Rewriting the input prompt by an LLM to improve output quality

Sina Hashemi
sinaHashemi@ut.ac.ir

Vahid Moradi
moradilak@ut.ac.ir

1 Problem statement

In recent years, large language models (LLMs) have demonstrated impressive capabilities across a wide range of tasks, from question answering to generating code and assisting in scientific research. However, the quality of their output is often dependent on the quality of the input prompts. Slight variations in wording, structure, or clarity can lead to vastly different responses. This creates a significant challenge: users may not always know how to formulate the most effective prompt to get accurate and relevant results.

Recent advances in prompting techniques (e.g. chain-of-thought prompting) have shown that how a question is presented to an LLM can significantly affect its ability to reason through the answer. Even so, users may not always know how to craft the best prompt for a given task. **This research proposes an automated solution: use one LLM to rewrite a user's input prompt into a potentially more effective prompt**, then feed this rewritten prompt to another LLM (or the same LLM) to generate the final answer. The main idea is that the rewriting LLM model can act as an informed "intermediary," interpreting the user's intent and providing clarity or structure to help the answering LLM model understand better, thereby enabling the answering model to perform more effectively. In fact, the goal is to use a model for prompt rewriting and train it in a way that aligns with the answering model, so it can better understand the target model's language and rewrite the user's prompt in a manner that suits the answering model's way of understanding. This approach leverages the generative and interpretive abilities of LLMs themselves to improve prompting, rather than relying solely on human engineering of prompts.

We will focus on tasks that are known to be challenging for LLMs without careful prompts, such as multi-step reasoning problems, mathematical word problems, and implicit reasoning questions. In these cases, a prompt rewriter might, for example, break a complex question into simpler parts, fill in assumed context, or otherwise reformulate the query in a way that makes the solution more reachable by the answering model. A key question we will address is how to instruct the rewriting LLM to produce an improved prompt without changing the original query's intent or introducing unwanted bias. We will also explore how much improvement can be gained with prompt rewriting when using various types of LLMs as the main respondent.

The problem of the effect of rewriting input prompts on the performance and accuracy of models is to be worked on. Our motivation is to reduce costs and save time while achieving better accuracy. The interesting thing is that the user prompts are going to be rewritten by the model and the user does not interfere in the process. In terms of improving the accuracy of the models after rewriting the input prompts, it can also be said that it is of scientific importance.

The practical importance of this work lies in its potential to improve the user experience with LLM, streamline workflows, and improve the quality of LLM-generated content in various domains. Standardizing terminology, removing ambiguities, respecting input-length constraints, and explicitly guiding the model's reasoning can lead to more predictable and desirable outcomes, making LLMs more accessible, effective for a wider range of users and applications, and increasing their accuracy.

2 What you proposed vs. what you accomplished

What we proposed to:

- Select and preprocess dataset
- Train (Llama-3.2) on selected dataset using the PPO method as rewriter model for Llama-3.2 as answering model model
- Evaluating the performance of the model
  - Evaluating on 3 answering model
    * without rewriting prompt
    * with rewriting prompt before PPO training
    * with rewriting prompt after PPO training

What we accomplished to:

- We accomplished everything we wanted to do, except that we changed the training method from PPO to GRPO. Reason:
  - **Better fit for goal:** GRPO aligned more with our aim of finding the best style and expression for the answering model by generating multiple outputs and scoring them via a reward function.
  - **Memory efficiency:** GRPO avoided using a value model, making it more memory-efficient.
  - **Rule-based preference training:** We used a rule-based reward function to train the prompt-rewriting model on our preferences, and GRPO suited this setup better.

3 Related work

Prompt engineering has emerged as a key technique for enhancing the performance of LLMs. Recent studies have explored various methods for optimizing the prompts to improve the model output without modifying the underlying model parameters, our work draws inspiration from several threads of research in prompt engineering and meta-optimization of prompts using LLMs. That Some of which are mentioned below.

Recently, Srivastava et al. (2024) proposed PRoMPTed, an approach where an LLM is used in the loop to rewrite each test instance's prompt in a zero-shot setting. In this approach, a new method called "PRoMPTed" has been introduced to improve the answers of language models, in which language models are in a loop in such a way that, using an auxiliary model (Meta LLM), it rewrites and modifies the question based on the initial output of the main model, so that the main model gives a better answer. That is, the second model looks at the answer of the first model and tries to change the question in such a way that the first model gives a better answer. The results of this method have shown that the rewritten prompts by the auxiliary model (GPT-3.5), which can be weaker than the main model (GPT-4), provide higher accuracy.

Another relevant study is by Li et al. (2024), who focused on personalized text generation and introduced a system to learn prompt rewriting policies. They trained a smaller rewriter model through a combination of supervised learning and reinforcement learning to automatically revise user prompts, aiming to incorporate personal context into the prompt. Their results on writing tasks showed that the rewritten prompts produced by the learned rewriter led to outputs that were preferred over those from the original prompts.

In the context of information retrieval augmented QA, Ma et al. (2023) explored query rewriting for LLMs. They introduced a "Rewrite-Retrieve-Read" pipeline where an LLM rewriter first reformulates the user's query to be more amenable for document retrieval, then the retrieved information is used to answer the query. Their experiments on open-domain QA tasks showed that a learned rewriter (including one implemented by a smaller LLM) can significantly boost the accuracy of a frozen reader LLM by crafting better search queries.

Our approach is also related to efforts in multi-turn conversational systems. Sarkar et al. (2025) present one of the first studies of using LLMs to mediate human-LLM chats by rewriting user queries in situ; in which, an LLM chatbot is used to get a prompt from the user and an LLM rewriter is used to rewrite the prompt. The general idea is that another language model (the rewriter) reads the user's question and rewrites it if necessary. In this way, the chatbot better understands what the user wants. The beauty of this approach is that this rewriter model looks at the conversation history, in addition to the question itself, to understand exactly what the user is looking for.

In another article (Kong et al., 2024), the PRewrite method was introduced, which has a simple idea and first tells LLM itself to rewrite the user's prompt, then gives the user the answer. the point of this method is that it requires the use of a very very large language model.

In one another article (Cheng et al., 2024), the authors decided to train the model using Element, using a robust model (GPT-4) to rewrite the prompts and produce a good dataset.

Overall, these works suggest a growing interest in LLM-based prompt optimization. Unlike traditional prompt engineering which relies on human intuition or brute-force search over prompt variations, using LLMs themselves to improve prompts offers a dynamic and automated strategy. Our project builds upon this foundation by developing a system that automatically rewrites user input prompts to produce more optimized versions. By standardizing terminology, removing ambiguities, and aligning prompts with model input constraints, our approach aims to improve the responsiveness and reliability of LLMs across various tasks.

In other words, our contribution will extend this literature by systematically evaluating prompt rewriting on explicit reasoning benchmarks and by comparing different model pairings for the rewriter-answerer roles. We also intend to contribute insights on the kinds of prompt transformations that are most helpful (e.g., adding step-by-step structure, clarifying ambiguous references, etc.), which could inform future prompt engineering guidelines. In summary, we build on the idea that prompts can be treated as another output to be optimized, with LLMs as the optimizers.

4 Your dataset

In this project, we plan to use the **GSM8K**(Cobbe et al., 2021) dataset, which is an open source dataset, and in the next phases, a dataset will be created from it that has an additional column called the rewritten prompt. The reason for using this dataset was that since the math questions have been answered and each question has a unique answer, it is easier to reward the model and complete its training.

So, we will primarily evaluate our approach on benchmarks that require reasoning, as these are expected to benefit most from better prompts. Our starting point is the GSM8K dataset, a collection of 8.79K grade-school math word problems that demand multi-step arithmetic reasoning. GSM8K is a standard test for LLM reasoning ability, and its problems come with detailed solutions, enabling automatic checking of answer correctness. We anticipate that many GSM8K questions (which are written in plain language) could be rephrased to highlight relevant details or logical structure, helping the main LLM avoid mistakes.

For consistency, prompts in the dataset will be posed to the models in a zero-shot manner (no task-specific examples given in the context), since our focus is on zero-shot prompt effectiveness. In the rewritten prompt condition, the original query will first be transformed by the rewriter LLM (with any task-specific instructions, like "let's think step by step" if we choose to include them), and the resulting prompt will be what the main LLM sees. The baseline condition will use the original query directly with the main LLM. By comparing accuracy and other metrics between these conditions, we will quantify the improvement attributable to prompt rewriting.

4.1 Data preprocessing

In this section, we first randomly selected 50 data points for training and 50 data points for evaluation. Ideally, the number of training data points should have been higher, but due to limited resources and existing issues, we used fewer data points for training.

For evaluating the answering model's responses—where our expectation was to obtain the final answer as a single number—we needed to extract only the numerical answer from the answer field in the dataset, and we wrote a function for this purpose.

5 Baselines

We evaluate two simple, reproducible baselines that use only off-the-shelf models and deterministic decoding. Our method—*GRPO-Trained LLM Rewriter (Ours)*—is **not** counted as a baseline and is described elsewhere.

(B1) Direct solve (no rewriting)

We pass each GSM8K question directly to a target solver model and extract the numeric answer. Concretely, we use publicly available, 4-bit quantized instruct models as the solvers:

- **Llama-3.2-3B-Instruct** (`bnb-4bit`)

- **Qwen2.5-3B-Instruct** (`bnb-4bit`)
- **Gemma-3-4B-It** (`bnb-4bit`)

We wrap the question with a lightweight instruction that forces a single numeric answer between `<answer>...</answer>` tags (function `gsm8k_prompt`). The solver then generates with *greedy decoding* (`do_sample = False`) and a generous `max_new_tokens` budget (set to 1024 in evaluation code). The predicted number is parsed by `extract_gen_answer` and compared with the ground-truth numeric target parsed from GSM8K's `####` line via `extract_gt_answer` (exact match on the numeric string). We also log total *input tokens* consumed per example.

(B2) Static LLM rewriter + solve

Here we interpose a *generic* rewriter (no RL, no task-specific fine-tuning) in front of the same solvers above. The rewriter is **Llama-3.2-3B-Instruct** (`bnb-4bit`) prompted with a short system message to "output only the rewritten question" in a direct style, with an optional domain hint "math word problem." Rewriting uses low sampling `temperature = 0.1` (near-deterministic) and `max_tokens = 1024` for the rewrite itself. The rewritten question is then sent to the solver using the same `gsm8k_prompt` and greedy decoding as in (B1). We again parse the final numeric answer and track input token counts. This baseline probes whether generic paraphrasing without any learning signal helps or hurts downstream solving.

Why these baselines?

(B1) is the minimum, most interpretable comparator that captures the raw capability of the target solver. (B2) isolates the effect of *rewriting as a mechanism* without learning: if (B2) changes accuracy or cost relative to (B1), those changes are attributable to paraphrase/formatting rather than optimization. We hypothesize that the method is model-specific and test this by applying it to three diverse model families (Llama, Qwen, and Gemma).

Implementation details common to both

All models are loaded through Unsloth with 4-bit quantization for speed/memory. We standardize tokenizer behavior by applying a Llama-3.1 chat template when missing (just to be safe), setting *left padding*, and adding a `<|pad|>` token if absent (pad id propagated to the model and generation config). Decoding is greedy for solvers to avoid variance. Randomness is controlled with `seed = 42` (NumPy and PyTorch).

Hyperparameters and tuning policy

We did **not** tune any baseline hyperparameters on the evaluation set. Decisions were fixed *a priori* for determinism and adequacy:

- **Solvers:** `do_sample = False`, `max_new_tokens = 1024`, default repetition penalties (unchanged), early stop on EOS.
- **Rewriter (B2):** `temperature = 0.1`, `max_tokens = 1024`, style = "direct," domain hint = "math word problem."

No LoRA/adapter training is used for either baseline.

Train/validation/test split

We load GSM8K (`openai/gsm8k`, branch `main`) and create a fixed, disjoint partition with a single call in code:

- **Test (evaluation) set:** first **50** examples after shuffling with `seed = 42`.
- **Train set for *Ours***: next **50** examples (used only to train the GRPO rewriter; not used to fit or tune baselines).

A separate validation set is not used; baseline hyperparameters were fixed and not tuned.

Metrics and reporting

The primary metric is **exact-match accuracy** on the final numeric answer. We additionally report **average input tokens** (prompt length) to quantify the cost impact of rewriting. All results are computed per solver and per baseline setting (with/without rewriting). Exact numbers for Llama, Qwen and Gemma appear in the results table in the main text; qualitatively, Based on the obtained results, evaluating the answering model with (B2) for the LLaMA model improved the outcome compared to (B1). In contrast, for the other two models, the opposite occurred. However, for all models, using (B2) reduced the number of input tokens.

6 Your approach

Imagine there are three people named A, B, and C. Person A is a friend of person B, and because of the frequent interactions between A and B, A knows B's behavior well and understands how to express concepts or requests so that B can understand A's words better and respond more appropriately. On the other hand, person C is a stranger to both of them, and A does not know how best to interact with C.

Now the question is: Can this greater familiarity between A and B lead to requests made to model B producing better results and outputs compared to person C with the same abilities? And, if the same principles and behavioral preferences used with person B are applied to person C (this time, C has a different personality from B), will they lead to the same results?

In our approach, we want to use an auxiliary model (rewriter model) to rewrite user prompts for the question answering task based on the responding model's preferences and in order to enable it to respond more accurately, in such a way that the initial prompt is given to the main model (responding model) and the final answer is obtained.Figure 1

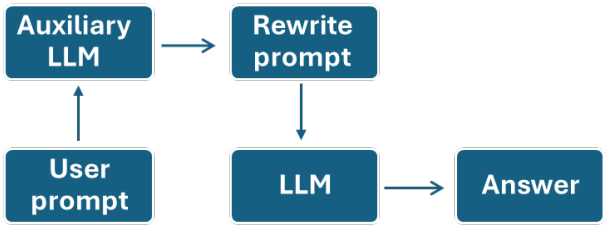

Figure 1: Roadmap

The idea is explained in detail below.

Idea. We train a *prompt rewriter* a quantized small instruction-tuned LLM with LoRA adapters via GRPO (Generative RL with Preference Optimization). The rewriter transforms a user math word problem into a format that a *frozen responder model* can solve more accurately and with fewer tokens. Learning is driven by a scalar reward combining responder accuracy, semantic fidelity to the original question, and a soft length target.

6.1 How it works

1. **Rewrite.** The rewriter receives the original question and produces a rewritten prompt tailored to the responder.

2. **Answer.** The frozen responder is prompted with a standardized (GSM8K-style) instruction that *must* return its final numeric answer inside `<answer>...</answer>` tags.

3. **Reward.** We compute a scalar reward for each rewrite as:

$$R = w_{\text{acc}} \cdot R_{\text{acc}} + w_{\text{sem}} \cdot \text{BERTScoreF1} - w_{\text{len}} \cdot \frac{|\ell - \ell^*|}{\ell^*},$$

where $R_{\text{acc}} \in \{-1, +1\}$ is exact-match (EM) correctness on the parsed numeric answer, ℓ is the rewrite length, and ℓ^* is a target length.[1] The preferences we considered for designing the reward function were threefold, the most important being that the prompt rewriting model should rewrite the prompt in a way that makes the responding model produce the correct answer in its output. Therefore, we gave this aspect the highest weight.

We also wanted to ensure that the meaning of the question would not change, so we used BERTScore to assign a similarity score, but we kept its weight relatively low to avoid negatively affecting quality too much.

In addition, to reduce costs as much as possible, we aimed to minimize the input to the responding model by controlling the length of the rewritten questions so they would not become unnecessarily long.

4. **GRPO update.** With six variations of the question (`num_generations=6` sampled rewrites per prompt) with temperature of 0.6, rewards are evaluated and the rewriter's LoRA parameters are updated via `GRPOTrainer`.

Answer parsing & metrics. We extract the responder's numeric output from `<answer>...</answer>` and compute EM on the trimmed numeric string. We also track tokens (prompt or rewritten generation) as an efficiency proxy.

6.2 Implementation status

End-to-end training and evaluation are **working**. The notebook trains LoRA via GRPO and reports EM and token usage with and without rewriting.

[1] Defaults used in code: $w_{\text{acc}} = 2.0$, $w_{\text{sem}} = 0.3$, $w_{\text{len}} = 0.1$, $\ell^* = 60$.

6.3 Libraries

- **unsloth** (4-bit loading, LoRA, chat templates).
- **trl** (`GRPOConfig`, `GRPOTrainer`;).
- **transformers**, **torch**, **datasets**, **numpy/pandas**, **tqdm**.
- **bert_score** (semantic fidelity).

6.4 What we implemented (and where)

All code lives in the uploaded notebook `gsm8k-GRPO.ipynb`. Key components:

- Reward: `compute_reward(...)` (accuracy, BERTScore-F1, length penalty) and `grpo_reward_fn(...)` adapter for `GRPOTrainer`.
- Prompting: `gsm8k_prompt(...)`, `system_prompt(...)`, `create_prompts(...)`.
- Evaluation: `evaluate_prompts(...)`, extraction helpers `extract_gt_answer(...)`, `extract_gen_answer(...)`.
- Model I/O: `load_model_and_tokenizer(...)` with chat-template fix, left padding, and explicit pad token.

6.5 Models and training setup

- **Backbones:** primary `unsloth/Llama-3.2-3B-Instruct-bnb-4bit` (used as frozen responder and trainable rewriter with LoRA). Hooks for `unsloth/Qwen2.5-3B-Instruct-bnb-4bit` and `unsloth/gemma-3-4b-it-unsloth-bnb-4bit`.
- **LoRA (rewriter):** r=64, α=64, dropout 0, bias `"none"`, target modules {`q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, `down_proj`}, gradient checkpointing `"unsloth"`.
- **Tokenizer/formatting:** Apply Llama-3.1 chat template if missing; *left* padding; explicit `"<|pad|>"` (avoid using EOS as PAD).

Table 1: Accuracy (Exact Match) on GSM8K mini test set ($n = 50$). The prompt rewriting model is model Llama-3.2-3B, and it has been trained based on the preferences of model Llama-3.2-3B.

Base Solver	No-Rewrite	Untrained Rewriter	Trained Rewriter
Llama-3.2-3B	0.380	0.420	0.440
Qwen-2.5-3B	0.500	0.360	0.320
Gemma-3-4B	0.700	0.560	0.600

Table 2: Total prompt-time tokens (lower is better). The prompt rewriting model is model Llama-3.2-3B, and it has been trained based on the preferences of model Llama-3.2-3B.

Base Solver	No-Rewrite	Untrained Rewriter	Trained Rewriter
Llama-3.2-3B	3055	2141	2168
Qwen-2.5-3B	3120	2214	2245
Gemma-3-4B	3141	2273	2304

- **GRPO config (key):** `learning_rate=5e-6`, `adam_beta1=0.9`, `adam_beta2=0.99`, `weight_decay=0.1`, `warmup_ratio=0.1`, linear scheduler; `optim="paged_adamw_8bit"`, `max_grad_norm=0.1`; `num_generations=6`; `per_device_train_batch_size=6` (*must be a multiple of* `num_generations`); `grad_acc_steps=1`; `max_prompt_length=256`; `max_completion_length=1792` (for a 2048 context); `num_train_epochs=1`; `logging_steps=1`.
- **Responder decoding (eval):** `do_sample=false`, max new tokens ≤ 1024, strict `<answer>` format.

6.6 Results

Table 1 reports EM on the 50 held-out items and Table 2 reports the approximate token budget counted at prompt time (lower is better). Numbers are taken directly from the notebook's runs.

On Llama-3B, rewriting helps even without RL (+4 points) and improves further with GRPO (+6 points over no-rewrite) while reducing the prompt token budget by 30%. On Qwen-3B and Gemma-4B, the rewriter trained against Llama's solver does not transfer perfectly: GRPO still improves over the untrained rewriter but can be below each model's strong no-rewrite baseline. This suggests the learned rewriting policy is partially solver-specific. Training with multiple solvers or

a solver-aware objective is a promising direction.

7 Error analysis

Our approach. We train a prompt *rewriter* with GRPO to optimize a composite reward that balances downstream solver correctness (exact match on the numeric answer), semantic fidelity to the original question, and brevity. At inference, the pipeline is: (i) rewrite the user problem under strict format constraints (final answer must appear as `<answer>...</answer>`), then (ii) pass the rewrite to the same small solver used for baselines.

Small-batch evaluation. On a 5-item sample, our approach achieved 1/5 exact-match (20%). The single correct case (Q1) preserves ratios and totals and outputs `<answer>109</answer>`. The four errors expose concrete failure modes:

- **Q2 (format/spec mismatch):** The rewrite asked for *both* large and small buttons, and the solver returned a tuple-like string without `<answer>...</answer>`, so no numeric EM was recorded.
- **Q3 (reading/arithmetic slip):** The solver misread "Indras has 6 letters" as 5 and propagated the error, yielding 12 instead of the ground truth 13.
- **Q4 (information loss in rewrite):** The rewrite collapsed the multi-person relations to a bare comparison, removing the numeric chain needed to compute the difference; the solver then asked for more information.
- **Q5 (semantic distortion):** The rewrite converted per child joins (multiplicative growth) into fixed additive increments, producing 15 instead of 25.

8 Contributions of group members

In this project, an attempt was made to carry out all the segments simultaneously in the laboratory and to the extent possible, not to work separately. However, considering the circumstances in the country that had led to changes in the project process, the following division can be made based on the percentage of progress of the different segments, in which the major contribution of each group member has been identified:

- segment 1: Idea by **Sina Hashemi**
- segment 2: Research and investigation for starting and carrying out the work — done **collaboratively**
- segment 3: did data selection, data preprocessing , part of the evaluation — done by **Vahid Moradi**
- segment 4: built and trained models and part of the evaluation — done **Sina Hashemi**
- segment 5: analyze the output of the model, error analysis and annotations — done by **collaboratively**
- segment 6: conclusion and reports writing — done **collaboratively**

9 Conclusion and Future Work

In this project, we have outlined a method to improve LLM output quality by interposing an LLM-based prompt rewriter between the user and the answering model. By having one model rewrite the input prompt, we leverage the language understanding capabilities of LLMs to craft better queries for themselves or other models. This approach addresses a practical pain point: users often do not know the optimal way to ask a question to get a good answer.

We studied whether lightweight prompt rewriting can improve small LLM solvers on GSM8K and established two baselines: (i) sending the original problem to the solver (No-Rewrite) and (ii) using an untrained rewriter to restyle the question (Untrained Rewriter). These baselines reveal that merely restyling the prompt yields mixed outcomes across models. On Llama-3.2-3B, the untrained rewrite modestly improves EM ($0.38 \rightarrow 0.42$) while lowering prompt-time tokens ($3055 \rightarrow 2141$), but on Qwen-2.5-3B and Gemma-3-4B it degrades accuracy relative to No-Rewrite.

For comparison (not as a baseline), we introduced a trained rewriter optimized with GRPO. On Llama-3.2-3B it delivers the best trade-off we observed—EM 0.44 with a substantially smaller token budget (2168 vs. 3055). However, transfer is imperfect: the trained rewriter underperforms the No-Rewrite baseline on Qwen-2.5-3B ($0.50 \rightarrow 0.32$) and falls short of Gemma-3-4B's strong baseline ($0.70 \rightarrow 0.60$), though it remains better than the untrained rewrite on both models. Overall, the results suggest that (a) a naïve, solver-agnostic rewrite is unreliable, and (b) optimization

can help, but the learned policy is partially solver-specific.

We got what we expected from our work.

Limitations. Our prototype uses a small 50/50 mini split drawn from the public test set for rapid iteration and does not include a separate validation set; no hyperparameters were tuned on the held-out 50. Models are 3–4B parameters quantized to 4-bit, and we report single-run numbers without ablation on reward components or decoding strategies. These choices constrain statistical power and external validity.

Future Work. We plan to (i) test this method on newer datasets; (ii) increase the flexibility of the prompt rewriting model and allow it to write any part of the message, including both the system prompt and the user prompt; (iii) using cascades of rewriters (where prompts are refined in stages) or specialized rewriters (where the rewriting LLM is fine-tuned or otherwise optimized for particular domains or user styles); (iv) If we have access to suitable resources, we will use a much larger amount of data for training, with a greater number of generations during training. ; and (v) conduct user studies to determine whether users prefer interacting with a system that automatically refines their questions, potentially improving satisfaction by reducing the need for iterative prompt formulation.

The broader goal is a reusable, compute efficient rewriting policy that reliably improves accuracy *and* reduces tokens across diverse solvers.

References

Cheng, J., Liu, X., Zheng, K., Ke, P., Wang, H., Dong, Y., Tang, J., and Huang, M. (2024). Black-box prompt optimization: Aligning large language models without model training. In Ku, L.-W., Martins, A., and Srikumar, V., editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3201–3219, Bangkok, Thailand. Association for Computational Linguistics.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. (2021). Training verifiers to solve math word problems.

Kong, W., Hombaiah, S., Zhang, M., Mei, Q., and Bendersky, M. (2024). PRewrite: Prompt rewriting with reinforcement learning. In Ku, L.-W., Martins, A., and Srikumar, V., editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 594–601, Bangkok, Thailand. Association for Computational Linguistics.

Li, C., Zhang, M., Mei, Q., Kong, W., and Bendersky, M. (2024). Learning to rewrite prompts for personalized text generation. In *Proceedings of the ACM Web Conference 2024*, WWW '24, page 3367–3378. ACM.

Ma, X., Gong, Y., He, P., Zhao, H., and Duan, N. (2023). Query rewriting for retrieval-augmented large language models.

Sarkar, R., Sarrafzadeh, B., Chandrasekaran, N., Rangan, N., Resnik, P., Yang, L., and Jauhar, S. K. (2025). Conversational user-ai intervention: A study on prompt rewriting for improved llm response generation.

Srivastava, S., Huang, C., Fan, W., and Yao, Z. (2024). Instances need more care: Rewriting prompts for instances with LLMs in the loop yields better zero-shot performance. In Ku, L.-W., Martins, A., and Srikumar, V., editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 6211–6232, Bangkok, Thailand. Association for Computational Linguistics.