Large Language Models are Built-in Autoregressive Search Engines

Noah Ziems, Wenhao Yu, Zhihan Zhang, Meng Jiang
University of Notre Dame
{nziems2, wyu1, zzhang23, mjiang2}@nd.edu

Abstract

Document retrieval is a key stage of standard Web search engines. Existing dual-encoder dense retrievers obtain representations for questions and documents independently, allowing for only shallow interactions between them. To overcome this limitation, recent autoregressive search engines replace the dual-encoder architecture by directly generating identifiers for relevant documents in the candidate pool. However, the training cost of such autoregressive search engines rises sharply as the number of candidate documents increases. In this paper, we find that large language models (LLMs) can follow human instructions to directly generate URLs for document retrieval.

Surprisingly, when providing a few Query-URL pairs as in-context demonstrations, LLMs can generate Web URLs where nearly 90% of the corresponding documents contain correct answers to open-domain questions. In this way, LLMs can be thought of as built-in search engines, since they have not been explicitly trained to map questions to document identifiers. Experiments demonstrate that our method can consistently achieve better retrieval performance than existing retrieval approaches by a significant margin on three open-domain question answering benchmarks, under both zero and few-shot settings. The code for this work can be found at https://github.com/Ziems/llm-url.

1 Introduction

Along with the success of deep learning, dual-encoder based retrievers have become the dominant method for Web searching (Zhu et al., 2021; Zhao et al., 2022). For example, DPR (Karpukhin et al., 2020) employs two independent encoders to encode the question and the document respectively, then estimates their relevance by computing a single similarity score between two representations.

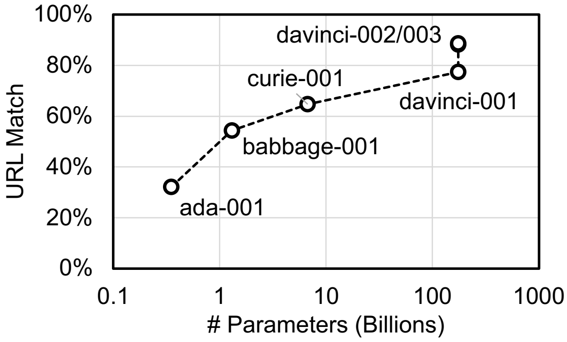

Figure 1: Successful URL reconstructions by different size of GPT-3 as URL generators. The models are prompted with the first 100 words of a Wikipedia page then tasked with generating the URL of the page they came from. Tested on 10k Wikipedia pages sampled from the top 100k most frequent Wikipedia entities.

However, these methods suffer from two major drawbacks. First, the representations of questions and documents are typically obtained independently in modern dual-encoder dense retrieval models (Karpukhin et al., 2020), allowing for only shallow interactions between them (Khattab et al., 2021). Second, the question or document representation is embedded into a single dense vector, potentially missing fine-grained information when computing the similarity between the two vector representations (Khattab and Zaharia, 2020).

Instead of computing similarity between question and document embeddings, autoregressive search engines aim to directly generate document identifiers then map them to complete documents in the predetermined candidate pool. This approach has attracted increasing interest in information retrieval (IR) and related fields (Tay et al., 2022; Bevilacqua et al., 2022; Wang et al., 2022). Compared to dual-encoder dense retrieval methods, autoregressive search engines enjoy a number of advantages. First, autoregressive generation models produce document identifiers by performing deep token-level cross-attention, resulting in a better esti-

Findings of the Association for Computational Linguistics: ACL 2023, pages 2666–2678
July 9-14, 2023 ©2023 Association for Computational Linguistics

mation than shallow interactions in dense retrievers. Second, autoregressive search engines have been shown to have strong generalization abilities, outperforming BM25 in a zero-shot setting (Tay et al., 2022). While it is theoretically possible to scale an autoregressive search engine to the size of a large language model (LLM), such as GPT-3 with 175B parameters, in practice it is not feasible due to the computational overhead of training such a large autoregressive search engine from scratch (Tay et al., 2022). To reduce the high training cost of autoregressive search engine, a smaller model size is preferred. However, the results of our pilot study in Figure 1 show smaller language models are significantly worse at mapping passages to document identifiers than larger ones. Moreover, different retrieval tasks can have unique retrieval requirements. One task may require a model to retrieve factual evidence to support or refute a claim (*i.e.*, fact checking) (Onoe et al., 2021) while another may require a model to retrieve specific trivia information about an entity (*i.e.*, entity linking) (Petroni et al., 2021; Zhang et al., 2022). It would be better if the retriever was capable of generalizing to new retrieval tasks with only a few examples.

In this work, we explore the use of in-context demonstrations to prompt LLMs to directly generate web URLs for document retrieval, namely LLM-URL. Surprisingly, we find that by providing a few (query, URL) pairs as contextual demonstrations, large language models (e.g. GPT-3) generate Web URLs where nearly 90% of the corresponding documents contain answers to open-domain questions. In this way, LLMs can be thought of as built-in search engines, as they have not been explicitly trained to map questions or documents to identifiers. Instead of using newly-created document identifiers, LLM-URL leverages existing and widely used document identifiers directly, *i.e.*, URLs. We compare our approach to existing document retrieval methods on three different open-domain question answering (QA) datasets: WebQ (Berant et al., 2013), NQ (Kwiatkowski et al., 2019), and TriviaQA (Joshi et al., 2017). Further, to avoid exceeding the limit on the number of input tokens of LLMs, we employ an unsupervised passage filtering module to remove irrelevant portions of supporting documents. To summarize, our main contributions are as follows:

1. We reveal that LLMs are built-in autoregressive search engines capable of document retrieval by directly generating Web page URLs under both zero and few-shot settings.

2. We show retrieving documents by generating URLs with LLMs significantly outperforms existing methods for document retrieval, as measured by Recall@K. Further, we show that breaking the retrieved documents into passages then using a ranker to filter the passages significantly reduces the number of supporting passages while maintaining high recall.

3. We show the retrieved documents improve downstream QA performance as measured by EM when compared to baseline methods.

2 Related Work

2.1 Traditional Document Retrievers

Traditional methods such as TF-IDF and BM25 explore sparse retrieval strategies by matching the overlapping contents between questions and passages (Robertson and Zaragoza, 2009; Chen et al., 2017; Yang et al., 2019). DPR (Karpukhin et al., 2020) revolutionized the field by utilizing dense contextualized vectors for passage indexing. It is first initialized as a pretrained BERT model, then trained discriminatively using pairs of queries and relevant documents, with hard negatives from BM25. Recent research has improved DPR via better training strategies (Xiong et al., 2020; Qu et al., 2021; Zhang et al., 2023a) and passage re-ranking (Mao et al., 2021; Yu et al., 2021; Ju et al., 2022). However, representations of questions and documents are typically obtained independently in modern dual-encoder dense retrieval models (Karpukhin et al., 2020; Xiong et al., 2020), allowing for only shallow interactions between them (Khattab et al., 2021).

2.2 Autoregressive Search Engines

Recent works have investigated the use of auto-regressive language models to generate identifier strings for documents as an intermediate target for retrieval (Yu et al., 2022), such as Wikipedia page titles (De Cao et al., 2020), root-to-leaf paths in a hierarchical cluster tree (Tay et al., 2022), or distinctive n-grams that can be mapped to full passages (Bevilacqua et al., 2022). Since the series of work was carried out almost simultaneously by different research groups, they are often referred to multiple different names in the literature, such as autoregressive search engine, differential search

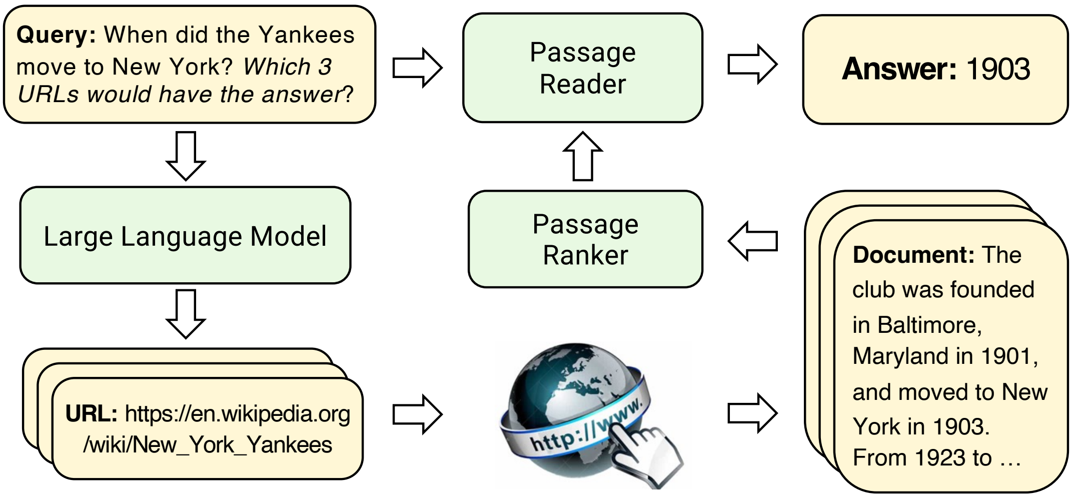
Query: When did the Yankees move to New York? *Which 3 URLs would have the answer?*
Passage Reader
Answer: 1903
Large Language Model
Passage Ranker
Document: The club was founded in Baltimore, Maryland in 1901, and moved to New York in 1903. From 1923 to …
URL: https://en.wikipedia.org /wiki/New_York_Yankees
http://www.

Figure 2: The overall pipeline of our proposed LLM-URL. Given a question, LLM-URL first generates a set of URLs which are extracted from the generated text. The URLs are retrieved from the Internet then broken into passages which are ranked and filtered such that only the most relevant are kept. Finally, these passages are given as input to a reader model along with the original question to generate a final answer.

index (DSI), and neural document indexers (NDI). Compared to traditional dense document retrievers, these methods leverage a generation model to produce the document indexes. By forcing the generation model to explain every token in the question and document using cross-attention, the generation abilities of the model significantly improve. Our work is closely related to these works, showing experimentally that properly prompting pretrained large language models can achieve better performance than traditional dense retrieval models (Ouyang et al., 2022; Yu et al., 2023) .

3 Proposed Method

In this section we describe a new method, which we refer to as LLM-URL, that employs a large language model (LLM) to perform effective and efficient web document retrieval for knowledge-intensive NLP tasks such as open-domain question answering (ODQA).

ODQA is a two step process consisting of a *retriever* and a *reader*. Given a question q, the goal of the *retriever* is to find the top-n passages $\mathcal{P}_n$ relevant to answering q. Given q and the top-n relevant passages $\mathcal{P}_n$, the goal of the *reader* is to use internal knowledge along with $\mathcal{P}_n$ to generate a correct answer a to question q. The passage retriever plays an essential role in this process. When $\mathcal{P}_n$ contains more passages that have the correct answer, the reader has a higher chance of finding it. Instead of *heavily training a dedicated retriever*, our LLM-URL solves the problem in a different way as shown in Figure 2.

Given a question q, our LLM-URL should find a set of relevant passages to $\mathcal{P}_n$ and give it to the reader. First, it prompts a LLM (e.g., GPT-3) to directly generate m URLs for q. By default, it uses "Which m Wikipedia URLs would have the answer?" as the instruction which is appended to each input question as the prompt. We also append the beginning of the Wikipedia URL (`https://en.wikipedia.org/wiki`) to the end of the prompt to encourage the generation of URLs and restrict generation to the Wikipedia article URL format. As LLM has the ability of in-context learning, we take this advantage to enable the few-shot setting in the prompt. The prompt described above also includes a series of in-context demonstrations. Each demonstration contains a question sampled from the training set following the prompt described above. At the end of each demonstration, m URLs which point to gold-labeled documents are listed. In the zero-shot setting, the original prompt is used without any demonstrations. In the few-shot setting, the original prompt appended to a series of d demonstrations (d=10 in this work).

Given the prompt, the LLM returns a generated sequence of tokens. Ideally these tokens would construct a sequence of m separated URLs. In practice, the generated sequence often has extra information such as a proposed answer that is unreliable and needs to be filtered. We use a regular expression to extract all URLs from the sequence and discard all

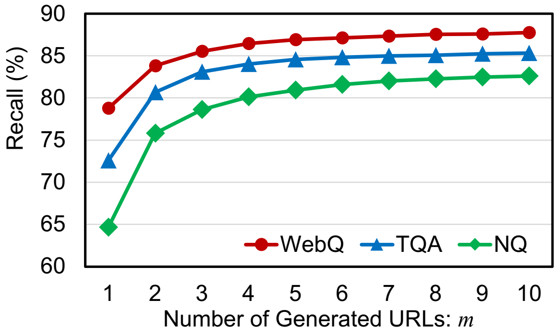

Figure 3: We prompt LLM-URL to generate m documents and measure the recall as m increases. Significant recall improvements are seen when m is small but as it increases the marginal benefit decreases.

extra information. This also filters out many URLs that are improperly formatted. After extraction, GET requests are made using the extracted URLs and the contents of each retrieval is used to create a set of fetched documents $\mathcal{D}_f$. Often, $|\mathcal{D}_f| < m$ because some of the generated URLs do not follow a correct format or do not point to real web pages on the Internet.

The set of fetched documents $\mathcal{D}_f$ can be passed directly to a reader if m is a small value or the reader being used can handle many large documents. However, this is usually not the case. Often, $\mathcal{D}_f$ needs to be filtered such that only a small number of the most relevant passages are given to the reader. To do this, our LLM-URL first breaks each document $d \in \mathcal{D}_f$ into a set of small passages. The passages from each document are collected into a new set, $\mathcal{P}_f$. A scoring function is used to quantify the relevance of each passage with respect to the question q, with high values indicating high relevance with respect to q and low scores indicating low relevance. A simple scoring function such as BM25 can be used or a more complex one such as DPR (Karpukhin et al., 2020) can. The passages in $\mathcal{P}_f$ are then sorted from highest to lowest and the top n are kept as $\mathcal{P}_n$. Finally, $\mathcal{P}_n$ are given to a reader along with q to generate an answer.

Advantages of LLM-URL : Existing autoregressive retrieval methods such as DSI and SEAL use a pre-trained large language model then fine tune it to take questions as input and generate relevant document identifiers as output (Tay et al., 2022; Bevilacqua et al., 2022). Both DSI and SEAL do extensive experiments on a variety of document identifiers which are generated by a heavily trained language model. Examples of these identifiers include unstructured atomic identifiers, naively structured string identifiers, hierarchical document clustering, and others. LLM-URL instead uses pre-existing document identifiers that exist on the internet: URLs. Using URLs instead of the aforementioned identifiers has multiple advantages. URLs often contain words related to the information they link to, allowing for strong association of topics with their URLs. For example, the title of each Wikipedia page is used in its URL, allowing the LLM is able to directly generate the URL by leveraging semantic information from the question. To validate the importance of URLs themselves, we also experiment with prompting the LLM to generate Wikipedia titles instead of URLs and find Recall@1 significantly reduces compared to prompting for URL generation. We believe this is because the URL format itself helps prompt the model for specific information in a specific format. Further, the use of URLs allows us to simply obtain the evidence document via a HTTP request without any need of training a model or building an index to find the mapping between identifiers and documents.

4 Experiments

In this section, we present and discuss results from our experiments to "directly" demonstrate that our LLM-URL is a strong retriever and "indirectly" show that it achieves competitive performance on the ODQA task against state-of-the-art solutions.

Large Language Model: Following Figure 1, the large language model we use to generate URLs for our experiments is GPT-3 *text-davinci-003* with greedy decoding and a temperature of 0. A variety of different prompts are tested for generating URLs, but little difference in performance is observed, so we simply use the best performing prompt which is discussed in Section 3.

Datasets: We use three ODQA datasets including Web Questions, Natural Questions, and Trivia QA. We use them to perform evaluation on both the task of document or passage retrieval and ODQA itself.

4.1 Retrieval

We expect retrievers to find the most relevant documents and/or passages. We conduct experiments on both document retrieval and passage retrieval.

Evaluation metrics. Recall@k (k=1, 10, 100) is calculated by measuring the percentage of docu-

Method	Document Recall@1			Document Recall@10		
	WebQ	NQ	TriviaQA	WebQ	NQ	TriviaQA
Contriever (Izacard et al., 2021)	63.8	53.2	60.6	63.8	80.8	82.5
BM25 (Robertson and Zaragoza, 2009)	49.5	47.2	63.0	81.5	76.8	82.3
Google API	61.1	55.5	51.4	-	-	-
LLM-URL (Zero-Shot)	76.8	61.7	71.3	87.7	83.2	85.5
LLM-URL (Few-Shot)	**79.7**	**62.6**	**73.5**	**89.9**	**83.9**	**86.8**

Table 1: Document retrieval as measured by Recall@k. Google API Recall@10 results are left out due to high cost.

Method	Passage Recall@1			Passage Recall@10			Passage Recall@100		
	WebQ	NQ	TriviaQA	WebQ	NQ	TriviaQA	WebQ	NQ	TriviaQA
Contriever	18.2	18.8	34.0	55.7	54.8	67.9	79.8	**79.6**	83.3
BM25	19.1	22.8	46.2	51.8	55.6	71.7	76.6	**79.6**	83.9
LLM-URL (Zero-Shot)	22.2	24.0	46.7	63.1	60.6	76.6	83.8	78.3	83.6
LLM-URL (Few-Shot)	**22.3**	**25.5**	**49.1**	**64.8**	**60.8**	**77.8**	**85.9**	79.0	**84.8**

Table 2: Passage retrieval as measured by Recall@1, Recall@10 and Recall@100. Here LLM-URL is equipped with BM25 to perform the ranking task.

ments or passages in the top-k which contain one of the gold labeled answers while exact match is calculated by the percentage of predicted answers which match one of the gold labeled answers. While LLM-URL is not constrained by which URLs can be generated for document retrieval, we restrict all generations to Wikipedia URLs only for fair comparison, as discussed in Section 3 All baseline models also use Wikipedia for retrieval, with some fetching documents in real time and others fetching from an offline corpus.

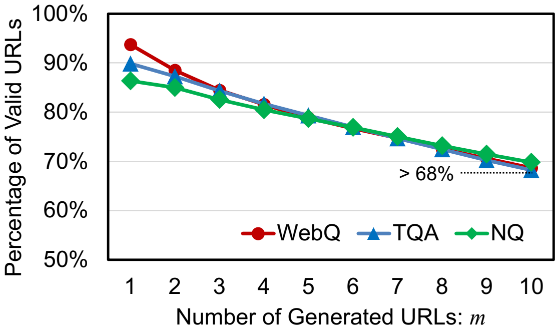

Figure 4: The percentage of valid URLs generated from LLM-URL as the total number of generated URLs m increases from 1 to 10. As m increases, invalid URL generations become more frequent.

4.1.1 Document Retrieval

Baselines: Contriever (Izacard et al., 2021) and BM25 (Robertson and Zaragoza, 2009) are usually used for passage retrieval. Contriever is a dual encoder which uses a dot product between dense representations of a question and passage to calculate relevance. BM25 is a sparse retriever which uses the overlapping contents between question and passage to calculate relevance. Because we use the same passage size to chunk Wikipedia documents, we were able to map their retrieved passages back to the original documents. We use Google API (Brin and Page, 1998) restricted to Wikipedia as a third baseline to retrieve relevant documents given a question.

Existing works such as DSI and SEAL have investigated the use of autoregressive language models to generate identifier strings for documents as an intermediate target for retrieval. DSI is a Transformer which has been trained to map directly from question to document identifiers by memorizing the contents of the entire corpus (Tay et al., 2022). SEAL is a variant of DSI which uses ngrams as document ids to improve retrieval performance (Bevilacqua et al., 2022). Neither DSI nor SEAL report retrieval results on full documents and do not have publicly available implementations, so they are left out and discussed in Table 3 and Section 4.1.2 on passage retrieval.

Unlike the baselines, our LLM-URL employs an LLM. It has two settings: zero-shot and few-shot. In the zero-shot setting, no in-context demonstrations are given whereas in the few-shot setting a few demonstrations are appended to the prompt.

Method	Recall@1	Recall@10
DSI[1]	25.1	56.6
SEAL[1]	**26.3**	**74.5**
LLM-URL (Zero-Shot)	24.0	60.6
LLM-URL (Few-Shot)	25.5	60.8

[1] *explicitly trained for retrieval on NQ*

Table 3: Passage retrieval as measured by Recall@1 and Recall@10. LLM-URL is equipped with BM25 for passage ranking. Other datasets are left out due to not being reported in either paper and no public implementations.

Results: The results of our document retrieval experiments are shown in Table 1. In this setting Recall@k is calculated directly after the documents are retrieved with no intermediary steps. LLM-URL significantly outperforms baseline methods on all datasets for both Recall@1 and Recall@10. Specifically, zero-shot LLM-URL improves document Recall@1 relatively by 20.4%, 11.2%, and 13.2% over the strongest baseline on WebQ, NQ, and TriviaQA, respectively. Few-shot LLM-URL further expands the improvement to 24.9%, 12.8%, and 16.7%, respectively. URLs can be extracted from the large-scale parameters of LLMs, and these URLs can lead to more accurate documents than what existing methods can retrieve. Both the LLM parameters and in-context demonstrations are significantly useful in document retrieval.

Figure 3 shows Recall scores converge when the number of generated URLs m increases. Due to the diminishing returns from increasing m, our experiments do not explore values of m greater than 10.

Are the generated URLs valid? It is worth noting that the generated URLs are not always valid. Some generated URLs do not have valid URL syntax and some point to Wikipedia pages that do not exist. Rarely, URLs will be generated for domains aside from Wikipedia. For fair comparison, all of these faulty URLs are discarded and only documents coming from valid Wikipedia articles are kept.

Further analysis is done to measure the ratio of valid Wikipedia URLs while the total number of generated URLs m increases from 1 to 10, shown in Figure 4. The number of valid URL generations remains surprisingly high (i.e., higher than 68%) as m increases from 1 to 10. However, the rate of valid generations appears to fall off as m increases, indicating there are diminishing returns from each marginal increase of m.

Method	Zero-Shot QA EM		
	WebQ	NQ	TriviaQA
Contriever + InstructGPT	16.8	19.1	52.4
BM25 + InstructGPT	16.0	20.5	53.3
Google + InstructGPT	19.9	27.8	58.7
GenRead (InstructGPT)	24.8	28.2	59.3
DSI[1] + FiD	-	31.4[2]	-
SEAL[1] + FiD	-	**43.6**	41.8
InstructGPT (no docs.)	18.6	20.9	52.6
LLM-URL (Zero-Shot)	28.1	26.4	60.1
LLM-URL (Few-Shot)	**29.0**	27.3	**60.7**

[1] *explicitly trained for retrieval on NQ*

[2] *result from Bevilacqua et al. (2022)*

Table 4: Zero-shot open-domain QA performance as measured by exact match (EM). All LLM-URL models use InstructGPT as the reader unless otherwise stated.

4.1.2 Passage Retrieval

Baselines: Four methods, including Contriever, BM25, DSI (Tay et al., 2022), and SEAL (Bevilacqua et al., 2022), were introduced in Section 4.1.1. Google API was used for document retrieval and not applied to passages.

Results: The results of our passage retrieval experiments are shown in Table 2. In this setting Recall@k is calculated on the top-k passages ranked by the ranker instead of just on the raw documents shown in Table 1. LLM-URL performs slightly better than baseline methods for Recall@1 and Recall@10 and as well as baseline methods for Recall@100. In the zero-shot setting, LLM-URL improves relative Recall@1 by 16.2%, 5.3%, and 1.1% with respect to the strongest baseline on WebQ, NQ, and TriviaA respectively. The few-shot setting of LLM-URL expands the improvement to 16.8%, 11.8%, and 6.3%, respectively. For Recall@10, similar improvements can be seen.

For Recall@100, performance is better relative to baseline models for all datasets except NQ. In the zero-shot setting, LLM-URL improves the relative Recall@100 by 5.0% for WebQ and performs slightly worse than the best baseline method on NQ and TriviaQA by 1.7% and 0.4% respectively. The few-shot setting of LLM-URL for Recall@100 shows a slight improvement on WebQ and TriviaQA, but performs slightly worse than the strongest baseline on NQ.

Despite being limited to only the passages from 10 documents, LLM-URL performs better than baseline methods for smaller k and performs as well as baseline methods for higher values of k.

The comparison between LLM-URL and existing document identifier-based methods such as DSI and SEAL are shown in Table 3. For Recall@1, zero-shot LLM-URL performs slightly worse than the best baseline by 8.8%. This performance gap is slightly smaller in the few-shot setting with LLM-URL performing 3.1% worse than the best baseline. For Recall@10, zero-shot LLM-URL performs worse than the best baseline by 18.7%. Few-shot LLM-URL performs only slightly better than the zero-shot setting, performing worse than the best baseline by 18.4%.

4.2 Open-Domain Question Answering

Evaluation metric: We use exact match (EM), which is short for *exact string match with the correct answer*, because the goal of ODQA is to find an exact answer to any question using Wikipedia articles.

Results: Here we discuss the downstream QA performance of LLM-URL. In this setting, an answer only has an exact match if the normalized generated text is within the list of acceptable answers to a question. When combined with InstructGPT as a reader, LLM-URL performs significantly better on WebQ and slightly better on TriviaQA when compared with the best performing baseline methods. On NQ, LLM-URL+InstructGPT performs worse than baseline NDIs and only slightly worse than the best remaining baseline. In the zero-shot setting, LLM-URL+InstructGPT improves upon the best baseline method by 13.3% and 1.3% on WebQ and TriviaQA respectively. LLM-URL +InstructGPT performs worse than the best baseline method by 39.5% on NQ. In the few-shot setting, LLM-URL+InstructGPT performs better than the best baseline method by 16.9% and 2.3% on WebQ and TriviaQA respectively. LLM-URL+InstructGPT performs worse than the best baseline method by 37.4% on NQ.

Despite not being explicitly trained for retrieval, LLM-URL+InstructGPT performs significantly better than baseline methods for WebQ, achieves on-par performance with existing methods for TriviaQA, and performs slightly worse than existing methods for NQ.

Our results indicate LLM-URL could be a promising solution to retrieval for a wide range of knowledge intensive tasks with little to no training data required.

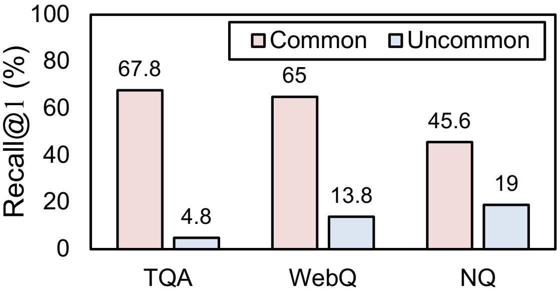

Figure 5: Common vs Uncommon entity recall@1. Common is defined as a question containing entities in the top-1 million most common entities on Wikipedia. LLM-URL performs much better when retrieving information about common entities.

4.3 Discussions

4.3.1 Time Sensitive Queries

There are a number of additional qualitative benefits that LLM-URL has over existing methods. One large advantage of LLM-URL is that the documents are retrieved in real time from the source. So long as the source stays up to date without the URL itself changing, our proposed method is capable of answering time sensitive queries without any extra modifications.

In contrast, existing dual encoder approaches such as Contriever require a document to be re-encoded each time it changes. Existing methods such as SEAL (Bevilacqua et al., 2022) and DSI are also tricky to keep up to date for time sensitive queries as the LLM would have to be retrained to learn the new content of the updated documents.

4.3.2 Frequent Entities Analysis

Following (Mallen et al., 2022), we analyze the retrieval performance of LLM-URL when the gold-labeled answer entity is common versus when it is not. For each question-answer pair in a given dataset we check to see if the labeled entity exists within the top one-million common entities from Wikipedia. Using this, we split our dataset into two distinct subsets: question-answer pairs that contain a common entity and those that do not. In measuring the performance of our model on these two distinct sets across Web Questions, Natural Questions, and TriviaQA, we find LLM-URL performs significantly better on common entity question-answer pairs. The results of our analysis are shown in Figure 5. Across all three datasets,

LLM-URL	Exists	Answer	Contriever	Answer	BM25	Answer
wiki/Jellyfish	✓	✓	Smack (ship)	✗	Collective noun	✗
wiki/Collective_noun	✓	✗	Collective noun	✗	Determiner	✗
wiki/Smack_(group)	✗	✗	Cetacean intelligence	✗	Glass sea creatures	✗
wiki/Cnidaria	✓	✓	Well smack	✗	Minotaur	✗
wiki/Medusozoa	✓	✓	Plankton	✗	Mass noun	✗
wiki/Scyphozoa	✓	✓	Sperm whale	✗	Well smack	✗
wiki/Cubozoa	✓	✓	Loaded question	✗	Nomenclature	✗
wiki/Hydrozoa	✓	✓	Jabberwocky	✗	Archomental	✗
wiki/Staurozoa	✓	✓	Merrow	✗	Honey Smacks	✗
wiki/Rhizostomeae	✓	✓	Loaded question	✗	Well smack	✗

Table 5: Case study of retrieved documents from the question "*A 'smack' is a collective noun for a group of which sea creatures?*". "Exists" means whether the URL points to a valid Wiki page. "Answer" means whether the document contains the answer. We omit the prefix of generated URLs for brevity (`https://en.wikipedia.org/`). For BM25 and Contriever, we show the document titles of the top-10 retrieved passages, respectively. The correct answer is "jellyfish."

the recall of common-entity question-answer pairs is many times greater than the recall from the rest of the dataset.

Previous work has shown LLMs in the closed-book setting, where the model must rely solely on the information contained within its weights, perform much better on common-entities versus uncommon ones (Mallen et al., 2022). Our results show this problem extends beyond the closed-book setting and also applies to retrieval when using LLM-URL. This also could explain the high word count from documents we found when evaluating LLM-URL. The average Wikipedia article is 644 words, but the average word count from Wikipedia documents retrieved via LLM-URL was 10k. We believe this discrepancy is caused by common entities having much more detail in their Wikipedia articles and in turn having much higher word count.

4.3.3 Case Study

In Table 5, we show a case study comparing LLM-URL with two baseline retrieval methods, BM25 and Contriever, on the question "A 'smack' is a collective noun for a group of which sea creatures?" which is in the TriviaQA test set. The gold-labeled answer to this question is "jellyfish".

In the closed-book setting, InstructGPT mistakenly predicts "dolphins" as the answer. When using Contriever to retrieve 10 passages from Wikipedia given the query, none of the passages contains the gold answer. For instance, Contriever retrieves passages about "smack", a kind of fishing vessel, along with other passages about sperm whales, plankton, and other unrelated topics. Similar results are found while using BM25 as the retriever.

In contrast, LLM-URL performs much better in this scenario, retrieving 7 documents which contain the answer. The top retrieved document is exactly about the gold answer "jellyfish". The fourth to the tenth documents all talk about different types of jellyfish. After being chunked into passages then sorted by the ranker, the top 10 passages are concatenated. Among them, it contains "A group of jellyfish is called a smack," which contains the answer to the question and comes directly from the first retrieved document, titled "jellyfish." When InstructGPT is then prompted with these 10 passages along with the question, the gold answer "jellyfish" is correctly generated.

This case study highlights multiple advantages of LLM-URL . First, LLM-URL finds documents related to both the question and the answer. It directly locates documents that talks about "jellyfish" instead while BM25 and Contriever locate documents related to the question only–not the answer. Second, LLM-URL is more precise than BM25 or Contriever. In this case, 7 out of 10 generated URLs from LLM-URL point to a Wikipedia document that contains the answer. However, both BM25 and Contriever fail to retrieve any documents containing the answer. Third, the set of documents retrieved by LLM-URL are complementary to each other, while in BM25 or contriever, each document in the top-10 is selected independently. This is because the LLM is able to refer to previous generated URLs before it generates the next one, allowing each newly generated URL to be conditioned on all the previous URLs. This leads to a more informative evidence context in open-domain question answering.

5 Conclusion and Future Work

In this paper, we explored whether large language models can generate URLs prompted by human instructions for document retrieval. Surprisingly, we found that by providing a few (query, URL) pairs as in-context demonstrations, large language models (e.g. GPT-3) generated Web URLs where near 90% of the corresponding documents contain correct answers to open-domain questions in WebQ. Furthermore, by breaking the retrieved documents into passages then ranking them with BM25, we showed a significant number of unnecessary passages could be filtered out while retaining high recall, which outperformed baseline methods by a significant margin.

There are numerous exciting directions for future work. While a number of broad spectrum retrieval benchmarks such as BIER (Thakur et al., 2021) exist, it remains to be seen whether the few-shot demonstrations shown in this work can be further tuned for specific retrieval tasks. Promptagator (Dai et al., 2022) shows significant performance improvements can be achieved by tuning prompts in a similar way.

Further, it remains to be seen whether fine tuning the prompt for each individual question can further improve the retrieval performance. As with Promptagator, prior work has shown using clustering to select diverse demonstrations for any given question further improves retrieval performance as well as downstream QA performance.

Limitations

Despite the strong performance on the presented datasets, our approach is limited in its ability to update knowledge state and adapt to new domains. A major feature of *retrieve-then-read* is the ability to swap in new documents when new information is learned, such as temporally more recent documents, or adding in documents from a new domain to quickly adapt to a new downstream task. Our approach relies on a large language model to contain all this knowledge and adding new knowledge would likely require some retraining. In addition, large generation models still suffer from hallucination errors, resulting in incorrect predictions. When tasked with generating 10 URLs, LLM-URL may only generate 6 or 7 which link to valid documents. Finally, our approach involves very large language models, slow web requests, and document processing which may make it cumbersome to use in practice.

Acknowledgements

This work was supported by NSF IIS-2119531, IIS-2137396, IIS-2142827, CCF-1901059, and ONR N00014-22-1-2507. Wenhao Yu was partly supported by the Bloomberg Data Science Fellowship.

References

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pages 1533–1544.

Michele Bevilacqua, Giuseppe Ottaviano, Patrick Lewis, Wen-tau Yih, Sebastian Riedel, and Fabio Petroni. 2022. Autoregressive search engines: Generating substrings as document identifiers. *arXiv preprint arXiv:2204.10628*.

Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh International Conference on World Wide Web 7*, WWW7, page 107–117, NLD. Elsevier Science Publishers B. V.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. In *Procs. of ACL*.

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*.

Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. 2020. Autoregressive entity retrieval. In *International Conference on Learning Representations*.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *ACL*, pages 1601–1611.

Mingxuan Ju, Wenhao Yu, Tong Zhao, Chuxu Zhang, and Yanfang Ye. 2022. Grape: Knowledge graph enhanced passage reader for open-domain question answering. *arXiv preprint arXiv:2210.02933*.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

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.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: A benchmark for question answering research. *TACL*, pages 452–466.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Hannaneh Hajishirzi, and Daniel Khashabi. 2022. When not to trust language models: Investigating effectiveness and limitations of parametric and non-parametric memories. *arXiv preprint ArXiv:2212.10511*.

Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. 2021. Reader-guided passage reranking for open-domain question answering. In *Findings of ACL-IJCNLP*.

Yasumasa Onoe, Michael J. Q. Zhang, Eunsol Choi, and Greg Durrett. 2021. CREAK: A dataset for commonsense reasoning over entity knowledge. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS*.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*.

Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, et al. 2021. Kilt: a benchmark for knowledge intensive language tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2523–2544.

Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2021. Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering. In *Procs. of NAACL*.

Stephen E. Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Yi Tay, Vinh Q Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, et al. 2022. Transformer memory as a differentiable search index. *arXiv preprint arXiv:2202.06991*.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Yujing Wang, Yingyan Hou, Haonan Wang, Ziming Miao, Shibin Wu, Hao Sun, Qi Chen, Yuqing Xia, Chengmin Chi, Guoshuai Zhao, Zheng Liu, Xing Xie, Hao Sun, Weiwei Deng, Qi Zhang, and Mao Yang. 2022. A neural corpus indexer for document retrieval. In *Advances in Neural Information Processing Systems*.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *International Conference on Learning Representations*.

Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. 2019. End-to-end open-domain question answering with bertserini. In *NAACL 2019 (demo)*.

Donghan Yu, Chenguang Zhu, Yuwei Fang, Wenhao Yu, Shuohang Wang, Yichong Xu, Xiang Ren, Yiming Yang, and Michael Zeng. 2021. Kg-fid: Infusing knowledge graph in fusion-in-decoder for open-domain question answering. *arXiv preprint arXiv:2110.04330*.

Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. 2023. Generate rather than retrieve: Large language models are strong context generators. *International Conference for Learning Representation (ICLR)*.

Wenhao Yu, Chenguang Zhu, Zaitang Li, Zhiting Hu, Qingyun Wang, Heng Ji, and Meng Jiang. 2022. A survey of knowledge-enhanced text generation. *ACM Computing Surveys (CSUR)*.

Zhihan Zhang, Xiubo Geng, Tao Qin, Yunfang Wu, and Daxin Jiang. 2021. Knowledge-aware procedural text understanding with multi-stage training. In *WWW '21: The Web Conference 2021*.

Zhihan Zhang, Wenhao Yu, Zheng Ning, Mingxuan Ju, and Meng Jiang. 2023a. Exploring contrast consistency of open-domain question answering systems on minimally edited questions. *Trans. Assoc. Comput. Linguistics*.

Zhihan Zhang, Wenhao Yu, Mengxia Yu, Zhichun Guo, and Meng Jiang. 2023b. A survey of multi-task learning in natural language processing: Regarding task relatedness and training methods. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2023*.

Zhihan Zhang, Wenhao Yu, Chenguang Zhu, and Meng Jiang. 2022. A unified encoder-decoder framework with entity memory. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022*.

Wayne Xin Zhao, Jing Liu, Ruiyang Ren, and Ji-Rong Wen. 2022. Dense text retrieval based on pretrained language models: A survey. *arXiv preprint arXiv:2211.14876*.

Fengbin Zhu, Wenqiang Lei, Chao Wang, Jianming Zheng, Soujanya Poria, and Tat-Seng Chua. 2021. Retrieving and reading: A comprehensive survey on open-domain question answering. *arXiv preprint arXiv:2101.00774*.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Our limitations section is not numbered, but it is the last section of our paper.

☒ A2. Did you discuss any potential risks of your work?
We do not believe any of the risks mentioned in the checklist apply to this paper.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Our introduction is found in section one and the abstract comes before that.

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☒ Did you use or create scientific artifacts?

Left blank.

□ B1. Did you cite the creators of artifacts you used?
No response.

□ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
No response.

□ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
No response.

□ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
No response.

□ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
No response.

□ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
No response.

C ☑ Did you run computational experiments?

Section 4 describes our experiments

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Yes these details are found in a graph in Section 1 and hyperparameters are found in our experiments section (4).

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Yes we discuss our hyperparameters in section 4

☒ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Our method is deterministic so there is no variation in our results.

□ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Not applicable. Left blank.

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

□ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

□ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

□ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

□ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

□ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.