constructing the search query.

3 Methodology

We present *Rewrite-Retrieve-Read*, a pipeline that improves the retrieval-augmented LLM from the perspective of query rewriting. Figure 1 shows an overview. This section first introduces the pipeline framework in section 3.1, then the trainable scheme in section 3.2.

3.1 *Rewrite-Retrieve-Read*

A task with retrieval augmentation can be denoted as follows. Given a dataset of a knowledge-intensive task (e.g., open-domain QA), $D = \{(x, y)_i\}, i = 0, 1, 2, \ldots, N$, x (e.g., a question) is the input to the pipeline, y is the expected output (e.g., the correct answer). Our pipeline consists of three steps. (i) Query rewrite: generate a query $\tilde{x}$ for required knowledge based on the original input x. (ii) Retrieve: search for related context, doc. (iii) Read: comprehend the input along with contexts $[doc, x]$ and predict the output $\hat{y}$.

A straightforward but effective method is to ask an LLM to rewrite queries to search for information that is potentially needed. We use a few-shot prompt to encourage the LLM to think, and the output can be none, one or more queries to search.

3.2 Trainable Scheme

Besides, total reliance on a frozen LLM has shown some drawbacks. Reasoning errors or invalid search hinders the performance (Yao et al., 2023; BehnamGhader et al., 2022). On the other hand, retrieved knowledge may sometimes mislead and compromise the language model (Mallen et al., 2022). To better align to the frozen modules, it is feasible to add a trainable model and adapt it by taking the LLM reader feedback as a reward.

Based on our framework, we further propose to utilize a trainable small language model to take over the rewriting step, as is shown in the right part of Figure 1. The trainable model is initialized with the pre-trained T5-large (770M) (Raffel et al., 2020), denoted as *trainable rewriter*, G_θ. The rewriter is first trained on pseudo data to warm up (§3.2.1), then continually trained by reinforcement learning (§3.2.2).

3.2.1 Rewriter Warm-up

The task, query rewriting, is quite different from the pre-training objective of sequence-to-sequence generative models like T5. First, we construct a pseudo dataset for the query rewriting task. Inspired by recent distillation methods (Hsieh et al., 2023; Ho et al., 2022), we prompt the LLM to rewrite the original questions x in the training set and collect the generated queries $\tilde{x}$ as pseudo labels. The collected samples are then filtered: Those that get correct predictions from the LLM reader are selected into the warm-up dataset, denoted as $D_{Train} = \{(x, \tilde{x}) | \hat{y} = y\}$. The rewriter G_θ is fine-tuned on D_{Train} with the standard log-likelihood as the training objective, denoted as

$$\mathcal{L}_{warm} = -\sum_t logp_\theta(\hat{\tilde{x}}_t \mid \tilde{x}_{<t}, x). \quad (1)$$

The rewriter model after warm-up shows modest performance, which depends on the pseudo data quality and rewriter capability. Highly relying on the human-written prompt line, $\tilde{x}$ can be sub-optimal. The relatively small scale of the rewriter size is also a limitation of the performance after the warm-up. Then we turn to reinforcement learning to align the rewriter to the following retriever and LLM reader.

3.2.2 Reinforcement Learning

To further fine-tune the rewriter to cater to the LLM reader, we adopt a policy gradient reinforcement learning framework.

Task Formulation In the context of reinforcement learning, the rewriter optimization is formulated as a Markov Decision Process 5-tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$. (i) The state space $\mathcal{S}$ is a finite set limited by the vocabulary and the sequence length. (ii) The action space $\mathcal{A}$ is equals to the vocabulary. (iii) The transition probability P is determined by the policy network, which is the rewriter model G_θ. (iv) The reward function R gives a reward value that depends on the current state. The policy gradient is derived from rewards, used as the training objective. (v) γ denotes the discount factor. More specifically, the rewriter G_θ after the warm-up is the initial policy model π_0. At each step t, the action a_t is to generate the next token $\hat{\tilde{x}}_t$ based on the observation of the present state, $s_t = [x, \hat{\tilde{x}}_{<t}]$. When the generation is stopped by the End-Of-Sentence token, one episode is ended. After finishing the retrieval and reading, a reward is computed by evaluating the final output, i.e., a score for the LLM reader prediction.

Policy Optimization We adopt Proximal Policy Optimization (PPO) (Schulman et al., 2017), following (Ramamurthy et al., 2022). Maximization

of the expectation of the reward R is formulated as

$$\begin{aligned}
&\max_{\theta} \mathbb{E}_{\hat{\tilde{x}} \sim p_\theta(\cdot|x)}[R(x, \hat{\tilde{x}})], \\
&\max_{\theta} \mathbb{E}_{(s_t,a_t)\sim\pi_{\theta'}}[min\{k_{t,\theta} A^{\theta'}(s_t, a_t); \\
&\qquad \text{clip}\left(k_{t,\theta}, 1-\varepsilon, 1+\varepsilon\right) A^{\theta'}(s_t, a_t)\}], \\
&k_{t,\theta} = \frac{p_\theta(a_t \mid s_t)}{p_{\theta'}(a_t \mid s_t)},
\end{aligned} \tag{2}$$

where θ' is the temporarily fixed policy for sampling and θ is updated. A denotes the advantage function, which is formulated based on the estimation of value network V_ϕ. The value network V_ϕ is initialized from the policy network π_0. The formulation follows Generalized Advantage Estimation (GAE) (Schulman et al., 2015).

$$\begin{aligned}
\delta_t &= R(s_t, a_t) + V_\phi(s_{t+1}) - V_\phi(s_t), \\
\hat{A}_t^\theta(s_t, a_t) &= \sum_{t'=0}^{\infty} \lambda^{t'} \delta_{t+t'},
\end{aligned} \tag{3}$$

where λ is the bias-variance trade-off parameter.

The reward function R reflects the quality of the generated queries, which needs to be consistent with the final evaluation of the task. $\hat{\tilde{x}}$ is fed to the retriever and the reader for a final prediction $\hat{y}$. A part of the reward function is the measures of $\hat{y}$ compared to the golden label y (e.g., exact match and F_1 of the predicted answers), denoted as R_{lm}. Besides, a KL-divergence regularization is added to prevent the model from deviating too far from the initialization (Ramamurthy et al., 2022; Ziegler et al., 2019).

$$R(s_t, a_t) = R_{lm}(\hat{\tilde{x}}, y) - \beta \text{KL}(\pi_\theta \| \pi_0). \tag{4}$$

The final loss function is composed of policy loss and value loss.

$$\begin{aligned}
\mathcal{L}_\theta &= -\frac{1}{|\mathcal{S}|\,T} \sum_{\tau\in\mathcal{S}} \sum_{t=0}^{T} \min(k_{t,\theta} A^{\theta'}, \text{clip}\, A^{\theta'}), \\
\mathcal{L}_\phi &= \frac{1}{|\mathcal{S}|\,T} \sum_{\tau\in\mathcal{S}} \sum_{t=0}^{T} \left(V_\phi(s_t) - R_t\right)^2, \\
\mathcal{L}_{ppo} &= \mathcal{L}_\theta + \lambda_v \mathcal{L}_\phi.
\end{aligned} \tag{5}$$

Here, $\mathcal{S}$ denotes the sampled set, and T is for step numbers.

4 Implementation

Rewriter For the frozen pipeline in §3.1, we prompt an LLM to rewrite the query with few-shot in-context learning (Brown et al., 2020; Min et al., 2022). Our prompt follows the formulation of *[instruction, demonstrations, input]*, where the input is x. The instruction is straightforward and demonstrations are 1-3 random examples from training sets and are kept constant across all runs, mainly for the task-specific output format illustration, i.e., a short phrase as an answer for HotpotQA, and an option as an answer for MMLU. For the training scheme in §3.2, we fine-tuning a T5 as the rewriter.

Retriever We use the Bing search engine as the retriever. It requires no candidate index construction like a dense retriever, nor candidates like a textbook. But it allows for a wide knowledge scope and up-to-time factuality. With Bing API, the retrieval is performed in two approaches. (i) For all retrieved web pages, we concatenate the snippets that are related sentences selected by Bing. This method is similar to using a search engine in a browser, input a query and press Enter, then collect the texts shown on the search result page. (ii) For retrieved web pages, we request the URLs and parser to get all the texts. This is similar to clicking on items on the search result page. Then we use BM25 to keep those with higher relevance scores with the query, reducing the document length.

Reader The reader is a frozen LLM, where we adopt ChatGPT (gpt-3.5-turbo) and Vicuna-13B. It performs reading comprehension and prediction with few-shot in-context learning. In our prompt, following the brief instruction and the demonstrations, the input is x or $[doc, \hat{\tilde{x}}]$ with retrieval augmentation.

It has been proved that both the phrasing of prompt lines (Zhang et al., 2023a) and the selection of demonstrations show effects on the in-context learning performance (Su et al., 2022; Zhang et al., 2023b). As it is not the focus of this work, we pay no more attention to prompt editing.

5 Experiments

5.1 Task Settings

5.1.1 Open-domain QA

Three open-domain QA datasets are used for evaluation. (i) HotPotQA (Yang et al., 2018) consists of complex questions that require multi-hop reasoning. We evaluate the full test set. (ii) AmbigNQ (Min et al., 2020) provides a disambiguated version of Natural Questions (NQ) (Kwiatkowski et al., 2019). For ambiguous questions in NQ, minimal constraints are added to break it into several similar

Direct prompt
Answer the question in the following format, end the answer with '**'. {demonstration} Question: {x} Answer:
Reader prompt in retrieval-augment pipelines
Answer the question in the following format, end the answer with '**'. {demonstration} Question: {doc} {x} Answer:
Prompts for LLM as a frozen rewriter
Open-domain QA: Think step by step to answer this question, and provide search engine queries for knowledge that you need. Split the queries with ';' and end the queries with '**'. {demonstration} Question: {x} Answer: *Multiple choice QA:* Provide a better search query for web search engine to answer the given question, end the queries with '**'. {demonstration} Question: {x} Answer:

Table 1: Prompt lines used for the LLMs.

but specific questions. The first 1000 samples are evaluated in the test set. (iii) PopQA (Mallen et al., 2022) includes long-tail distributions as it contains more low-popularity knowledge than other popular QA tasks. We split the dataset into 13k for training and 714 for testing.

Open-domain QA benchmarks are sets of question-answer pairs denoted as $\{(q, a)_i\}$. We use ChatGPT for both the reader and the frozen rewriter. The evaluation metrics are Exact Match (EM) and F_1 scores. For the reward function in RL, we use an indicator to reward if the retrieved content hits the answer and penalize if misses the answer, denoted as Hit. The total reward is a weighted sum of EM, F_1, and Hit.

$$Hit = \begin{cases} 1 & a \text{ in } doc, \\ -1 & else \end{cases} \tag{6}$$
$$R_{lm} = EM + \lambda_f F_1 + \lambda_h Hit.$$

5.1.2 Multiple-choice QA

For multiple-choice QA, our evaluation is conducted on Massive Multi-task Language Understanding (MMLU) (Hendrycks et al., 2021), an exam question dataset including 4 categories: Humanities, STEM, Social Sciences, and Other. Each category is split into 80% for the training set and 20% for the test set.

Multiple-choice QA can be formulated as $\{(q', a)_i\}$, where $q' = [q, c_0, c_1, c_2, c_3]$. c denotes the options, generally there are four for each question. The retrieved documents that are included in the officially provided contaminated lists are ignored. The questions with options are rewritten into search queries. The answer is one option. EM is reported as metrics and used for the reward.

$$R_{lm} = EM. \tag{7}$$

We use ChatGPT as a frozen rewriter and the reader. We also use Vicuna-13B as the reader for evaluation due to the rate limit issue of ChatGPT. More information on datasets and training setup are presented in the appendix.

5.2 Baselines

The following settings are implemented to evaluate and support our methods. (i) **Direct**: The standard in-context learning without any augmentations. (ii) **Retrieve-then-read**: The standard retrieval-augmented method. Retrieved documents are concatenated with the question. (iii) **LLM as a frozen rewriter**: As is introduced in §3.1, we prompt a frozen LLM to reason and generate queries by few-shot in-context learning. (iv) **Trainable rewriter**: Applying the fine-tuned rewriter, the output queries are used by the retriever and the reader. Table 1 presents prompt line forms. Please note that the prompts for prediction are kept the same for each task.

5.3 Results

Experimental results on open-domain QA are reported in Table 2. For the three datasets, query rewriting consistently brings performance gain with both a frozen rewriter and a trainable rewriter. On AmbigNQ and PopQA, the standard retrieval augments the reader, indicating useful external knowledge is retrieved. On HotpotQA, the standard retrieval hurts the reader. This shows that using complex questions as queries cannot compensate for the parametric knowledge, but bring noises instead (Mallen et al., 2022). This suggests that multi-hop questions are not suitable queries for the web search engine. The scores increase by adding the rewriting step. On PopQA, our trainable rewriter surpasses standard retrieval while being inferior to the LLM rewriter. This indicates that the