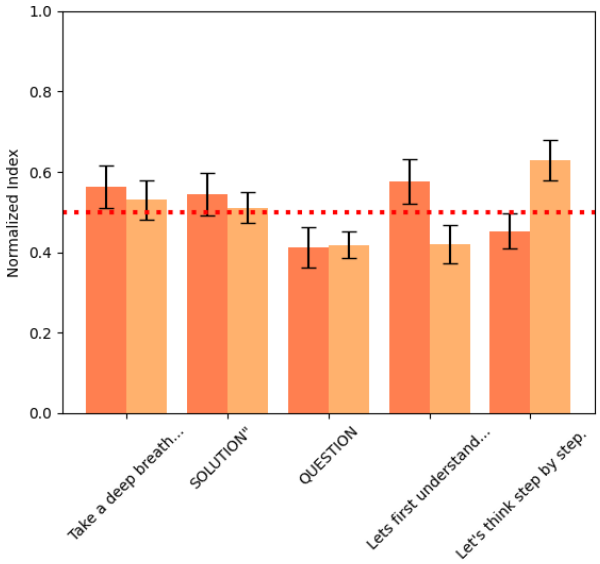

Figure 3. Phrase index for the 20 best and worst performing prompts. Dark orange represents the average of the 20 best performing prompts, while light orange represents the average of the worst performing prompts. The red horizontal dotted line indicates the expected value if performance were random. Error bars represent the standard error of the mean.

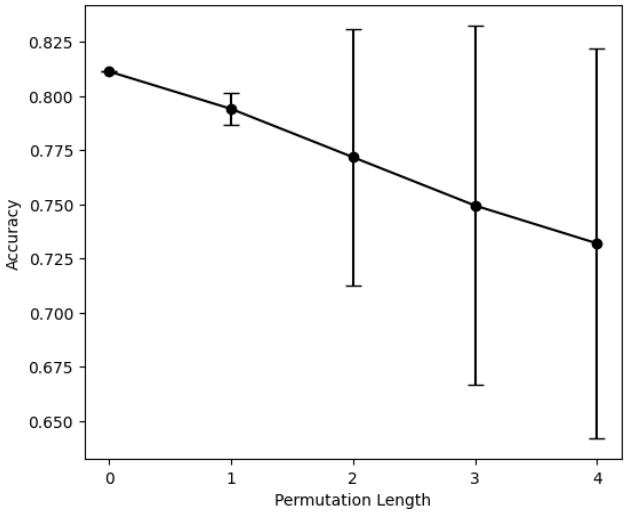

Figure 4. Average prompt accuracy compared to the number of phrases contained in the prompt permutation. The number 0 represents no prompt added to the question. Error bars are the standard deviation.

The GitLab repository for this study is found at: Complete Prompt Permutation Code; or directly via https://code.ornl.gov/summer_2024/prompt_engineering/ complete-prompt-permutation.

2.3 TEMPLATE PROMPT PERMUTATION

2.3.1 Data and Methods

In our second experiment, we evaluated the LLM on a summarization task. Fig 5 illustrates the architecture of the Database Summarization Template Permutation experiment which consists of three key components. (a) In the upper right of the image, all permutations of instructions are generated. (b) In the upper left, these permutations are used by a Large Language Model (LLM) to generate summaries based on the provided template. (c) Finally, the generated summaries are evaluated by the LLM using four specific metrics, depicted in the lower part of the image. This flow outlines the process of generating and evaluating summaries across different permutations to analyze performance variations.

For the data, we created a custom dataset of database names of length 5. We additionally included six instructions given to the LLM to generate summarized text regarding features of the database name in question. Every permutation of these six instructions was generated, giving us a total of 6! = 720 prompts to test against all database names. We used a prompt template that a specific database name and permutation of six numbered instructions are inserted into. Samples of the prompts and templates are found in Appendix B. Each permutation of instructions was then tested for each database and subsequently given a score of how good the summary is. The scoring mechanism consisted of asking an LLM to score a summary, as in [10]. The LLM, according to another prompt template, was asked to rank the summary in a category as an integer from a start to an end, for example, 1 to 5. In our study, we used templates based on a modified version of templates used in [9] taken from [13] that was based on our specific task and included the work ”Likert-scale,” which tended to increase variability of scores and had been found to increase LLM understanding of the scale [19]. The categories scored included relevance, fluency, consistency, and coherence. Each category was tested 100 times to improve granularity and consistency in response. For each permutation, all database names were tested, and then all scores were averaged into a single score for each category for the permutation. After evaluation, the scores were then normalized from their original Likert score, for example, between 1 and 5, to a score between 0 and 1. These scores, therefore, form a score vector as shown in Equation 3 for each permutation of instructions.

These scores, therefore, form a score vector as shown in Equation 3 for each permutation of instructions.

$$\hat{s} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} where \ \ x_i \in [0,1] \tag{3}$$

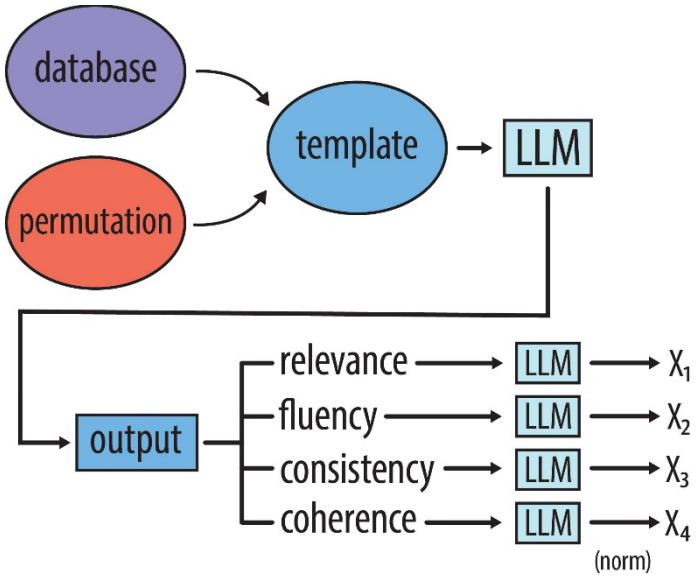

Figure 5. Architecture of the Database Summarization Template Permutation Experiment. (a) Generation of all permutations of instructions (upper right). (b) LLM generation with template (upper left). (c) Evaluating the generated summary with an LLM on the four listed metrics (lower part).

2.3.2 Results

The score vectors were plotted in 4-D space in Fig. 6. We can see that there is no separate clustering; thus there is not a similarly performing group of prompts. Additionally, most of the values are consistent in all categories, and values outside of the central area, when having a larger value in one category, often has a lower value in another category. This could mean that, when trying to optimize a prompt for a specific objective, like maximizing fluency, it often loses generalizability across all categories.

Similarly to GSM8k, we plotted the phrase index in the 100 best and worst prompts in Fig. 7. Although we did not use permutations of varying lengths, it was not necessary to normalize the phrase index; however, we did so for consistency. Since the individual prompt phrases were rather long, we used shortened phrases for readability, with the long versions provided in Appendix B.

We observed that the order of phrases did not matter for 3 out of 6 phrases, with notable differences found for 'incoming ref,' 'studies,' 'year,' and 'info.' 'Year' and 'studies' performed better at the end of the prompt, while 'incoming ref' and 'info' performed better at the beginning. Thus, we demonstrated that the seemingly random variation in order is somewhat predictable. We hypothesize that this pattern is due to the generation of additional context for the model; placing 'incoming references' and general 'information' at the beginning provides the model with a more conducive environment during its initial generation. Later, when the model addresses more specific, shorter questions (such as the year of the study) or more inferential questions (such as what types of studies can use the database), it is better prepared. However, the average index of each prompt phrase remained centered around 0.5, indicating that the tendency of a phrase to prefer the beginning or end of a prompt is generally weak; this effect is only noticeable across a large number of prompts. Next, to compare how the difference in a prompt can affect the change in score, we first introduce the concept of a hamming distance between two prompts. Since these prompt lists are of a set length, with an "alphabet" of prompts, we can generate a hamming distance between two prompts by counting the number of differences in phrases between two prompts. Notice that a hamming distance of 1 is impossible since every prompt in the alphabet must be an element of the prompt. Thus, if one phrase changes, another must swap with it. To quantify the change in score between two prompts, we simply calculate the Euclidean distance between

score vectors. All combinations of two different prompts were generated, excluding the reflexive duplicates, to generate a dataset of hamming distance and Euclidean distance pairs. We then plot them to understand the relation between these variables.

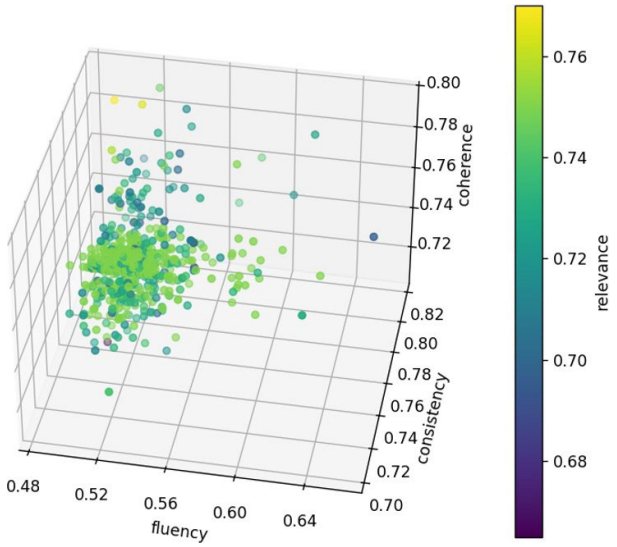

Figure 6. Score vector distribution. The x-axis is fluency, the y-axis consistency, the z-axis coherence, and the fourth color-dimension is relevance, with brighter values corresponding to positive values.

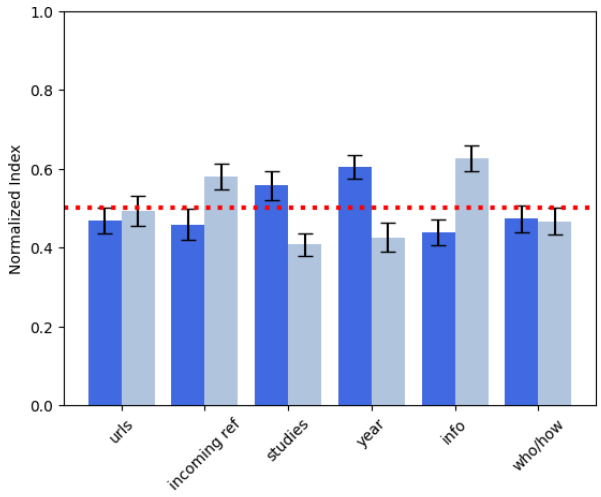

Figure 7. Phrase index in 100 best and worst performing prompts. Dark blue represents average of 100 best performing prompts, light blue represents average of worst performing prompts. Red horizontal dotted line is the expected value given that the performance is random. Error bars are standard error of mean.

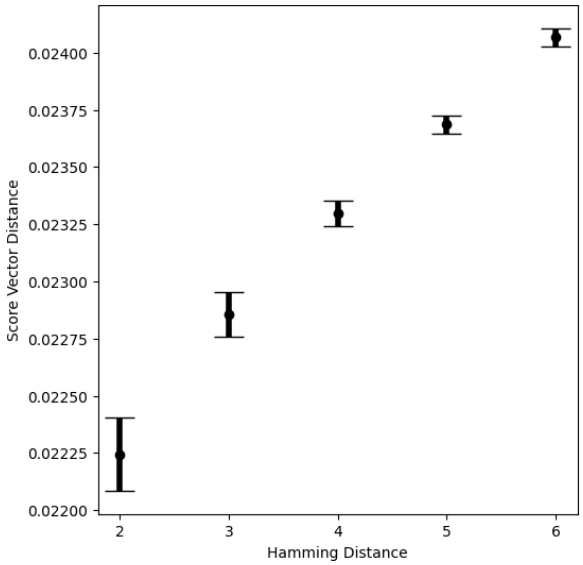

Figure 8. We compared the average score vector distance to the hamming distance. The x-axis is the hamming distance between two permutations, and the y-axis is the score vector distance. Error bars are the standard error of the mean.

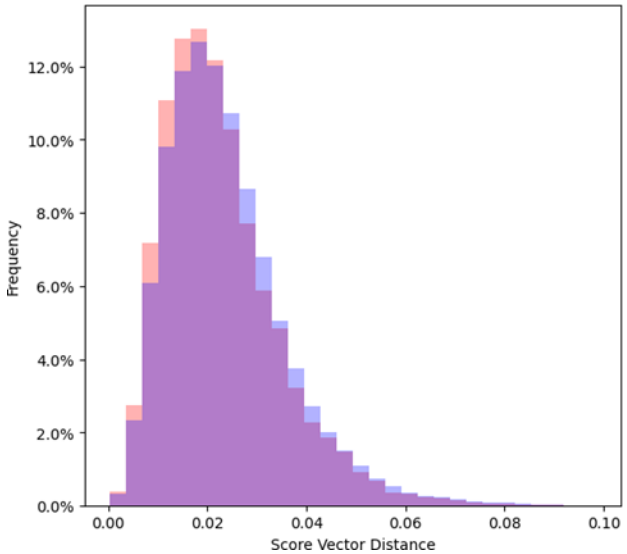

Figure 9. Normalized histogram of score vector distance comparing two categories of hamming distance. The x-axis is the score vector distance, and the y-axis is the percent frequency within the category. Light red represents the distribution with a hamming distance of 2 and 3. Light blue represents the distribution with a hamming distance of 4, 5, and 6. The purple represents the distribution overlap.

We first plot the hamming distance compared to average score vector distance, along with error bars of standard error in Fig. 8. We see that there is a clear positive relationship between hamming distance and score vector distance. The relationship appears to be slightly curved at the beginning, increasing a certain amount at first before slowing down slightly. This is generally what we expect and want, as when optimizing a prompt, if there is a well-performing prompt, it would be beneficial if a specific size change to the prompt corresponded to a predictable effect in score, so the prompt can be optimized with the predictable magnitudes of changes.

However, a different pattern emerges when we plot the distributions of the underlying averages. In Fig. 9, we present two frequency histograms comparing Hamming distances of 2-3 (low) with those of 4-6 (high). We did not create six separate distributions, as that would be difficult to interpret, but the same trend applies. The distributions overlap significantly, with only a small shift observed for larger Hamming distances. When comparing the actual means in Fig. 8, we observe that the range of the means occupies only a small portion of the distribution. This suggests that, while the general trend of greater differences in prompts leading to greater differences in scores holds true, in practice, making a small tweak to a prompt has nearly the same chance of significantly altering the score as completely changing the order.
The GitLab repository for this study is found at: Template Prompt Permutation; ordirectly via https://code.ornl.gov/summer_2024/prompt_engineering/ template-prompt-permutation.

3. DISCUSSION

We largely found in our study that the position of each prompt phrase matters for task performance, the performance of the prompt decreases as you add phrases, and optimizing for a specific ordering is difficult. This is consistent with literature on other findings on prompt sensitivity. Certain modifications to the prompt like capitalization and spacing can have an significant effect on an LLM response [20]. In [4], the order of logical premises presented to the LLM changes its overall accuracy, where ideally it should not, as logical premises are order agnostic.

Our study additionally has implications for general prompt engineering. We recommend keeping prompts short in order to minimize how much ordering matters, and possibly trying to minimize having relationally independent phrases within the prompt, as this can introduce an area of optimization that is needed. When relationally independent phrases are introduced, we recommend keeping in mind that phrases that generate context and other usable information for later phrases are key for an initial attempt at optimization.

There have been methods, such as those proposed by [27], [8], and [6], that can automatically optimize an initial prompt using a training set for the problem at hand to test the performance of the prompt. The primary improvement in these methods is achieved through continuously mutating the prompt over many iterations, with a focus on maximizing a performance score while exploring different prompting strategies.

However, the challenge with discrete prompting is the assumption that similar prompts will yield similar performance. If a prompt performs poorly, the algorithm typically moves on, ignoring similar variations. This approach becomes problematic when a poorly performing prompt could actually perform better with slight modifications. The algorithm might dismiss the entire class of similar prompts after a single poor result, potentially missing out on better-performing variants. While this risk is low, it becomes a more significant concern as the optimization process continues over a larger number of prompts.

There is no easy solution to this issue besides testing additional reorderings of prompts, which can increase both computation time and energy consumption. We suggest that future research focus on developing a robust and energy- efficient system to address this challenge.

Our results also have implications for prompt security. Small perturbations can significantly impact LLM output, which poses risks for systems that depend on robustness. If an attacker gains access to a prompt and is able to append something to it, they could decrease the model's accuracy or alter its behavior entirely. Examples of this so-called 'prompt hacking' can be seen in [19].

4. LIMITATIONS

Despite our findings, there are some limitations in this study that should be disclosed.

For the summarization-based task, we used an LLM-based evaluation. However, some studies, such as [23], have found this approach to be unreliable. If we continue using an LLM- based evaluation strategy, it might be preferable to use an architecture like G-Eval [12], which employs a Chain-of- Thought (CoT) methodology to improve scoring, rather than relying solely on prompts.

Additionally, we used a small dataset of database names for the summarization experiment and tested only one set of prompt phrases. We also used just one model, llama3-instruct- 7B, for inference and did not experiment with changing any of the hyperparameters. As a result, we do not know how model size or different models might affect the experiments.

5. FUTURE WORK

In addition to addressing the limitations mentioned above, there are several future directions for this research. Regarding HPC improvements, the parallelization could be enhanced in terms of speed or redundancy. In our study, we attempted to set the OLLAMA_NUM_PARALLEL parameter in Ollama to a value greater than 1, but this did not significantly decrease the runtime.
Future work could also focus on reducing the current parallel bottleneck, which we believe may be related to server response times. During our experiments, we observed that a 50ms inference time did not achieve the theoretical maximum of 20 inferences per second, suggesting that something within the Ollama server or the intermediary code between requests is slowing down the process.

There is also potential for improving the system's fit with the Frontier supercomputer. Specific GPU binding closest to hardware cores was not used due to issues with Ollama, so everything was done manually. Addressing this issue could increase runtime efficiency by better aligning the architecture with the hardware.

Additionally, we aim to increase confidence in the claim that prompt sensitivity is correlated with prompt length and to determine if this correlation holds across different examples, tasks, and scenarios. Strengthening this claim could lead to advancements in both automatic prompt optimization and human-driven prompt engineering.

Our testing procedures could also be expanded to cover a wider range of domains with greater variability. Furthermore, these methods could be integrated into different prompt optimization workflows to reduce the number of permutations generated and tested. However, our current methods are too computationally expensive for widespread application testing.

6. CONCLUSION

Our study underscores the significant impact of prompt phrasing and order on the performance of LLMs. We found that small modifications to prompts can lead to substantial variations in outcomes, supporting existing research on prompt sensitivity. This highlights the challenges of optimizing prompts, particularly when adding phrases or altering their order. Our findings also reveal limitations in current prompt optimization methods, which may overlook potential improvements in similar prompts. Future research should address these limitations, explore more robust and energy-efficient optimization strategies, and enhance parallelization in high- performance computing environments. Additionally, examining prompt sensitivity across diverse domains and scenarios could provide deeper insights into effective prompt engineering. Understanding the security implications of prompt sensitivity is also crucial, especially in sensitive applications like healthcare. These advancements are essential for improving LLM reliability and application efficiency.

7. ACKNOWLEDGMENTS

This research was supported in part by an appointment to the Oak Ridge National Laboratory Pathways to Computing Internship Program, sponsored by the U.S. Department of Energy and administered by the Oak Ridge Institute for Science and Education. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

And supported by Low-dose Understanding, Cellular Insights, and Molecular Discoveries program was supported by the U.S. Department of Energy, Office of Science, Office of Biological and Environmental Research, under Contract UT-Battelle, LLC- ERKPA71

8. REFERENCES

[1] L. Bailey, G. Ahdritz, A. Kleiman, S. Swaroop, F. Doshi-Velez, and W. Pan, "Soft prompting might be a bug, not a feature," 2023.

[2] S. Barrit, N. Torcida, A. Mazeraud, S. Boulogne, J. Benoit, T. Carette, T. Carron, B. Delsaut, E. Diab, H. Kermorvant *et al.*, "Neura: a specialized large language model solution in neurology," *medRxiv*, pp. 2024–02, 2024.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[4] X. Chen, R. A. Chi, X. Wang, and D. Zhou, "Premise order matters in reasoning with large language models," *arXiv preprint arXiv:2402.08939*, 2024.

[5] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[6] C. Fernando, D. Banarse, H. Michalewski, S. Osindero, and T. Rockta¨schel, "Promptbreeder: Self-referential self-improvement via prompt evolution," *arXiv preprint arXiv:2309.16797*, 2023.

[7] Y. Gu, X. Han, Z. Liu, and M. Huang, "Ppt: Pre-trained prompt tuning for few-shot learning," *arXiv preprint arXiv:2109.04332*, 2021.

[8] Q. Guo, R. Wang, J. Guo, B. Li, K. Song, X. Tan, G. Liu, J. Bian, and Y. Yang, "Connecting large language models with evolutionary algorithms yields powerful prompt optimizers," *arXiv preprint arXiv:2309.08532*, 2023.

[9] A. Leidinger, R. Van Rooij, and E. Shutova, "The language of prompt- ing: What linguistic properties make a prompt successful?" *arXiv preprint arXiv:2311.01967*, 2023.

[10] Y.-T. Lin and Y.-N. Chen, "Llm-eval: Unified multi-dimensional automatic evaluation for open-domain conversations with large language models," *arXiv preprint arXiv:2305.13711*, 2023.

[11] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre- train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[12] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, and C. Zhu, "G-eval: Nlg evaluation using gpt-4 with better human alignment," *arXiv preprint arXiv:2303.16634*, 2023.

[13] Microsoft, "Promptflow GitHub repository," https://github.com/microsoft/promptflow/tree/main/examples/flows/, 2024, [Online; accessed July 2024].

[14] Ollama, "Ollama GitHub repository," https://github.com/ollama/ollama, 2024, [Online; accessed July 2024].

[15] ORNL, "Frontier Specs," https://www.olcf.ornl.gov/frontier/, 2024, [On- line; accessed July 2024].

[16] R. Pryzant, D. Iter, J. Li, Y. T. Lee, C. Zhu, and M. Zeng, "Automatic prompt optimization with" gradient descent" and beam search," *arXiv preprint arXiv:2305.03495*, 2023.

[17] A. ROCm, "AMD ROCm Software GitHub repository," https://github. com/ROCm/ROCm, 2024, [Online; accessed July 2024].

[18] A. Sabbatella, A. Ponti, I. Giordani, A. Candelieri, and F. Archetti, "Prompt optimization in large language models," *Mathematics*, vol. 12, no. 6, p. 929, 2024.

[19] S. Schulhoff, M. Ilie, N. Balepur, K. Kahadze, A. Liu, C. Si, Y. Li, A. Gupta, H. Han, S. Schulhoff *et al.*, "The prompt report: A systematic survey of prompting techniques," *arXiv preprint arXiv:2406.06608*, 2024.

[20] M. Sclar, Y. Choi, Y. Tsvetkov, and A. Suhr, "Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting," *arXiv preprint arXiv:2310.11324*, 2023.

[21] L. Team, "The Llama 3 Herd of Models," https://ai.meta.com/research/ publications/the-llama-3-herd-of-models/, 2024, [Online; accessed July 2024].

[22] S. Vatsal and H. Dubey, "A survey of prompt engineering methods in large language models for different nlp tasks," *arXiv preprint arXiv:2407.12994*, 2024.

[23] L. Wang, W. Xu, Y. Lan, Z. Hu, Y. Lan, R. K.-W. Lee, and E.-P. Lim, "Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models," *arXiv preprint arXiv:2305.04091*, 2023.

[24] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[25] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with chatgpt," *arXiv preprint arXiv:2302.11382*, 2023.

[26] J. Wu, T. Yu, R. Wang, Z. Song, R. Zhang, H. Zhao, C. Lu, S. Li, and R. Henao, "Infoprompt: Information-theoretic soft prompt tuning for natural language understanding," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[27] S. Yang, H. Zhao, S. Zhu, G. Zhou, H. Xu, Y. Jia, and H. Zan, "Zhongjing: Enhancing the chinese medical capabilities of large language model through expert feedback and real-world multi-turn dialogue," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 17, 2024, pp. 19 368–19 376.

9. APPENDIX A. GSM8K TASK PROMPTS

TABLE I
PROMPT PHRASES TO PERMUTE

Prompts[A]
Let's think step by step. [24]
Let's first understand the problem, extract relevant variables and their corresponding numerals, and make a plan. Then, let's carry out the plan, calculate intermediate variables (pay attention to correct numerical calculation and commonsense), solve the problem step by step, and show the answer. [23]
SOLUTION" [6]
Take a deep breath and work on this problem step-by-step. [27]
{{ QUESTION }}[B]

[A]Citation number at the end of the prompt is not included.
[B]Place to insert the question of an entry in the GSM8k dataset.

10. APPENDIX B SUMMARIZATION TASK PROMPTS

TABLE II DATABASE NAMES

Prompts
Congenital Heart Surgeon's Society (CHSS)
The Kids' Inpatient Database (KID)
The Nationwide Readmissions Database (NRD)

TABLE III
INSTRUCTIONS TO PERMUTE

Prompts	***Shorthand***
Provide a brief overview of the information repository.	info
Who created the information repository, and what methods were used to collect the data?	who/how
Publications that cite or reference the information repository.	incoming ref
Websites or links related to the information repository.	urls
The year the information repository created.	year
The types of research or studies that could be conducted using data from this information repository.	studies

TABLE IV
FINAL PROMPT FORMAT

Please provide details about the information repository named {{ DATABASE_NAME }}. Specifically, we are looking for the following information: Summarize {{ PERMUTATIONS }} The output should include the NAME of the information repository, {{ DATABASE_NAME }}, the purpose of the information repository and answers to the instructions.

TABLE V
EVALUATION FORMAT EXAMPLE

You will be given one summary written about an information repository.

Your task is to rate the summary on one metric.

Please make sure you read and understand these instructions carefully. Please keep this document open while reviewing, and refer to it as needed.

Evaluation Criteria:

Relevance (1-5) - selection of important content from the source. The summary should include only important information about the database. Annotators were instructed to penalize summaries which contained redundancies and excess information.

Evaluation Steps:

1. Read the summary and the source document carefully.

2. Compare the summary to the source document and identify the main points of the source document.

3. Assess how well the summary covers the main points of the source document, and how much irrelevant or redundant information it contains.

4. Assign a relevance score on a Likert scale from 1 to 5.

Example:

Summary:

{{ Summary }}

Only respond with an evaluation number on a Likert scale from 1 to 5.

Evaluation Form (scores ONLY):

- Relevance: