Research Group Social Computing
TUM School of Computation, Information and Technology
Technical University of Munich

Enhancing LLMs: The Role of Formatting and Synonyms

How seemingly minor prompt formatting choices impact performance

Pontus Amgren ✉, **Tobias Leibrock** ✉, **and Gabriel Trannoy** ✉

TUM School of Computation, Information and Technology, Technical University of Munich

✉ pontus.amgren@tum.de
✉ tobias.leibrock@tum.de
✉ gabriel.trannoy@telecom-paris.fr

09 March, 2025

Abstract — In this paper we investigates the impacts of prompt formatting and semantic variations on the performance of Large Language Models (LLMs). By systematically analyzing the sensitivity of LLMs to minor prompt structure variations and synonym replacement, we identify optimal formatting strategies that influence influence model accuracy on a large corpus of test tasks. By providing a novel Python package, `REFORMAT`, we enable users to apply these strategies effortlessly to their prompts. Our experiments use the Super-Natural-Instructions dataset to validate our approach across multiple models and types of tasks, demonstrating that even minor formatting changes can lead to major performance variations. This work underscores the critical role of prompt design and format in applications and should be regarded as a starting point for future work in this direction.

1 Introduction

Large Language Models (LLMs) have revolutionized natural language processing, demonstrating remarkable capabilities across a wide range of tasks. As these models become increasingly powerful and widely adopted, the way we interact with them - through carefully crafted prompts - has become a crucial area of study. Prompt engineering has emerged as a key methodology for optimizing model performance without requiring additional training data or computational resources.

1.1 Motivation

The field of prompt engineering includes multiple aspects, from content selection and few-shot learning strategies to the structural elements of prompt design. While considerable attention has been given to prompt content and learning approaches, the impact of prompt formatting and presentation has received less attention [1], [2]. When formatting prompts the user can focus on large scale reformatting which includes changing the actual content of the prompt, or small scale formatting which includes changing the wording, example selection, and formatting of the prompt. These smaller scale formatting choices are often overlooked as they are assumed to be unimportant and do not directly make sense to humans.

The common assumption that formatting choices have negligible impact on model performance compared to factors like data quality or model architecture has led to prompt formatting aspects being largely overlooked in both research and practical applications. However, emerging evidence suggests that formatting decisions can substantially affect model responses, potentially introducing unintended biases and inconsistencies in model behavior [2], [3]. This sensitivity to formatting presents challenges for reproducibility in research and reliability in practical applications.

Since each model is trained on a different dataset, the models can perform better when data is formatted similar to the training data therefore increasing the likelihood of good performance on the task. This can be used to create so called "expert rules" which are rules that are known to work well for a given model on any generic task template.

1.2 Related Work

Recent work by White et al. (2023) [4] and He et al. (2024) [3] has shown that pre-trained LLMs exhibit unexpected sensitivity to formatting choices, with performance variations of up to 76 accuracy points for LLaMA-2-13B and approximately 10 points on average across multiple tasks and models. Their research provides a foundation for understanding the significance of prompt formatting in model performance.

Another recent work, by Wahle et al. (2024) [5] investigates the effects of prompt paraphrasing on different tasks. They find that lexical changes, such as synonym replacement, has a positive impact on summarization, wrong candidate generation, and title gen-

eration tasks. However, on tasks such as question generation and question rewriting, lexical changes may have a lesser effect, or even hinder performance.

Our work builds upon these findings by investigating two key aspects of prompt engineering: formatting and semantic variation through synonym replacement. We reproduce the results from [4] and extend the work by introducing a novel approach to prompt augmentation through synonym substitution. This combined methodology allows us to explore both structural and semantic dimensions of prompt engineering while maintaining prompt intent and meaning. To create a practical application for our findings we develop a Python package called `REFORMAT`, which allows users to easily apply the formatting rules and synonym replacement to their own prompts. For this we use the Natural-Instructions dataset [6] to empirically discover so called "expert templates" that are used to generate prompts for the language models. These templates include choices for wording, example selection, formatting, and synonyms.

In addition to rule-based formatting, we explore an LLM-driven approach to prompt optimization. While rule-based methods rely on empirically validated templates, the LLM-based method utilizes a iterative strategy, optimizing prompts for specific tasks rather than relying on general formatting principles. This aligns with recent LLM-as-a-judge approaches, where LLMs are used to evaluate and refine model outputs in alignment tasks [7], [8]. However, this method comes with trade-offs. Rule-based formatting is stable and reliable, as it is derived from extensive empirical testing across multiple tasks. LLM-based formatting has the potential for higher task-specific optimization, as the model directly adjusts prompts based on its own feedback and output from the actual prompts.

2 Methodology

We tackle the prompt enhancement problem in two distinct ways; formatting changes and prompt rephrasing. These methods were developed and evaluated separately, and later combined into one package. The formatting part is largely based on White et al. (2023) [4], where we reproduce the results from the paper. Additionally, we introduce formatting values which were not included in [4]. Effects of rephrasing is explored alongside formatting, and uses the same dataset for evaluation. The rephrasing in implemented as lexical changes on the word-level, discussed in further detail in Section 2.5.

2.1 Dataset

We use the Super-Natural-Instructions dataset [6] throughout the project, which is the same dataset used in [4]. The dataset contains over 1600 tasks, each consisting of a definition, positive and negative examples, and a number of instances of the task, among other things. The instances contain both an input and a ground-truth output. Some tasks are of generative nature, and other of discriminative. For example, translation tasks for the former, and binary classification tasks for the latter.

We focused on a subset of the tasks that were easy to evaluate, and contained a large amount of samples to test. A complete list of these tasks can be found in the Appendix.

The chosen tasks contain many different types that can be grouped into four major categories: text classification, multiple-choice question answering, natural language inference, and language understanding. These major categories include tasks such as toxic language detection, stereotype detection and text quality assessment for text classification. Task distribution into the four categories can be seen in Figure 1. This is a subset of the 53 tasks used in *How I learned to start worrying about prompt formatting* [4]. For our work we removed tasks that require program execution or advanced NLP evaluation metrics to distinguish between a good and bad model response. All chosen tasks are based on a simple correct and false response and an example task can be seen in Figure 5.

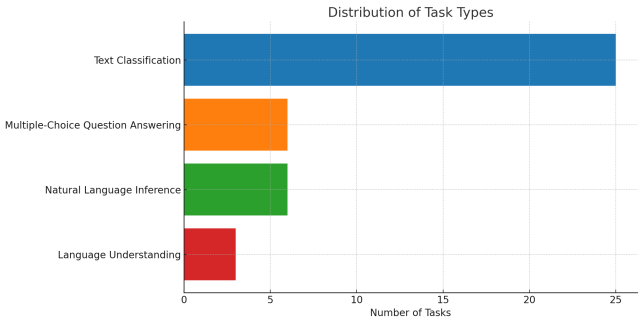

Figure 1 Grouping of all 40 tasks evaluated in the Super-Natural-Instructions dataset. Grouped by general NLP task type. Majority of tasks are text classification tasks, making up 62.5% of the tasks.

2.2 Formatting Classes

We consider four main classes of formatting: *separation*, *casing*, *item formatting*, and *enumeration*. We use the same naming convention as [4]

for the classes, but not the same set of allowed class values, although there is a significant overlap. To clarify the effect of each formatting class, we use the incomplete prompt in Figure 2 as reference.

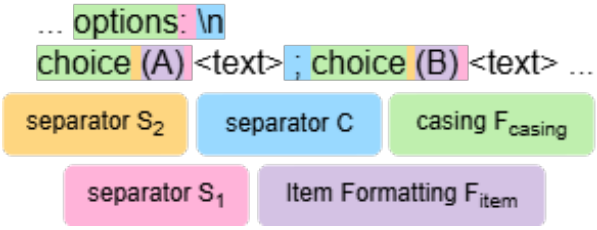

Figure 2 Part of a prompt where the formatting types have been visualized.

separation determines how different parts of the prompt are separated, and consists of three main parts: separation of descriptors and text $\mathcal{S}_1$, separation of discriptor and formatted items $\mathcal{S}_2$, and separation of prompt fields $\mathcal{C}$. $\mathcal{S}_1$ is applied after descriptors and after formatted items. In the example prompt, this corresponds to the ': ' after 'options' and also the blank space after (A) and (B). The effect of $\mathcal{S}_2$ on the example prompt are the blank spaces between 'choice' and the formatted items (A) and (B). $\mathcal{C}$ is used to separate larger constituents of the prompt such as: definition, demonstrations, and answer options. In the example prompt, this is illustrated by a `newline` character after 'options: ', and also by the semicolon that separated the two answer options.

casing $\mathcal{F}_{\text{casing}}$ determines the casing of descriptors in the prompt (e.g. 'example' or 'choice'). Note that other parts of the prompt (e.g. `<text>`) is unaffected by the type of casing.

item formatting $\mathcal{F}_{\text{item}}$ is applied to enumerables that signify different demonstration instances and different choices for a classification task. $\mathcal{F}_{\text{item}}$ is divided into two main parts: item wrapping $\mathcal{F}_1$ and item enumeration $\mathcal{F}_2$. $\mathcal{F}_1$ determines what characters surround the enumerable. For the example prompt, this would represent the parentheses surrounding A and B. $\mathcal{F}_2$, on the other hand, determines the choice of enumerables themselves, whether alphabetical, roman, or other (e.g. "A, B, C, ...", or "i, ii, iii, ...").

A complete list of values for all classes are given in Figure 3 in Appendix.

2.3 Format Evaluation

To manage prompts, and facilitate formatting, we follow a template structure. Defined as a class in Python, different parts of a prompt, such as definition and demonstrations can be handled separately. The different formatting classes can then easily be applied to only the intended sections.

We create templates that vary across the formatting search space, and evaluate these on the tasks listed in Section 2.1. For each instance of a task, we only consider whether the generated output was correct or not. We do this for the models described in Section 2.4, and compile the results for each task as best and worst performance, as well as performance difference across the models. Additionally, for each model, we present the highest-performing combination of formats.

2.4 Tools

For our evaluation scripts and analysis of the dataset we used Python. It was chosen as the implementation language, largely due to its established role in the NLP domain. Besides making implementation easier with easy access to APIs and NLP libraries, we also contribute to the community by developing our own Python package for prompt improvement. Through the `aisuite` library, we connect to OpenAI and Groq APIs to access the following models: GPT-4o, Llama 3.3 70B (llama-3.3-70b-versatile), Llama 3.1 8B (llama-3.1-8b-instant), Mixtral 8x7B (mixtral-8x7b-32768), and Gemma 2 9B (gemma2-9b-it). These are the models that were evaluated and for which expert rules exist. However, any model that is available through the OpenAI or Groq API can be used without the need for major changes.

2.5 Synonym Replacement

In addition to investigating the effect of reformatting, we also consider rephrasing the prompt, in an attempt to understand whether some words are more effective than others when creating a prompt. Given a formatted prompt, we replace certain key words in the task definition with synonyms such that semantic similarity is preserved. We then evaluate the downstream task performance of the augmented prompts and compare them to each other, as well as the original prompt.

Due to time and processing constraints, focus is put on certain key words that were chosen as candidates for synonym replacement. The process to determine these candidates consists of two main steps. First, word frequency analysis on the task definitions from the Natural-Instructions dataset was carried out, in order to get a list of words commonly occurring in task definitions. In the next step, the synonym candidates

were manually chosen from the most common words. Stop words such as "the" or "a" were discarded, and common words such as "task", "answer", and "select" were chosen.

When considering a candidate word in a sentence to be replaced by its synonyms, we refer to this as a target word. To generate synonyms for a target word in a text, the following steps are taken. First the input text is tokenized and the target word is masked out. Second, a BERT model is used to predict the most likely tokens for the masked index. Finally, the top tokens (with exception of the token corresponding to the target word) are decoded and returned as an array of strings.

Before accepting a generated synonym as a replacement for a target word, a check for semantic similarity is performed. The check is done by computing the cosine similarity between the embedding of the original and augmented text, and comparing this to a threshold value. The threshold value was determined beforehand via a qualitative approach. A small number of sentence pairs of varying degrees of similarity were manually compared, and their semantic similarity was calculated. A threshold value of 0.99 was then selected to reflect the point at which sentences were considered semantically equivalent by the authors.

A total of 40 tasks from the Natural-Instructions dataset [6] are used to evaluate the effect of prompt rephrasing. Given a language model and a target word, we iterate through the tasks and generate modified prompts by applying synonym generation to the task definition. Then, for up to all instances in the task, both the original prompt, and the augmented prompts are used in the model completion to generate an answer. The results are saved in JSON format, keeping track of both total number of evaluations, as well as the number of correct answers, for all synonyms. One JSON synonym performance file is kept for each unique language model.

When provided with an input prompt to augment via synonym replacement, we use the synonym performance file corresponding to the language model of choice to find possible synonym replacements. We compare every word with possible synonyms and replace words with better performing synonyms, given that the semantic similarity does not change too much. Pseudocode for the algorithm is given in Algorithm 1 in the Appendix. If the performance file for the chosen model is non-existent, then the prompt remains unchanged, as no general rules across models is available.

3 Results

We evaluate the performance of our approach on a range of model families and individual models demonstrating that our approach is applicable to a wide range of models including closed-source and open-source models. By only relying on the input prompt and the model's output, our approach is model agnostic and can be applied to any LLM.

For our expert rules results that are used in `REFORMAT` we average our Super-Natural-Instructions results that were found through random search over the formatting space of all formatting classes combined. This allows us to generate expert rules that combine the best formatting classes for a large amount of different tasks. We compare these empirically found expert rules with the baseline performance of the unformatted prompts in additional Super-Natural-Instructions tasks.

3.1 Super-Natural-Instructions Results

In general we observe similar results to [4] across the Super-Natural-Instructions dataset. This includes large variations in some tasks and small to negative improvements in other tasks. In Table 2 we show the results for a small subset of the Super-Natural-Instructions dataset for Llama-3.2-8B. We observe that some tasks show large improvements and others only show small to negative improvements. We measured a small improvement between baseline and optimal formatting of 0.07 for task 070, and a similarly small improvement of 0.08 for task 317. This demonstrates the task already being performed at the highest possible accuracy for the model in this case Llama-3.2-8B. For other tasks such as 069 and 279 we can observe a larger difference, with task 069 having an improvement of 0.15 and task 279 having a difference of 0.12. This indicates that the model is capable of performing the task at a higher accuracy than the baseline, with a different format that either helps the model to generalize better or locate the task better in the parameter space. For both types, small and large improvements, there are formats that perform considerably worse than the baseline, indicating that for all kind of tasks we can find formats that perform consistently worse. We will discuss this finding further in the discussion section and go into adversarial applications of our approach in our outlook.

Comparing our best performing formats for this subset of tasks to the results from *How I learned to start worrying about prompt formatting* [4], in Table 1, we

	Task	Separator	Casing	Item	Enumeration	Accuracy
Ours	Task069	"::"	upper	"()"	numbers	0.93
Theirs	Task069	"::"	upper	")"	alphabetical	0.85
Ours	Task070	"\n"	title	"[]"	numbers	0.64
Theirs	Task070	" "	title	"<>"	numbers	0.80
Ours	Task279	"\t"	title	")"	none	0.49
Theirs	Task279	"::"	title	"none"	none	0.64

Table 1 Comparison of discovered formatting classes with highest performance for specific tasks compared to the results from *How I learned to start worrying about prompt formatting* [4]. The discovered formatting classes share many common elements in the formatting classes, agreeing with the findings from [4]. Our additional elements for the formatting classes were chosen rarely indicating that the additional elements are not crucial for model performance. Both our and their results are based on Meta's Llama model family with our results using Llama-3.3-8B and their results using Llama-3.1-70B.

Task	Baseline	Worst	Best	Delta
Task069	0.78	0.56	0.93	0.37
Task070	0.57	0.26	0.64	0.38
Task279	0.41	0.32	0.49	0.17
Task317	0.57	0.19	0.69	0.50

Table 2 Accuracy variations between formatting classes for Llama-3.2-8B on a small subset of the Super-Natural-Instructions dataset. Worst formatting class is the formatting class in this evaluation that performed the worst compared to the baseline. Best formatting class is the formatting class that improves the accuracy the most compared to the baseline. For this evaluation 80 formatting combinations were tested with 200 task samples per formatting combination.

can observe that our best performing formats are similar to the ones found in the paper. This indicates that even though we use a newer version of the Llama model family, the formatting classes found in the paper are still relevant and useful. We find some minor differences but conclude that the formatting classes found in the paper are still among the best performing ones. Our additional formatting classes are chosen rarely showing that the paper included the most important options for each class.

To find the expert rules for the different model families we run our random search over all 40 tasks for each model and average the performance result for each combination of formatting classes. Afterwards we select the best performing options for each formatting class and use them as expert rules. The results for each model are shown in Table 3. We can observe for models from the same model family, both models perform best with similar formatting choices. The largest spread is found for Llama-3.1-8B with 0.07, with the lowest spread found for GPT-4o with 0.03. The spread also corresponds with benchmark performance for the models, showing that more capable models are more robust to the actual formatting of the task. Noteworthy is the similarity of the formatting choices for the same model family, indicating that models trained or distilled form the same dataset tend to prefer similar formatting choices. Both Llama models by Meta only differ in the item and enumeration formatting choices with the item formatting being very similar.

3.2 Synonym Replacement Results

Synonym replacement is implemented for GPT-4o, Llama-3.3-70B, and Llama-3.1-8B. The evaluation results are shown in Table 4. The results indicate that the implemented synonym replacement does not consistently improve performance on the downstream task. For GPT-4o-mini, rephrasing resulted in a decrease in performance by three percentage points. Similar results are shown for Llama-3.3-70B, though accuracy only dropped by one percentage point. Only for Llama-3.1-8B we see an improved result for the rephrased prompts.

3.3 REFORMAT

With our findings we created the `REFORMAT` package, which is a Python package that allows users to improve the performance of language models by formatting the input prompt according to our empirically found expert rules. The package is available open-source on GitHub and can be used via command line or Python imports. Additional information about the package can be found in the README[1] file on GitHub. `REFORMAT` offers two main modes of operation that improve user prompts in different ways.

Expert Mode allows users to apply our empirically found expert rules to their prompts. The user can

[1] The README file for REFORMAT is available here

Model	Separator	Casing	Item	Enumeration	Delta
Llama-3.1-8B	" "	upper	")"	prefix_upper_a	0.07
Gemma-2-9B	"\t"	none	"[]"	prefix_lower_a	0.07
Llama-3.3-70B	" "	upper	"()"	alphabetical	0.04
Mixtral-8x7B	"\n"	none	"none"	lower_roman	0.06
GPT-4o	" "	title	")"	none	0.03

Table 3 Expert rules for different models tested ordered by model parameter size. The expert rules are the best performing options for each formatting class. We can see a trend of the larger models having a lower spread in performance, indicating that the larger models are more robust to the formatting classes. Smaller models such as Llama-3.1-8B and Gemma-2-9B have a higher spread indicating a larger variance in performance for the different formatting classes.

Model	Original	Rephrased	Difference
GPT-4o-mini	0.62	0.59	-0.03
Llama-3.3-70B	0.74	0.73	-0.01
Llama-3.1-8B	0.25	0.26	+0.01

Table 4 Results for different models evaluated on 10 instances from 39 tasks for a total of 390 instances. The results are displayed as accuracy from the original and rephrased prompts, as well as difference between these, which can be interpreted as the gain from rephrasing the prompt.

choose between multiple available prompt templates and apply the format improvements to the full prompt. We match the model formulation from the user to the closest available model in our expert rules and apply the corresponding formatting choices. The final prompt can be compared to the original prompt to see the formatting applied.

LLM Judge Mode allows users to iteratively improve on prompts for any language model. After choosing and filling out a prompt template we randomly sample the formatting class space and apply the formatting to the prompt. In this mode of REFORMAT the package is iteratively improving a single prompt for a specific task therefore the prompt needs to contain all necessary information for the entire task. The new prompt and original prompt are then fed to the defined language model and the model's output for both prompts is judged by an additional language model. This judgement is then used to rank the formatting options and decide on the current best one. After a predefined number of iterations the best prompt with the chosen formatting choices is returned.

4 Discussion

Our results confirm that prompt formatting has a significant impact on LLM performance. We observed substantial variations in model accuracy depending on formatting choices, with some models showing performance differences of over 30 percentage points between the best and worst formats. This supports previous findings by White et al. (2023) [2] and He et al. (2024) [3], confirming the idea that seemingly minor structural variations can lead to major performance shifts.

One of the major observation is that larger models tend to be more robust to formatting changes. While smaller models like Llama-3.1-8B exhibited greater performance swings, more advanced models such as GPT-4o demonstrated a lower sensitivity to formatting. This suggests that larger models are better at adapting to different prompt styles, likely due to their richer training data and more generalized understanding of language. Also the large parameter count of the models might play a role in this, as the model has more capacity to generalize over the formatting space and perform well on all formatting classes at the same time.

We also found that the best-performing formatting choices were remarkably consistent across models from the same family. For example, Llama-3.1-8B and Llama-3.3-70B preferred similar formatting structures, indicating that training data and model architecture significantly influence formatting preferences. This consistency suggests that prompt optimization could be integrated into the model training process and the model pipeline to improve performance. We believe this to be dependent on the training data distribution and especially the formatting of similar tasks in the training data.

For LLM-iterative-based vs. rule-based formatting, we identified clear trade-offs. Rule-based formatting with our empirically found expert rules offers stability and reproducibility, making it a reliable approach for users who require consistent performance improvements for their tasks. In contrast, LLM-iterative-based formatting optimizes prompts dynamically for specific

tasks, potentially yielding better results in task-specific contexts. However, it comes with risks, such as overfitting to short-term improvements and the possibility of unreliable judgment from the LLM evaluating the prompts similar to the results from Wei et al. (2024) [7] and Wang et al. (2025) [8].

Our synonym replacement experiments yielded mixed results with little to no consistent improvement. One possible limitation of our approach was the static synonym replacement strategy, which did not account for contextual variations. A more advanced approach, incorporating semantic embeddings or contextual rewording, could improve results in future studies. We belive this could be due to synonyms being very similar in the embedding space and the semantic similarity that is used by the actual model to derive the next token.

5 Outlook

As we based our work on the Super-Natural-Instructions dataset, we assume in our study that prompt formats are following a certain structure as described in Section 2.2. We also only worked, for synonyms replacement, with a specific and limited set of target words and replacements. Therefore we can assume that our results are more accurate for specific fields covering the more frequent words present in Super-Natural-Instructions dataset as stated in Section 2.5.

The aspect of adversarial attacks leveraging weaknesses of language models cannot be overlooked. In our case, an adaptation of our `REFORMAT` method could be used to select a worse performing format for a given model and task, reducing the quality of any answer. This can be achieved just by changing the maximizing accuracy goal by a minimization goal, a minus sign in the right place could be enough to do so. Such attacks can be harmful depending on the truth value the user wants or need, for example with automated process using API access and such modified prompt improvers or in medical fields where the slightest error can have nefarious consequences.

For open source models, the weights and activations could be investigated to help discover what format is performing better. It also seems that models from the same developers have similar best performing formats, and in that perspective the influence of the training data on the best performing prompt format. There may even exist format improvements already applied by the api providers for some specific models that can impact greatly the prompt formatting strategies. An other approach on formatting strategies is to include internal model tokens into the evaluation, such as the start and end tokens.

The space of all possible formats is too large for us to fully test out (approx. 10^5 different formats 3) and it does not posses any noticeable property for optimising a maximum research. Conducting an exhaustive search would be time and resource consuming, as it would require more than 10 million calls to the LLM. Therefore we conducted random searches across the overall format space to evaluate each models.

6 Conclusion

In this study we achieved the reproduction of the experimental approach introduced in *How I learned to start worrying about prompt formatting* [4] and we managed to obtain comparable results with prompt formatting. This extensive analysis of prompt formats, for each model that we studied, formed our basis to yield our expert rules later on. Our work also introduces the `REFORMAT` Python package which enable users to improve their prompts with reformatting strategies, throughout our empirically found expert rules, by iteratively improving their prompt with an LLM Judge, or with a synonym replacement process. On our limited synonym replacement testing we were unable to show significant differences in the model performance.

Acknowledgements

We would like to acknowledge the assistance of various AI tools that supported our work:

- Grammarly - Used to check the grammar and spelling of our writing
- DeepL - Used to check the grammar and spelling of our writing
- ChatGPT - Assisted in the ideation process and helped structure some section headings of this paper. Also used for pseudocode generation in synonym section.

These tools were used as aids to improve the quality of our work while maintaining the integrity of our original research and analysis.

We extend our gratitude to Jakob Sturm, our supervisor for this project, for his valuable guidance and support throughout our project. His feedback and insights were great and helped to refine our approach and get to the current state of our work.

Appendix

$$\mathcal{S}_1 = \{\texttt{"},\texttt{' '},\texttt{'\textbackslash n'},\texttt{' -- '},\texttt{'; \textbackslash n'},\texttt{' || '},\texttt{'< sep >'},\texttt{' - '},\texttt{'\textbackslash n '}\}$$
$$\mathcal{S}_2 = \{\texttt{"},\texttt{' '},\texttt{' '},\texttt{'\textbackslash t'}\}$$
$$\mathcal{C} = \{\texttt{"},\texttt{' ::: '},\texttt{' :: '},\texttt{' : '},\texttt{'\textbackslash n\textbackslash t'},\texttt{'\textbackslash n '},\texttt{': '},\texttt{' - '},\texttt{'\textbackslash t'},\texttt{'\#'},\texttt{'<>'},\texttt{':'}\}$$
$$\mathcal{F}_{\text{casing}} = \{\texttt{f(x) = x},\ \texttt{f(x) = x.title()},\ \texttt{f(x) = x.upper()},\ \texttt{f(x) = x.lower()}\}$$
$$\mathcal{F}_{\text{item}} = \{\texttt{x} \mapsto \texttt{f(g(x))} \mid \texttt{f} \in \mathcal{F}_{\text{item1}}, \texttt{g} \in \mathcal{F}_{\text{item2}}\}$$
$$\mathcal{F}_{\text{item1}} = \{\texttt{x} \mapsto \texttt{(x)},\ \texttt{x} \mapsto \texttt{x.},\ \texttt{x} \mapsto \texttt{x)},\ \texttt{x} \mapsto \texttt{x)},\ \texttt{x} \mapsto \texttt{[x]},\ \texttt{x} \mapsto \texttt{<x>},\ \texttt{x} \mapsto \texttt{x:},\ \texttt{x} \mapsto \texttt{x?}\}$$
$$\mathcal{F}_{\text{item2}} = \{\texttt{x} \mapsto \texttt{x + 1},\ \texttt{x} \mapsto \texttt{'A' + x},\ \texttt{x} \mapsto \texttt{'a' + x},\texttt{x} \mapsto \texttt{0x215F + x},\ \texttt{x} \mapsto \texttt{x00},\ \texttt{x} \mapsto \texttt{ROMAN}x\texttt{.lower}, x \mapsto \texttt{ROMAN}x\texttt{.upper}\}$$

Figure 3 All formatting classes that were used for prompt formatting, and their respective possible values or functions.

The 40 tasks that were used from the Super-Natural-Instructions dataset: `task050`, `task065`, `task069`, `task070`, `task114`, `task133`, `task190`, `task279`, `task280`, `task286`, `task316`, `task317`, `task319`, `task320`, `task322`, `task323`, `task325`, `task326`, `task327`, `task328`, `task335`, `task337`, `task385`, `task580`, `task607`, `task608`, `task609`, `task904`, `task905`, `task1186`, `task1283`, `task1284`, `task1297`, `task1347`, `task1387`, `task1420`, `task1502`, `task1612`, `task1678`, `task1724`.

Algorithm 1 The pseudocode describes the *apply_synonym_rules* function that uses evaluated synonyms to replace words from a given prompt with better performing synonyms. `Note: this pseudocode was generated by ChatGPT`

Require: $input\_text$, $synonym\_file$, $similarity\_threshold$
Ensure: Augmented text with optimized synonym replacements

Load $synonyms\_data$ from $synonym\_file$
Map each synonym to its category
for all word w in $input\_text$ **do**
    Identify category C of w
    if C is not found in $synonyms\_data$ **then**
        Continue
    end if
    Find better-performing synonyms $S = \{s \in C \mid score\_s > score\_w\}$
    Sort S by performance (highest first)
    for all synonym $s \in S$ **do**
        Replace w with s in $input\_text$ to form $temp\_text$
        if $similarity(input\_text, temp\_text) \geq similarity\_threshold$ **then**
            Update $w \leftarrow s$
            break
        end if
    end for
end for
return Augmented text

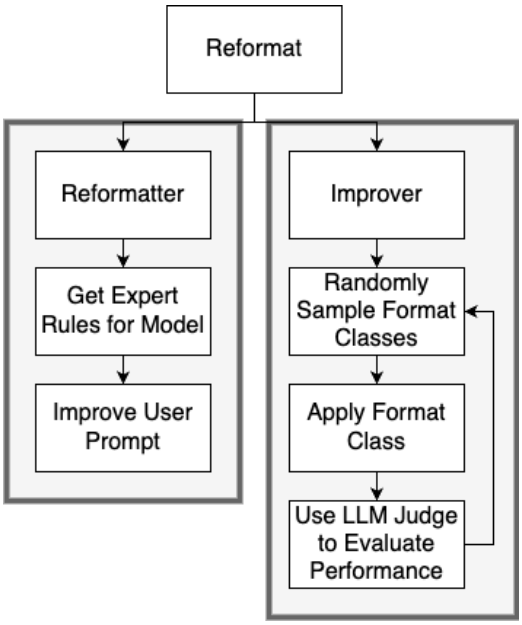

Figure 4 Both operating modes for `REFORMAT`. The user can choose between a predefined prompt template or use their own prompt. The user can then apply the formatting choices to the prompt or iteratively improve on the prompt with the LLM Judge Mode. In the end both modes return the improved prompt with the corresponding formatting choices.

Task Name: Coherence Classification
Definition: In this task, you will be shown a short story with a beginning, two potential middles, and an ending. Your job is to choose the middle statement that makes the story coherent/plausible by writing "1" or "2" in the output.
Example Input:
Beginning: Butch had a really old computer.
Middle 1: Butch decided to order at new computer online.
Middle 2: Butch noticed that a storm was approaching his town.
Ending: It arrived and Butch was much happier.
Correct Output: 1

Figure 5 Example of a Coherence Classification task used to assess NLP model capabilities. The task includes a definition given to the model with the actual example input. The correct output is used to judge if the model was able to give the correct response.

References

[1] A. Voronov, L. Wolf, and M. Ryabinin, *Mind your format: Towards consistent evaluation of in-context learning improvements*, 2024. arXiv: 2401.06766 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2401.06766.

[2] 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*, 2023. arXiv: 2302.11382 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2302.11382.

[3] J. He, M. Rungta, D. Koleczek, A. Sekhon, F. X. Wang, and S. Hasan, *Does prompt formatting have any impact on llm performance?* 2024. arXiv: 2411.10541 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2411.10541.

[4] 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*, 2024. arXiv: 2310.11324 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2310.11324.

[5] J. P. Wahle, T. Ruas, Y. Xu, and B. Gipp, "Paraphrase types elicit prompt engineering capabilities," *arXiv preprint arXiv:2406.19898*, 2024.

[6] Y. Wang, S. Mishra, P. Alipoormolabashi, Y. Kordi, A. Mirzaei, A. Arunkumar, A. Ashok, A. S. Dhanasekaran, A. Naik, D. Stap, *et al.*, "Super-naturalinstructions:generalization via declarative instructions on 1600+ tasks," in *EMNLP*, 2022.

[7] H. Wei, S. He, T. Xia, A. Wong, J. Lin, and M. Han, *Systematic evaluation of llm-as-a-judge in llm alignment tasks: Explainable metrics and diverse prompt templates*, 2024. arXiv: 2408.13006 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2408.13006.

[8] V. Wang, M. J. Q. Zhang, and E. Choi, *Improving llm-as-a-judge inference with the judgment distribution*, 2025. arXiv: 2503.03064 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2503.03064.

Team Responsibilities

During the project multiple team members contributed to different parts of the project. The following list summarizes the contributions of each team member but is not complete as some parts were done jointly or overlapped.

- Tobias Leibrock:
  - Setting up the project and GitHub repository
  - Initial reproduction of results and formatting classes
  - Setup `REFORMAT` package
  - Implement full functionality for format mode
  - Implement full functionality for LLM judge mode
  - Find expert rules for models and add them to the `REFORMAT` package
  - Setup ShareLaTeX project and outline of report
  - Initial draft of poster & final touches and restructuring
  - Repository and `REFORMAT` README files
  - Report Abstract, Introduction, Results, Discussion
- Pontus Amgren:
  - All thingd synonym related
  - Implemented synonym rules in `REFORMAT` package
  - Early work on prompt format mutation
  - Initial work on prompt template class
  - Considerable work on poster
  - Report Methodology, Synonym Results (Section 3.2)
- Gabriel Trannoy:
  - Assist in reproduction and analysis of results
  - Contribution on the poster
  - Report Outlook, Conclusion