Prompt Generate Train (PGT): Few-shot Domain Adaption of Retrieval Augmented Generation Models for Open Book Question-Answering

C. S. Krishna

Microsoft
krishnacs@microsoft.com

Abstract

We propose a framework – Prompt, Generate, Train (PGT) – to efficiently develop a generative question-answering model for open-book question-answering over a proprietary collection of text documents. The framework adapts a retriever augmented generation (RAG) model to the target domain using supervised finetuning and reinforcement learning with synthetic feedback in a few-shot setting. This, we hypothesize, will yield an aligned, uncertainty calibrated model that is competitive with GPT-4 based in-context retrieval augmented generation in generating relevant answers at lower serving costs.

The framework's synthetic generation pipeline will generate synthetic training data comprising <passage, question, answer> tuples using an open-source LLM and a novel consistency filtering scheme. The pipeline will be designed to generate both abstractive and extractive questions that span the entire corpus. The framework proposes to fine-tune a smaller RAG model comprising a dense retriever (Col-BERTv2) and a smaller sized LLM on the synthetic data-set. In parallel, the framework will train a Reward model to score domain grounded answers higher than hallucinated answers using an a priori relevance ordering of synthetically assembled samples. In the next phase, the framework will align the RAG model with the target domain using reinforcement learning (Proximal Policy Optimization). This step may improve the RAG model's ability to generate grounded answers and ignore out of domain questions. In the final phase, the framework will calibrate the model's uncertainty for extractive question-answers.

1 Introduction

A common use-case within enterprise settings is to expose a question-answering service over a proprietary corpus comprising a collection of text documents. The application, in response to the client's question, must be able to generate a factually grounded, attributed response by condensing information from a set of relevant documents in the underlying corpus. If the answer cannot be synthesized from these documents, the system should respond with a "cannot answer" rather than generate a misleading or factually incorrect response (referred to as hallucination in the literature). An additional requirement is to economize development and serving costs by using smaller LLM architectures (<10 Bn parameters).

It has been shown that specialized task and target domain data specific pre-training and/or finetuning enables smaller sized LLMs to outperform in-context learning with LSLMs (Izacard et al., 2022; Hsieh et al., 2023). However, smaller LLMs have inherent weaknesses. (Gudibande et al., 2023) demonstrated that certain properties such as chain of thought reasoning (Wei et al., 2022) only emerge at higher scales. A smaller LLM will therefore struggle to outperform LSLMs such as GPT-4 on questions that require reasoning ability. To address this limitation, the PGT framework needs to include a procedure for model uncertainty calibration. Informally speaking, the model must "know when it knows the answer" and "know when it doesn't know the answer". We make the notion of "knowing when the model knows" and "knowing when it doesn't know" more precise in the section on uncertainty calibration. This is a desirable feature since it enables easier integration into a cascading systems such as FrugalGPT (Chen et al., 2023) where the RAG model's answer can be surfaced only when the model is confident of its answer, else the client's question can be passed on to a human or alternate model.

Further, smaller LLMs are more prone to hallucination [ref?]. The framework design, therefore, needs to address hallucination mitigation. Given this perspective, we list the following design goals for the PGT framework:

1.1 Design Goals

- Few-shot Adaption: The framework needs to adapt the RAG model to the target domain in a few-shot setting. i.e., it must do so without access to a large volume of manually annotated <question, answer> tuples on which the model can be fine-tuned.
- Serving Cost Economization: PGT models must be cheaper to serve and more accurate than systems based on GPT-3.5/4 powered incontext retrieval augmented generation. Since serving cost is a function of model size, we choose models with less than 10 Bn parameters.
- Hallucination Mitigation: The framework should yield a model that only generates answers based on the underlying corpus. For out of domain questions or questions which cannot be answered based on the underlying corpus, the model should be able to generate "cannot answer this question based on given information" rather than hallucinate a baseless answer.
- Uncertainty Calibration: The framework should yield models calibrated for uncertainty, at least for a certain class of questions, such as extractive or yes/no questions. The application layer can then rely on the model's confidence in its generated answer to decide whether to surface answer to the end-user or take a pass on the question.

2 Related Work

The dominant framework to develop models for open-book question answering is based on incontext retrieval augmented generation (Ram et al., 2023)(Bing Search): in response to the user question, a dense or a sparse retriever fetches relevant documents. A prompt is populated by the user's question and the set of fetched documents. This prompt is then fed to an LSLM such as GPT-4 to generate the response. This framework does not require domain adaptation since the LSLM has been extensively pre-trained on public domain corpora such as Wikipedia, Common-Crawl etc. Forcing the LLM to rely on the fetched documents also constrains the LLM to generate grounded answers to a certain extent, although LLMs such as GPT-4 can still be factually wrong in subtle ways [refs?].

While such a framework is suitable for opendomain question services such as search engines, where the service is expected to answer questions from any domain, we argue that it is overkill for closed-domain question answering. In the latter setting, the model is only expected to generate answers for in-domain questions. Hence, it stands to reason that the LLM's knowledge base merely needs to cover the target domain for which a smaller LLM can suffice, motivating the feasibility of adapting a smaller LLM for the target domain.

A popular approach for domain adaptation for question-answering is to train a RAG architecture (Guu et al., 2020; Lewis et al., 2020) comprising a retriever and generator on annotated training data in the form of <question, answer> tuples from the target domain. (Shi et al., 2023), on the other hand, only trains the retriever component of the RAG architecture. (Izacard et al., 2022) showed that jointly fine-tuning the RAG model on as few as 1024 samples from the target domain improves performance relative to GPT-3 based in-context retrieval augmented generation. They also empirically demonstrated that performance increases when the model is fine-tuned on larger sized data-sets.

In most real world settings, such manually crafted data will not be available in sufficient volume or quality. An emerging body of work has adapted dense neural retrievers to the target domain by training on synthetically generated training data comprising <query, relevant passage> pairs (Bergum, 2023; Dai et al., 2022; Saad-Falcon et al., 2023; Abonizio et al., 2023). We extend this line of work to domain adaptation of full-fledged RAG models in a few-shot/zero-shot setting.

3 Methodology

The PGT framework utilizes two LSLMs – GPT-4 and Flan-TF XXL – to generate question answer pairs over all documents in the corpus. GPT-4 is used to generate a seed set of Y samples for extractive and abstractive question-answering formats. The Flan-TF XXL instance leverages the seed set to generate more samples. This design economizes the cost of generating the training dataset. The RAG model is then trained on this dataset, using both supervised fine-tuning and reinforcement learning.

To generate the synthetic data-set with the LSLMs, we split documents into document segments such that the document segment size is

bounded by the permissible context window size for the LSLM (1024-2048 tokens). We create an index I_1 over these document segments to facilitate random retrieval of a segment. Similarly, we maintain another index I_2 over document segments where the document segment size is upper bounded by the context window size of the RAG model's generator.

3.1 PGT Steps

The PGT framework consists of the following steps:

- 1. Phase 1a (optional): Adapt the retriever component of the RAG model to the domain by training on the Inverse Cloze Task (Khattab et al., 2021; Lee et al., 2019).
- Phase 1b: Synthetic Training Data Generation pipeline: In this phase, we generate two sets of data to fine-tune the RAG model and the Reward model.
- 3. Phase 2: This comprises 2 training procedures that can be executed in parallel.
 - (a) RAG model supervised fine-tuning
 - (b) Reward model training
- 4. Phase 3: Use of reinforcement learning (PPO) to align the RAG model to the target domain, with the Reward model generating a reward score for relevance.
- Further fine-tuning of the RAG model for uncertainty calibration for certain classes of questions.

We can repeat steps 2, 3 and 4 for a few iterations.

4 PGT Components

4.1 RAG-Retriever

We choose ColBERTv2 (Khattab and Zaharia, 2020) as the dense retriever in the RAG model. Dense retrievers work best with smaller sized documents of not more than 400 tokens. On the other hand, feeding the generator with chunks rather than whole documents can lead to context fragmentation and degrade the generated answer's quality. To resolve this trade-off, we modify the retrieval procedure as follows: we maintain a third index I_3 in which each document segment from I_2 is split into smaller sized, mostly disjoint chunks of token size ~ 300 tokens. The retriever component of

the RAG model indexes into I_3 to fetch relevant chunks.

For a given chunk c of a document and the question q we can compute the similarity score S(q,c) using the ColBERTv2 model instance. Let a document segment $d \in I_2$ correspond to chunks $(c_1,..,c_n)$ indexed in I_3 . We define the probability of fetching d given a question q to be proportional to the maximum similarity score over all the document chunks:

$$P_{\eta}(d|q) \propto exp(max(S(q,c_1),...,S(q,c_n)))$$
 (1)

Here, η refers to the tunable parameters of the retriever. This design and retrieval mechanism, we hypothesize a) ensures tighter coupling between the retriever and the generator, and b) mitigates context fragmentation, thereby improving retrieval and generation quality. We can optionally add a lexical similarity signal based on BM25 (Ma et al., 2020) to the similarity score.

4.2 RAG Generator

We choose a pre-trained instance of the Flan-T5 encoder-decoder architecture for answer generation. The encoder encodes the question and passages fetched by the retriever. The decoder, conditioned on the encoding, autoregressively computes the likelihood of generating the answer as

$$P_{\phi}(a|q,d) = \prod_{i} P_{\phi}(a_{i}|q,d,a_{< i})$$
 (2)

4.3 Reward model

We also initialize a Reward model using a pretrained BERT instance (Devlin et al., 2018). The Reward model is trained to generate a relevance score, given the passage, question, and answer.

5 Synthetic Data Generation

5.1 Seeding with GPT-4

In this phase, we prompt GPT-4 to generate two sets of Y <passage, question, answer> tuples across the following question formats: extractive (EX) and abstractive (AB) (Khashabi et al., 2020).

We randomly sample document segments using I_1 . We try to preserve documents boundaries as much as possible so that the LLM has a coherent passage as the basis for generating a question-answer pair. We append a q-a format specific prompts (Appendix-I) for each of the q-a formats.

This prompt along with the passage is fed to GPT-4 to generate the corresponding answer. We use the seeding set of Y <passage, question, answer> pairs in turn to prompt Flan-TF XXL to generate Z question answer pairs across both formats.

5.2 Generate non-matching question answer pairs

It is important to train the model to generate a "can't answer based on given references" response as well if the fetched passages cannot generate the required answer. To this end, we also fetch the top K' matching chunks using the retriever, remove the chunks that were used to generate the answer, and assemble a non-matching passage, p'_c . We then prompt the LSLM to generate a rationale r_c for why the given question q_c cannot be answered given context p'_c :

$$a'_c \leftarrow LSLM(p'_c, q_c)$$

We assemble a non-matching question-answer pair for every matching question answer pair (p_c', q_c, a_c') which we refer to as a non-matching tuple. During training, the ratio of number of non-matching to matching question-answer pairs is a design parameter that can be set based on down-stream requirements – intuitively, if we want the model to err on the side of caution, we should include as many non-matching question answer pairs as matching question answer pairs.

5.3 Generation with Flan-T5 XXL

In this phase, we tap Flan-T5 XXL to generate more training data, using the seed set from the previous phase to sample demonstration exemplars.

- 1. Start with the synthetic training data-set *T* populated via GPT-4.
- 2. Concatenate a prefix prompt P consisting of N passage-question-answer tuples sampled without replacement from T along with the meta prompt:

$$P = [(p_1, q_1, a_1)...(p_N, q_N, a_N)]$$

3. Select a candidate passage p_c at random (alternatively, based on ideas presented in (?) from I_1 and prompt Flan-T5 XXL to generate a candidate question-answer pair:

$$(q_c, a_c) = FlanT5XXL([P; p_c])$$

5.4 Consistency Filtering

We add (p_c, q_c, a_c) to T if it meets the following consistency filtering criteria:

- 1. For the generated question q_c the domain adapted retriever's top K fetched documents should span the passage p_c . If not, drop this tuple else proceed to the next step.
- 2. We again prompt Flan-T5 XXL/GPT-4 to generate an answer a_c' to the question q_c based on the top K fetched chunks from the previous step. Retain this sample only if there is a high semantic overlap between a_c and a_c'
- Confidence based threshold (Abonizio et al., 2023): the normalized log-probability of generating a question-answer pair for the candidate passage exceeds a threshold.
- 4. Uncertainty based threshold (optional): Use a suitable measure of the uncertainty (Lin et al., 2023) of the generated sample. Accept the sample only if the uncertainty measure is less than a threshold.
- 5. If the sample passes the consistency filtering steps, add it to T

6 Phase 2a: RAG model Supervised Finetuning

We present a new log-likelihood function, *in-context RAG-token model likelihood*, that combines RAG-token model likelihood (Lewis et al., 2020) with in-context RALM learning (Ram et al., 2023):

$$P_{\eta,\phi}(a|q;K,L,S) = \prod_{i=0}^{n_S-1} \prod_{j=1}^{S} \sum_{k \in topK} P_{\eta}(d_{i,k}|[q;a_{Si}^L])$$

$$P_{\phi}(a_{Si+j}|[q;d_{i,k};a_{<(Si+j)}])$$
(3)

The design parameter S is the stride size which determines after how many steps we refresh the context by fetching relevant documents conditioned on the question and a subset of the answer prefix. The design parameter L decides how many of the most recently generated tokens in the prefix to consider for context augmentation. a_{Si}^L refers to the L most recent tokens in the answer from the Si'th position and going backwards. K decides how many documents to marginalize over in generating the next token at each step. $d_{(i,k)}$ refers to

the k'th document-segment fetched by the retriever via I_2 in the i'th stride. $n_S = n/S$ determines the number of fetches during retrieval, where n is the token length of the answer. Note that when $S=1,\,L=0$, this function degenerates into the RAG-token model likelihood function.

We intuit that the in-context RAG-token model likelihood provides the flexibility for the retriever to fetch the right document segments conditioned on the evolving answer. This in turn improves the quality of the conditioned generation. By marginalizing over top K documents during training at every generation step, we improve the ability of the retriever to discern relevant from irrelevant documents.

6.1 Generation Procedure

The transition probability associated with generating the next token in the answer is given by:

$$P_{\eta,\phi} \left(a_{Si+j} \right] | [q; a_{<(Si+j)}; K, L, S) = \sum_{k \in topK} P_{\eta} \left(d_{i,k} | [q; a_{Si}^{L}] \right) P_{\phi} \left(a_{Si+j} | [q; d_{i,k}; a_{<(Si+j)}] \right)$$
(4)

This can be plugged into a standard beam decoder to generate the answer.

7 RAG Alignment Training

Reinforcement learning with human feedback (RLHF) training (Ouyang et al., 2022; Kadavath et al., 2022) further adjusts model parameters so as to generate answers that are aligned to human preferences. However, this procedure requires human preference feedback on answers, which may not always be available. Our goal is limited to aligning the RAG model such that its answers are grounded in the underlying corpus. Towards this end, we adapt RLHF but without recourse to human preference feedback to design a new technique – Reinforcement Learning with Synthetic Feedback (RLSF).

7.1 Phase 2b: Reward model Training

Let $S((p_c, q_c, a_c)) \in \mathbb{R}$ be a measure of the relevance of the response, given the context and the question. The relevance score should be low if the answer is hallucinated, factually incorrect or not grounded in the underlying context, and high otherwise. We want to train a Reward model that can estimate the relevance of a model's generated response, given the context and question.

We set up the training data-set for Reward model training as follows. For every pair of matching and non-matching tuples, (p_c, q_c, a_c) , (p_c', q_c, a_c') , we can assemble additional tuples, (p_c', q_c, a_c) , (p_c, q_c, a_c') to assemble a composite tuple:

$$\tau = ((p_c, q_c, a_c) (p'_c, q_c, a'_c) (p'_c, q_c, a_c) (p_c, q_c, a'_c))$$

We assemble an alternate data-set of such tuples $T^* = \{\tau^1,...,\tau^N\}$ for training the Reward model. For a given τ , we fix the following orderings based on relevance of the answer, given the question and context:

$$S((p_c, q_c, a_c)) > S((p'_c, q_c, a_c))$$

$$S((p_c, q_c, a_c)) > S((p'_c, q_c, a'_c))$$

$$S((p'_c, q_c, a'_c)) > S((p'_c, q_c, a_c))$$

$$S((p'_c, q_c, a'_c)) > S((p_c, q_c, a_c))$$
(5)

We train the Reward model by minimizing the following contrastive loss function:

$$loss(\theta) = -\frac{1}{4} \mathbb{E}_{\tau \sim T^*} \left[\log \left(\sigma \left(RM_{\theta} \left(\tau_1 \right) - RM_{\theta} \left(\tau_3 \right) \right) \right) + \log \left(\sigma \left(RM_{\theta} \left(\tau_1 \right) - RM_{\theta} \left(\tau_4 \right) \right) \right) + \log \left(\sigma \left(RM_{\theta} \left(\tau_2 \right) - RM_{\theta} \left(\tau_3 \right) \right) \right) + \log \left(\sigma \left(RM_{\theta} \left(\tau_2 \right) - RM_{\theta} \left(\tau_1 \right) \right) \right) \right]$$

$$(6)$$

7.2 Phase 3: Alignment using Reinforcement Learning (PPO)

We use proximal policy optimization (PPO) (Schulman et al., 2017) to further finetune the student LLM with the Reward model providing the reward signal for relevance of the answer as follows: We sample a passage-question pair using the question-generation pipeline from Phase I or optionally sample a tuple from T to yield a passage-question tuple (p,q). The RAG model then generates the answer conditioned on the passage, using Equation 4 with K=1. The Reward model scores the answer for relevance. We then finetune the RAG model w.r.t parameters ϕ of the generator by minimizing the PPO objective:

$$loss(\phi) = RM_{\theta}((p, q, a))$$
$$-\beta \log \left(\frac{P_{\eta, \phi}(a|q, p)}{P_{\eta, \phi'}(a|q, p)} \right)$$
(7)

8 Phase 4: Uncertainty Calibration

We want the student LLM to be calibrated for uncertainty. Informally, the model should "know when it knows the answer" and "know when it doesn't know the answer". We make this precise based on the definition of calibration outlined in (Guo et al., 2017). Let $p_M(a|q,p)$ be the probability assigned by the model that the answer it generated given the question and context is the correct response. Then, the model is perfectly calibrated if:

$$P(a|p_m = p) = p, \forall p \in [0, 1]$$

To calibrate the model, we train the model for predicting whether the answer it generated given the question and supporting evidence is correct or wrong. We do so by maximizing the "indirect logit" (Lin et al., 2022), the log-probability associated with the model predicting "correct" or "wrong" for it's own answer, given the question and supporting evidence from the corpus.

We present the recipe below for uncertainty calibration: We use the RAG model to generate an answer to the question using a beam generator and transition probabilities using Equation 4:

$$a \leftarrow P_{\eta,\phi}(a|a;K,L,S)$$

We then use the retriever component of the RAG model to fetch the top $M(\sim 3)$ document segments that were used to generate the answer:

$$(d_1, ..., d_M) \leftarrow P_{\eta}(.|[q; a])$$

We finetune the generator on a new instruction task: the task of predicting whether the answer is correct or wrong, given the question and document set. We first compute the log-probability associated with predicting "correct" or "wrong", using the RAG-generator.

$$y \leftarrow P_{\phi}(a, q, (d_1, ..., d_M))$$

We then minimize the cross-entropy loss based on the ground-truth label and the model's assessment of the answer's veracity:

$$loss(\phi) = C.E(y, \hat{y}) \tag{8}$$

References

- Hugo Abonizio, Luiz Bonifacio, Vitor Jeronymo,
 Roberto Lotufo, Jakub Zavrel, and Rodrigo Nogueira.
 2023. Inpars toolkit: A unified and reproducible synthetic data generation pipeline for neural information retrieval. arXiv preprint arXiv:2307.04601.
- Jo Kristian Bergum. 2023. Improving zero-shot ranking with vespa hybrid search part two. *blog.vespa.ai/*.
- Lingjiao Chen, Matei Zaharia, and James Zou. 2023. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv* preprint arXiv:2305.05176.
- Zhuyun Dai, Vincent Y Zhao, Ji Ma, Yi Luan, Jianmo Ni, Jing Lu, Anton Bakalov, Kelvin Guu, Keith B Hall, and Ming-Wei Chang. 2022. Promptagator: Few-shot dense retrieval from 8 examples. *arXiv* preprint arXiv:2209.11755.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.
- Arnav Gudibande, Eric Wallace, Charlie Snell, Xinyang Geng, Hao Liu, Pieter Abbeel, Sergey Levine, and Dawn Song. 2023. The false promise of imitating proprietary llms. *arXiv preprint arXiv:2305.15717*.
- Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR.
- Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3929–3938. PMLR.
- Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. arXiv preprint arXiv:2305.02301.
- Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022. Few-shot learning with retrieval augmented language models. *arXiv* preprint *arXiv*:2208.03299.
- Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield Dodds, Nova DasSarma, Eli Tran-Johnson, et al. 2022. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*.

- Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. 2020. UNIFIEDQA: Crossing format boundaries with a single QA system. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1896–1907, Online. Association for Computational Linguistics.
- Omar Khattab, Christopher Potts, and Matei Zaharia. 2021. Relevance-guided supervision for OpenQA with ColBERT. *Transactions of the Association for Computational Linguistics*, 9:929–944.
- Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48.
- Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. *arXiv preprint arXiv:1906.00300*.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.
- Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Teaching models to express their uncertainty in words. *arXiv preprint arXiv:2205.14334*.
- Zhen Lin, Shubhendu Trivedi, and Jimeng Sun. 2023. Generating with confidence: Uncertainty quantification for black-box large language models. *arXiv* preprint arXiv:2305.19187.
- Ji Ma, Ivan Korotkov, Yinfei Yang, Keith Hall, and Ryan McDonald. 2020. Zero-shot neural passage retrieval via domain-targeted synthetic question generation. *arXiv preprint arXiv:2004.14503*.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.
- Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *arXiv preprint arXiv:2302.00083*.
- Jon Saad-Falcon, Omar Khattab, Keshav Santhanam, Radu Florian, Martin Franz, Salim Roukos, Avirup Sil, Md Arafat Sultan, and Christopher Potts. 2023. Udapdr: Unsupervised domain adaptation via llm

- prompting and distillation of rerankers. *arXiv* preprint arXiv:2303.00807.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.
- Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. 2023. Replug: Retrieval-augmented black-box language models. *arXiv* preprint arXiv:2301.12652.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems, 35:24824–24837.

A Appendix I

Prompt Template for Extractive/YN Formats

"Given the passage, generate an extractive question answer pair, relevant to the passage. The answer to the question should be a span from the given passage or a yes/no answer. Provide a rationale for the answer as well. <Exemplars>

Passage: [X]".

Prompt Template for Abstractive Formats

"Given the passage, generate an abstractive question answer pair, relevant to the passage. The answer should be grounded in the passage. <Exemplars>

Passage: [X]".