ReFactX: Scalable Reasoning with Reliable Facts via Constrained Generation

Riccardo Pozzi[1,3][0000−0002−4954−3837]⋆, Matteo Palmonari[1][0000−0002−1801−5118], Andrea Coletta[2][0000−0003−1401−1715]⋆⋆, Luigi Bellomarini[2][0000−0001−6863−0162]⋆⋆, Jens Lehmann[3,5,7][0000−0001−9108−4278]⋆⋆⋆, and Sahar Vahdati[3,4,5,6][0000−0002−7171−169X]

[1] University of Milano-Bicocca, Milano, Italy
riccardo.pozzi@unimib.it
[2] Banca d'Italia, Roma, Italy
[3] Institute for Applied Informatics (InfAI), Leipzig, Germany
[4] TIB Leibniz Information Centre for Science and Technology, Hannover, Germany
[5] ScaDS.AI Dresden/Leipzig, Technische Universität Dresden, Dresden, Germany
[6] Data Science Institute, Leibniz University Hannover, Hannover, Germany
[7] Amazon, Dresden, Germany

Abstract. Knowledge gaps and hallucinations are persistent challenges for Large Language Models (LLMs), which generate unreliable responses when lacking the necessary information to fulfill user instructions. Existing approaches, such as Retrieval-Augmented Generation (RAG) and tool use, aim to address these issues by incorporating external knowledge. Yet, they rely on additional models or services, resulting in complex pipelines, potential error propagation, and often requiring the model to process a large number of tokens. In this paper, we present ReFactX, a scalable method that enables LLMs to access external knowledge without depending on retrievers or auxiliary models. Our approach uses constrained generation with a pre-built prefix-tree index. Triples from Wikidata are verbalized in textual facts, tokenized, and indexed in a prefix tree for efficient access. During inference, to acquire external knowledge, the LLM generates facts with constrained generation which allows only sequences of tokens that form an existing fact. We evaluate our proposal on Question Answering and show that it scales to large knowledge bases (800 million facts), adapts to domain-specific data, and achieves effective results. These gains come with minimal generation-time overhead. ReFactX code is available at https://github.com/rpo19/ReFactX.

1 Introduction

Large Language Models (LLMs) have demonstrated a variety of capabilities, ranging from natural language understanding and generation to reasoning, es-

⋆ This work was done while Riccardo Pozzi was visiting InfAI and TU Dresden.
⋆⋆ The opinions expressed are personal and should not be attributed to Banca d'Italia.
⋆⋆⋆ This work was done outside of Amazon.

pecially when enhanced with techniques like Chain-of-Thought (CoT) prompting [28]. However, LLMs are prone to hallucinations [20] and their internal knowledge remains limited to their training data [5,18]. This complicates their application to knowledge-intensive tasks such as Question Answering (QA), especially when these tasks necessitate accessing information beyond the parametric knowledge of LLMs, including time-sensitive data or confidential corporate information.

Retrieval-Augmented Generation (RAG) [16,30] and tool-use [31,27,15] represent two prominent families of solutions. They enable LLMs to consult external knowledge bases (KBs) or execute external tools to enrich their responses with grounded information. We will refer to these methods as Knowledge-Enhanced QA (KE-QA). They often rely on additional components such as retrievers, entity linking systems, or auxiliary models, which are orchestrated in information processing pipelines. However, such pipeline-based or service-based solutions have a number of drawbacks. They require the optimization of a larger number of parameters and make it difficult to effectively back-propagate supervision signals to early components of the pipeline [34]. These approaches are susceptible to error-propagation [14], and knowledge updates may be complex to propagate to all the different components (e.g., to QA and entity linking knowledge bases).

An interesting technique that may overcome such limitations is constrained generation [4,23], which restricts the model's output space during decoding to sequences that satisfy predefined structural, syntactic, or semantic constraints. A few recent approaches have applied constrained generation to KE-QA, achieving promising results [17,19]. However, as the application of this technique to KE-QA is quite novel, there is an important research challenge to address to advance these techniques: the scalability to large knowledge bases. Can these approaches be applied to large KBs such as Wikidata[8]? In this paper, we present Reliable Fact eXtractor (ReFactX), an approach that addresses this challenge and poses itself as an alternative to pipeline-based solution. ReFactX allows LLMs to integrate external knowledge without depending on additional models or external retrievers, but relying solely on constrained generation, supported by a pre-built prefix-tree index, designed to facilitate and speed-up fact retrieval.

At inference time, the LLM is instructed via In-Context Learning (ICL) [7] to invoke the *Fact* command when external facts are required. Once the command is recognized, constrained generation is activated: the model produces tokens only along valid paths in the prefix tree, guaranteeing that the output matches a fact from the knowledge base, which is derived from Wikidata. Once an entire KB fact is generated, the decoding mechanism reverts back to normal. It is important to note that every generated token is selected based on LLM's probability estimates. Constrained generation only narrows the vocabulary to the tokens that form an existing fact in the KB, but always leaves the final choice to the LLM.

Motivating Example. We illustrate our approach in Figure 1, by depicting an example with the question *"When was the director of Slumdog Millionaire born?"*. The model is instructed to first reason on how to reach the correct

[8] https://www.wikidata.org

answer, and then it starts to acquire facts. Once the *Fact* command is called, we enable constrained generation (underlined in blue) and we guide the model to generate an existing fact. This way the model is able to first find that "Danny Boyle" is the director of "Slumdog Millionaire", then to find his birth date, and finally to answer the question correctly.

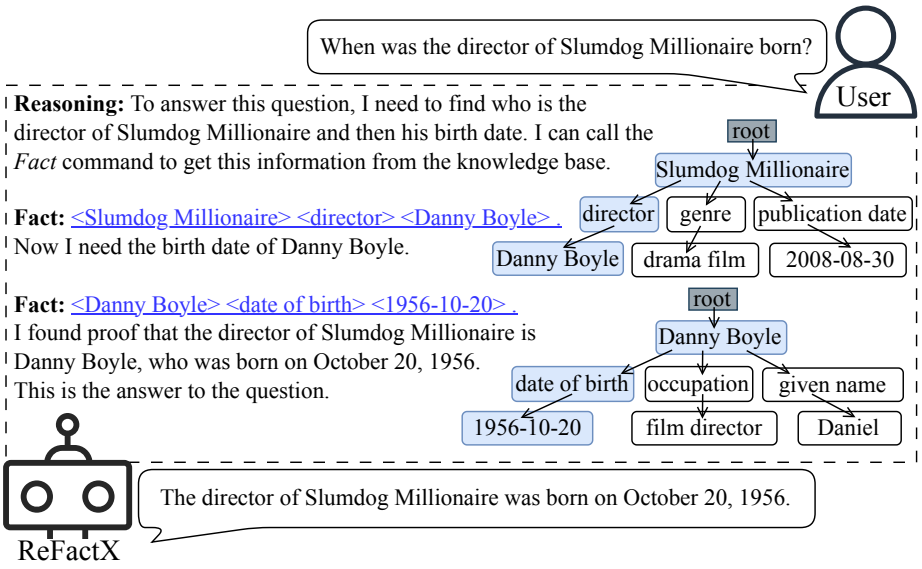

Fig. 1. ReFactX answering an open-domain question. The LLM sketches a plan, makes two *Fact* calls, and—with constrained generation (blue underline)—inserts valid facts from a Wikidata-based prefix tree before giving the final answer.

We can summarize the main contributions of this work as follows:

- **ReFactX**: a generic, constrained-generation system that enables any LLM to access very large KBs, with no external retrievers or pipelines.
- **Efficiency and scalability**: a disk-backed prefix tree capable of holding up to 800 million facts, adding only ∼1% latency.
- **Empirical validation**: insights on four QA datasets suggest that ReFactX can achieve competitive results, with accuracy gains exceeding 20 percentage points at over 90% precision on certain benchmarks—compared to LLMs answering only with their parametric knowledge.

2 Related Work

Several approaches have been proposed to incorporate external knowledge into LLMs [8]. In our work, we focus on Question Answering (QA)—more specifically,

on **Knowledge-Enhanced QA (KE-QA)**, where a model acquires external knowledge for answering questions.

Most of approaches can be classified as **input-based KE-QA**. They rely on external tools, such as search engines or dense retrievers [13], which can be arranged in pipeline-based workflows [16,30] or actively called by LLMs [31,22,15]. These approaches usually feed external knowledge as input (either in plain text or with dense embeddings) to the LLMs. Input-based KE-QA approaches, while effective in reducing hallucinations [25], require additional models or services for retrieving external information that can introduce delays, especially when calling remote APIs, increase training complexity [26], and raise substantially the number of input tokens [11], leading to higher response time and resource usage.

Other approaches can be classified as **memory-based KE-QA**. They propose memory modules, separated from model parameters, which make the models' internal mechanisms interact with external knowledge, e.g., by using cross-attention to incorporate external vectors [11,29]. While memory-based KE-QA approaches fuse external knowledge into LLMs' generation process more deeply, they typically require architectural modifications, complicating the use of pretrained LLMs.

Constrained generation [4,23], described in detail in Section 3.1, is a promising technique to avoid the drawbacks of input-based and memory-based KE-QA approaches. It has been successfully applied to various tasks, including code generation [23], guaranteeing that the model produces syntactically correct code, as well as entity linking [4] and information retrieval [2]. It has also been applied to instruction-following LLMs for several tasks that require the LLM to produce a specifically structured output, such as entity disambiguation [9].

Two recent studies [17,19] proposed **constrained generation for KE-QA**, especially for guiding LLMs through paths from a knowledge graph (KG). However, they do not address the problem of accessing facts from large knowledge bases in a scalable way (800 million facts, in our case), and still rely on entity linking systems for extracting question entities, similarly to input-based approaches. A first approach, *Decoding on Graphs (DoG)* [17], generates triples from a knowledge graph in a reasoning process similar to CoT [28] but grounded on KG knowledge, also allowing the model to alternate the constrained generation of triples and the reasoning with normal generation. A second approach, *Graph-Constrained Reasoning (GCR)* [19] generates entire paths from a KG using a fine-tuned LLM to find answer hypotheses, and then asks a top-tier language model to give a final answer based on the paths.

3 ReFactX: Enabling LLMs to Access Large-Scale KBs

ReFactX allows LLMs to efficiently access large KBs at inference time with constrained generation, generating valid facts, and weave them into chain-of-thought reasoning. In this work we construct our KB from Wikidata. The three subsections below walk through (i) the decoding mechanism, (ii) the scalable Wikidata index, and (iii) how both pieces plug into a QA workflow.

3.1 Constrained Fact-Generation Mechanism

The mechanism of constrained generation alters the autoregressive next-token generation process, in which the LLM parameterized by θ, iteratively estimates the probability of each token $P(t)$ from the vocabulary V to be the chosen as the next-token, given the current sequence $t_0..t_k$:

$$P_\theta(t|t_0..t_k) \forall t \in V \tag{1}$$

The next-token t_{k+1} can be chosen according to different sampling strategy; in greedy decoding the most probable token is chosen as follows:

$$t_{k+1} = \text{argmax}_{t \in V} P_\theta(t|t_0..t_k) \tag{2}$$

Intuitively, to constrain this process for generating only existing facts, we need to restrict V to only allow tokens that can form a fact from the KB. We define $V^A_{t_0..t_k}$, which contains all the tokens that can lead to an existing fact if added to the sequence $t_0..t_k$. Considering the example in Figure 2, at step t_{k+1}, when $t_0..t_k =$ "<Danny Boyle> <", $V^A_{t_0..t_k} =$ {"date","given"}.

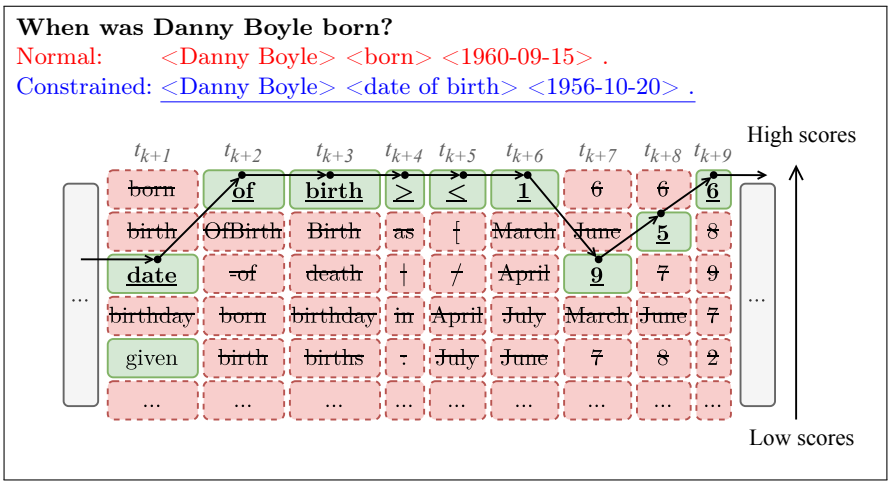

Fig. 2. Constrained decoding steers the LLM toward the correct fact. At each step, the constrained decoding mechanism chooses the highest-probability token that still completes a Wikidata fact, avoiding invalid branches and yielding "<Danny Boyle> <date of birth> <1956-10-20> .".

Next, we define the *next tokens* function NT_{KB} that, given $t_0..t_k$, obtains $V^A_{t_0..t_k}$ for the considered KB. Consequently, the token selection formula from Eq. 2 is updated to:

$$t_{k+1} = \text{argmax}_{t \in V^A_{t_0..t_k}} P_\theta(t|t_0..t_k), \text{ where } V^A_{t_0..t_k} = NT_{KB}(t_0..t_k). \tag{3}$$

However, the same result can be achieved by altering the probability distribution, setting the probability of all forbidden tokens to zero—without modifying the vocabulary:

$$P^c_\theta(t|t_0..t_k) = \begin{cases} P_\theta(t|t_0..t_k) & \text{if } t \in V^A_{t_0..t_k}, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

By using P^c_θ, instead of P_θ, we achieve constrained generation, relying on the assumption that the sampling strategy (Equation 2) would never choose any $t_{k+1}|P^c(t_{k+1}) = 0$. In practice, since at implementation level the models use log-probabilities, we directly set the log-probabilities of forbidden tokens to $-\infty$[9]:

$$\log P^c_\theta(t|t_0..t_k) = \begin{cases} \log P_\theta(t|t_0..t_k) & \text{if } t \in V^A_{t_0..t_k}, \\ -\infty & \text{otherwise.} \end{cases} \tag{5}$$

Figure 2 illustrates the application of constrained generation. It shows a fact produced by the same LLM in response to the question *"When was Danny Boyle born?"*—first using normal (unconstrained) generation, highlighted in red, and then using constrained generation, underlined in blue. While, the fact generated in normal mode is not correct, with constrained generation we are able to guide the LLM to the correct fact. The lower part of the figure details the mechanism of constrained generation. For each decoding step, allowed tokens are displayed in green boxes, whereas forbidden tokens are displayed in red boxes with dashed borders and a strikethrough. Tokens t_k are arranged in ascending order according to $P_\theta(t_k)$, from bottom to top.

Starting from the sequence $t_0..t_k =$ "<Danny Boyle> <", normal generation would generate $t_{k+1} = \text{argmax}_{t \in V} P_\theta(t|t_0..t_k) =$ "born", leading to an incorrect fact, while with constrained generation $t_{k+1} = \text{argmax}_{t \in V} P^c_\theta(t|t_0..t_k) =$ "date". This happens because in the KB there is no fact starting with "<Danny Boyle> <born". Subsequently, when generating t_{k+7}, the constrained generation mechanism guides the model to select "9" that leads to the correct birth year of Danny Boyle "<Danny Boyle> <born> <1956", avoiding the model to generate "<Danny Boyle> <born> <16" or "<Danny Boyle> <born> <196", both leading to incorrect information. This mechanism is additionally improved by beam search [12] which allows the model to explore multiple paths in the prefix tree in parallel.

3.2 Scaling to 800 Million Facts from Wikidata

We use the Wikidata truthy dump (Wikidata's highest-confidence statements, excluding qualifiers[10]) of the triples[11]. To filter out uninformative facts we retain only triples whose subject and relation have Wikidata identifiers. For the object,

[9] We use Hugging Face Transformers LogitsProcessor to alter the log-probabilities.
[10] https://www.wikidata.org/wiki/Wikidata:Database_download/en#RDF_dumps
[11] We use the dump from 11 December 2024.

we allow entities with Wikidata identifiers, English literals, numbers, dates, and literals with no language. Then, for each entity, we obtain a meaningful textual label (as Wikidata IDs lack meaning for LLMs) corresponding to the Wikipedia title if the entity is described in English Wikipedia[12]. Otherwise we use the template `rdf-schema#label (schema.org/description)`, e.g., "Jane Hajduk (American actress)", since the label alone is not unique. With this process we obtain more than 800 million facts like the ones underlined in Figure 1.

Now we tokenize the facts and we index them in a prefix tree for efficient access. Additionally, while creating the tree, we calculate the number of reachable leaves from each node to avoid that LLMs generate the same fact repeatedly, a behavior we witnessed during preliminary experiments.

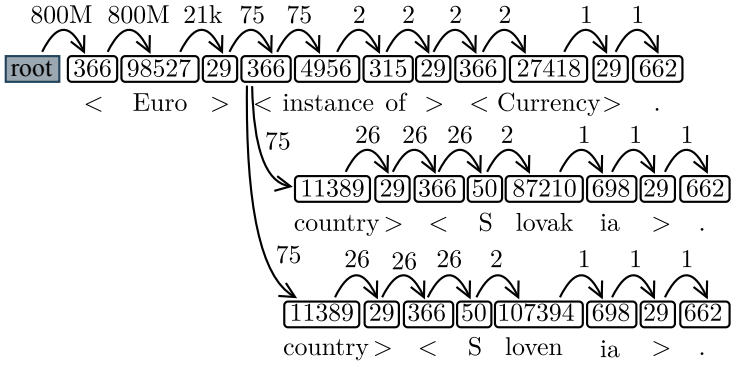

Fig. 3. Generating facts from the fact tree. Token ids are surrounded by rounded rectangles. The arrows $S \xrightarrow{n} t_{k+1}$ represent the selection of the next token t_{k+1} from the current sequence $S = t_0..t_k$ and n is the number of leaves reachable from S.

Duplicate fact generation is prevented as follows. Consider the prefix X = "<Euro> <country> <S" in Figure 3, where the number or reachable leaves is $numleaves(X) = 2$. After generating "<Euro> <country> <Slovakia> .", this number is decreased to 1 and from X the model is only allowed to select "lovenia> .". At this point, all leaves reachable from X have been generated, thus generating "S" from X' = "<Euro> <country> <" is forbidden by constrained generation. From X' the LLM can generate one of the remaining $numleaves(X') - 2 = 24$ facts (e.g., "<Euro> <country> <Italy> .").

While relatively small trees can be kept in memory, e.g., using Python dictionaries, when dealing with bigger trees, such as the 800-million-fact tree derived from Wikidata, this is often impossible. Therefore, we rely on solid relational database software. We use PostgreSQL[13] (version 17.2) and design the fact-tree table to ensure that all the information required for token generation is efficiently

[12] https://en.wikipedia.org/
[13] https://www.postgresql.org/

available. Given a prefix X, this includes the allowed next tokens $T = t^0..t^n$, the number of leaves reachable from X, and the number of reachable leaves if choosing t, for each $t \in T$ (to avoid selecting 0-leaves tokens and generating a fact twice). In Table 1, they are depicted respectively as *Next Tokens* and *#Lv. (number of leaves)*, and *Child.#Lv.*.

However, this tree-representation schema can quickly grow in disk space usage, as a single path of length L requires storing $L - 1$ rows. To reduce the disk space requirements, we apply two additional measures. First, after a certain prefix length L_c we stop adding rows and instead directly save the Python subtree in Pickle format[14], as shown in the last row of Table 1. In this way, during inference, subtrees that are manageable in size are directly loaded in memory (e.g., with $L_c = 7$, 99% of the subtrees use less than 116 KB). Secondly, we represent single-leaf sequences in a single row, as visible at the fourth row in Table 1, saving the rest of the sequence in *Child.#Lv.* (the token 29 comes after 694, and 662 is the last in the sequence). List items are saved as PostgreSQL arrays, while Pickle data is saved in *BYTEA*. For fast access, the *Prefix* is indexed with a B-Tree[15] index that provides logarithmic search time [6].

The ingestion process consists of creating the tree in-memory and then persisting it in the database. Nevertheless, hardware memory limits do not allow keeping the entire tree in-memory. So we perform this process in batches, constructing a limited-size tree, ingesting it, and then proceeding with the next one. This method, however, results in duplicated prefix rows, as identical prefixes from different batches are stored as different rows and must be merged during inference. The maximum number of duplicates corresponds to the number of batches used. With this mechanisms we are able to index the 800 million facts using 95 GB. Note that the index, storing token ids, depends on the LLM vocabulary, thus LLMs using distinct vocabularies require separate indexes.

At the implementation level, the functions NT_{KB} and *numleaves* access the prefix tree to obtain, respectively, the set of allowed next tokens and the number of reachable leaves for a given prefix. This information is acquired either through a SQL query to the database or from the in-memory subtree when the prefix length exceeds L_c.

Table 1. PostgreSQL table content. *#Lv.* stands for "number of leaves".

Prefix	**Next Tokens**	**#Lv.**	**Child.#Lv.**	**Subtree**
{root}	{366,1134,...}	5M	{5M,3,...}	
{root}	{366,8730,...}	5M	{5M,9,...}	
{root,366}	{537,7350,...}	2M	{2M,7,...}	
{root,694}	{29,...,662}	1	{}	
{root,366,...}	{21538,4168}	7	{4,3}	\x804

[14] https://docs.python.org/3/library/pickle.html
[15] https://www.postgresql.org/docs/current/btree.html

3.3 Embedding ReFactX into Question-Answering Workflows

You are a helpful question-answering assistant that bases its answers on facts from a knowledge base and always respects the prompt.

The process to answer questions:

- You receive an input question.
- You determine the reasoning path needed to answer the question based on the information available.
- You determine the kind of answer you are asked. It can be a yes/no, a single entity, or a list of entities. Pay attention to the questions whose answer is a list of entities (e.g. Which countries share a border with Spain?): you need to find all the answer entities and include them all in the final answer.
- You get relevant facts with the **"Fact:"** command. You can rely on these facts and use them a proof for your answer. While getting facts you continue the reasoning explaining it step by step.
- Often description or short description may be useful for answering questions.
- You conclude with a concise answer that depending on the question can be a yes/no, a single entity, or a list of entities. Pay attention to the questions whose answer is a list of entities.
- The answer MUST be based on the proofs you found with **"Fact:"**.

If you didn't find proofs with **"Fact:"** that support an answer you stop and you reply: "I don't know.".

If the question requires to find proof that an event happen and you didn't find any proof, you can assume that event didn't happen.

In case you are taking too long for answering (e.g., you already generated ten facts that are not useful for the question), you stop and you answer based on the proofs you acquired to that point.

You must always follow these instructions precisely and ensure your responses adhere strictly to this prompt.

Fig. 4. System prompt for the Wikidata KB. The prompt instructs the LLM to reason step-by-step, issue *Fact* calls to access facts from the Wikidata fact tree, and deliver an answer only after evidence is gathered—otherwise respond "I don't know."

ReFactX relies on In-Context Learning (ICL) [7] for instructing the model to access external knowledge. Our prompt, shown in Figure 4, instructs the LLM to, first, determine the reasoning path needed for answering, then to use the *Fact* command for getting relevant facts from the KB, and finally to answer based on the proofs acquired with the *Fact* command. Additionally, we instruct

the model to determine which kind of answer is required, to respond "I don't know" when no useful facts have been found, to understand when it is not able to find useful facts and stop, to strictly adhere to the prompt, and to be aware of the description predicate (often useful with the Wikidata KB). At the end of this prompt, we add two examples to better guide the model behavior. While two examples are not sufficient to represent all the possible types of questions, increasing the number of few-shot example may reduce the efficiency of our approach. Therefore, a two-shot prompt is used throughout this paper.

During inference, we detect when the LLM generates the sequence of tokens corresponding to the *Fact* command and we activate constrained generation; at this point, the model is forced to generate an existing fact. After an entire fact is generated, we switch the model back to normal generation, allowing it to continue reasoning, to call again the *Fact* command, or to answer. An example of ReFactX in action is illustrated in Figure 1.

4 Experimental Setup

Underlying Models We evaluate ReFactX on the QA task with the following models: *meta-llama/Llama-3.3-70B-Instruct* [21], *microsoft/phi-4* [1], *Qwen/Qwen2.5-72B-Instruct*, and *Qwen/Qwen2.5-7B-Instruct* [21] with the 800-million-fact tree derived from Wikidata indexed in PostgreSQL (see Section 3.2).

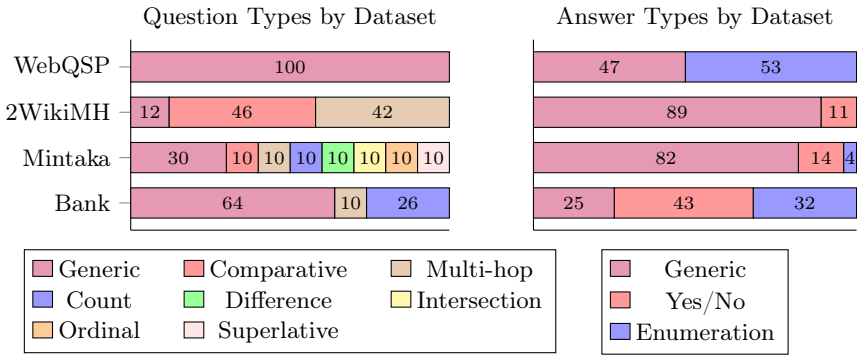

Fig. 5. Question and answer type distribution across the four evaluation datasets. Stacked bars show how each benchmark (Bank, Mintaka, 2WikiMH, WebQSP) varies in its mix of question categories (generic, comparative, multi-hop, and others) and answer forms (generic, yes/no, enumeration).

Datasets We evaluate our approach on three public benchmark datasets—Mintaka [24], 2WikiMultiHopQA [10], WebQSP [32]—and on an anonymized proprietary financial dataset, referred to as *Bank*. Mintaka [24] is a multilingual

dataset containing nine question types annotated by crowdworkers with Wikidata IDs; in this work, we only consider eight question types, and only English questions. This dataset allows us to analyze ReFactX's performance across diverse question types. We treat *Yes/No*—which is considered a question type in Mintaka—as an answer type, alongside *Generic* and *Enumeration*. Figure 5 shows the distribution of questions and answer types across all datasets.

2WikiMultiHopQA [10], to which we refer as 2WikiMH, is composed of multi-hop, comparative, and generic questions derived from Wikipedia and Wikidata. For this dataset, we evaluate on the *dev* set, as the ground truth for the test set is not publicly available. WebQSP [32] contains generic questions annotated with Freebase [3]. These two dataset are used by the existing KE-QA approaches based on constrained generation [17,19], providing insights into ReFactX's performance relative to prior work. For computational efficiency, we consider a limited sample of 200 questions for each public dataset, stratifying by question type.

The proprietary *Bank* dataset comes from Banca d'Italia[16], a large European central bank. It covers the financial domain and allows us to study how ReFactX adapts to domain-specific data. It includes 278 template questions derived from an anonymized corporate knowledge graph containing approximately 10,000 triples and nine relations, such as ownership and control. In this setting, we can evaluate LLMs while minimizing data contamination issues, as the models are unlikely to have been exposed to this data during training. We verbalize the triples, also adding inverted facts, so that ReFactX can use tail-to-head reasoning. Finally we build an in-memory prefix-tree index. Since in this dataset each relation is considered independent from the other ones, we add specific instructions in the prompt to consider each relation independently.

The choice of public benchmark datasets reflects two main considerations. First, we prioritized datasets that natively use only or also include Wikidata—namely Mintaka and 2WikiMH. Second, we included WebQSP, based on Freebase, to extend the comparison with prior work on constrained generation [17,19]. Additionally, we include the proprietary *Bank* dataset to evaluate on data free from any data contamination.

Metrics We evaluate on *Accuracy* (A) and *Precision* (P), to study, respectively, the overall correctness of our approach and its reliability when it provides an answer.

$$A = \frac{|\text{Correct Answers}|}{|\text{Questions}|}, \qquad (6a) \qquad P = \frac{|\text{Correct Answers}|}{|\text{Given Answers}|}. \qquad (6b)$$

Accuracy (A) corresponds to the ratio of questions answered correctly. Precision (P), instead, is normalized on the number of given answers, excluding "I don't know" and answers not given when the allowed maximum number of new tokens have been reached.

Precision and accuracy are calculated in two settings: (i) Exact Match: comparing the predicted answer with the ground truth with case-insensitive string equality; (ii) LLM-as-a-Judge [33]: we ask Llama3.3-70B in 16-bit precision

[16] https://www.bancaditalia.it/

whether the predicted answer is correct and complete with respect to the ground truth.

Reference Approaches We first evaluate the same LLMs used with ReFactX without constrained generation. In this setting, referred as *LLM-only*, we use the same settings and prompts, so that models generates not-grounded facts based solely on their parametric-knowledge, giving us clues on how important is to access external knowledge and, furthermore, on how much each dataset can be answered using only the LLM's internal knowledge.

Then, we compare against KE-QA methods that use constrained generation, specifically Decoding on Graphs (DoG) [17] and Graph-Constrained Reasoning (GCR) [19]. Additionally, we include Hybrid-QA (HQA) [15]—an input-based KE-QA approach. For these approaches, we report the results from the respective papers. HQA is evaluated on a 200-question sample of Mintaka, DoG and GCR evaluate on larger subsets of 2WikiMH and WebQSP: for 2WikiMH DoG uses 6,964 questions; for WebQSP respectively DoG and GCR consider 1,542 and 1,628 questions (filtered during preprocessing).

HQA [15] achieves state-of-the-art results on Mintaka [24]. Given a question, HQA first selects a limited set of few-shot examples maximizing their relevance for the question and the diversity between the examples, then, via ICL [7] with the selected few-shot examples, instructs the LLM to acquire external information with tools, such as a Wikipedia search engine and a Wikidata SPARQL query engine, and finally answers the question.

DoG [17] and GCR [19] are both based on constrained generation, while they consider smaller question-based prefix trees with respect to ReFactX. DoG [17], incorporates constrained generated triples from a KG inside a reasoning process similar to CoT [28], instructing the model with ICL [7], and alternating normal and constrained generation. The KG, composed of up to 120 triples, is constructed for each question, from the triples within 2-4 hops from question entities. Then, at inference time, a query-centric subgraph of the KG is obtained from the triples containing the question entities (extracted with an entity linking system). The model is allowed to generate only the triples from this query-centric subgraph, and at each generation, the subgraph is expanded with all the adjacent triples, allowing the model to form a reasoning path from question entities to the answer.

GCR [19], instead, considers a pre-built subgraph of Freebase [3] of 8 million triples containing the entities mentioned in the evaluation questions, then for each questions it constructs a smaller prefix tree from the paths obtained with breadth first search within 2-hops from question entities (obtained with entity linking). At this point an LLM, fine-tuned for the task, generates multiple reasoning paths from the prefix tree with constrained decoding and an answer hypothesis for each path. Finally, a powerful LLM, such as GPT-4o-mini, receives all the paths and the hypotheses answers and gives the final answer.

DoG and HQA use accuracy in the evaluation, while GCR calculates *Hit* and *F1*. Considering *enumeration* answer (containing a list of items), *Hit* counts

a prediction as correct whenever any item of the ground truth matches the prediction, whereas *F1* is the average of the *F1* of the single predictions. These measures are equivalent to accuracy for generic answers (not for enumerations).

In our experiments, we use beam search [12] with number of beams = 3 (GCR uses 10 beams, DoG 3), disable sampling, and set the maximum number of new tokens to generate to 1,000.

5 Results and Discussion

Generation-Time Overhead We compare ReFactX's KB-guided constrained generation time with unconstrained *LLM-only* generation time. Figure 6 plots the token generation times calculated in the two settings. The time overhead added by constrained generation is very limited: the total time for generating 4,000 tokens increases by only 1.3%, from 300.14 seconds to 303.89.

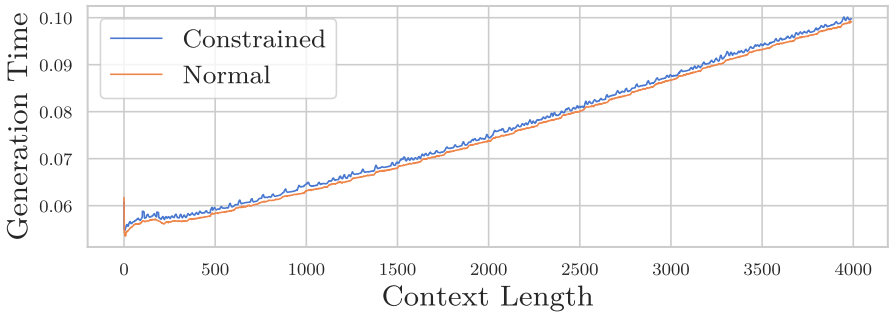

Fig. 6. Measured token generation time (in seconds). Results on 4,000 tokens, using Qwen2.5-3B with PostgreSQL running on the same machine, with key-value caching enabled, using one NVIDIA Tesla T4. We apply a moving average with a 10-tokens window to reduce noise.

Performance Analysis Table 2 shows ReFactX results on benchmark datasets, compared to *LLM-only*. The gap between Exact Match and LLM-as-a-Judge results likely derives from string comparison: correct answers with different date formats or item orders are counted as errors under Exact Match.

Compared to *LLM-only* models, ReFactX consistently achieves higher precision in the LLM-as-a-Judge evaluation. In terms of accuracy, ReFactX performs worse than *LLM-only* on the Mintaka and WebQSP datasets. This reveals these datasets are largely covered by the models' parametric knowledge—reaching 82.0% and 80.5% accuracy respectively. Instead with 2WikiMH, whose questions seem harder for the LLM parametric knowledge, ReFactX improves both accuracy and precision by more than 20% with respect to *LLM-only* models.

Table 2. Comparison between ReFactX and *LLM-only* across four datasets, considering four different LLMs (details in Section 4). We report Precision (P) and Accuracy (A), calculated using Exact Match (left) and LLM-as-a-Judge (right).

	Exact Match								**LLM-as-a-Judge**							
	Bank		Mintaka		2WikiMH		WebQSP		Bank		Mintaka		2WikiMH		WebQSP	
	P	A	P	A	P	A	P	A	P	A	P	A	P	A	P	A
ReFactX																
$\text{Llama}^{3.3}_{70B}$	**40.8**	36.0	**66.4**	40.5	**74.4**	64.0	**22.8**	17.0	50.0	**42.8**	**91.8**	56.0	93.6	81.0	85.2	63.5
$\text{Qwen}^{2.5}_{72B}$	39.9	**37.1**	43.8	28.0	71.9	**69.0**	18.5	14.0	46.1	**42.8**	82.8	55.5	**96.4**	**92.5**	84.8	65.5
Phi^{4}_{14B}	35.8	29.9	33.0	19.0	62.2	46.0	15.4	10.5	**51.3**	41.7	88.7	51.0	92.6	69.5	**89.7**	61.0
$\text{Qwen}^{2.5}_{7B}$	24.7	20.9	48.5	24.5	58.5	46.5	17.0	12.0	44.8	35.3	78.2	39.5	91.2	73.0	78.7	55.5
LLM-only																
$\text{Llama}^{3.3}_{70B}$	23.1	18.3	60.7	**59.5**	46.7	43.0	19.9	**19.5**	23.1	18.3	83.7	**82.0**	61.4	56.5	82.1	**80.5**
$\text{Qwen}^{2.5}_{72B}$	30.0	29.9	52.3	51.5	35.7	35.5	13.6	13.5	30.0	29.9	81.2	80.0	45.2	45.0	73.7	73.0
Phi^{4}_{14B}	21.8	20.1	43.6	42.5	25.5	24.0	14.4	14.0	25.7	23.7	76.9	75.0	41.5	39.0	75.3	73.0
$\text{Qwen}^{2.5}_{7B}$	21.7	18.0	45.3	41.0	21.6	18.0	10.9	10.5	28.1	21.9	65.7	60.0	35.9	30.0	66.7	64.0

Table 3. Insights from comparison with related work on Precision (P) and Accuracy (A). Results for related work are taken from their papers. ReFactX is evaluated using LLM-as-a-Judge†, HQA via manual annotation‡. DoG and GCR use exact match*. GCR results are calculated with different metrics§ (see Section 4).

	Mintaka		**2WikiMH**		**WebQSP**	
	P	A	P	A	P	A
ReFactX $\text{Llama}^{3.3}_{70B}{}^{\dagger}$	91.8	56.0	93.6	81.0	85.2	63.5
ReFactX $\text{Qwen}^{2.5}_{72B}{}^{\dagger}$	82.8	55.5	96.4	**92.5**	84.8	65.5
ReFactX $\text{Qwen}^{2.5}_{7B}{}^{\dagger}$	78.2	39.5	91.2	73.0	78.7	55.5
DoG[17]* $\text{Qwen}^{2.5}_{7B}$	–	–	–	84.2	–	**92.7**
GCR[19]* $\text{Llama}^{3.1}_{8B}$ + $\text{GPT}^{\text{4o-mini}}$	–	–	–	–	92.2§	74.1§
HQA[15] $\text{GPT}^{3.5\ddagger}$	–	85.9	–	–	–	–
HQA[15] $\text{GPT}^{4\ddagger}$	–	**95.9**	–	–	–	–

The Bank dataset proved especially challenging, with ReFactX achieving precision up to 51.3% and accuracy up to 42.8%. However, a type-wise analysis reveals good results on generic Yes/No questions achieving precision (P) of 85.2% and accuracy (A) of 78.9 (using LLM-as-a-Judge) with Qwen2.5-72B while with Llama3.3-70B we achieve $P = 77.8\%$ and $A = 70.6\%$. Our approach is, instead, particularly suffering with count questions: both models achieve less than 10% accuracy on these questions. Similarly, enumeration questions pose difficulties, with correct answers achieved in only 30% of cases on the Bank dataset.

On the Mintaka dataset, ReFactX struggles especially with superlative and count questions, with Llama3.3-70B achieving 14.3% and 55.0% accuracy, respectively. Qwen2.5-72B shows a contrasting pattern, achieving higher accuracy on superlatives (47.6%) but lower on counts (35.0%). On comparative and multi-hop questions, instead, Qwen2.5-72B achieves 52.4% and 57.9% accuracy, respec-

tively, while Llama3.3-70B reaches 66.7% and 73.7%, still below the performance levels in 2WikiMH, composed mostly by comparative and multi-hop questions.

On WebQSP, enumeration answers prove particularly difficult in terms of precision for ReFactX with both Llama3.3-70B and Qwen2.5-72B, while when using Phi-4 it maintains consistent precision across all answer types.

Results from Table 3, showing ReFactX with reference approaches, suggest our approach can achieve competitive results. In fact, compared to DoG on 2WikiMH—although evaluated on different dataset samples—ReFactX, even if considering a large KB, achieves 73.0% accuracy with Qwen2.5-7B and 92.5% with Qwen2.5-72B, while DoG stands in the middle with 84.2%.

Discussion The generation-time overhead measurements along with the results on the benchmark QA datasets demonstrate that LLMs with constrained generation and a prefix tree can effectively access external knowledge without any additional model, retriever, or external service. Additionally, by storing the prefix tree on disk using a database service, we are able to scale to a large knowledge base of 800 million facts derived from Wikidata. These advantages come with only a $\sim$1% increase in generation time.

In particular, ReFactX shows greater effectiveness on the 2WikiMH dataset. This can be explained by the nature of the dataset, which requires only simple answers and contains only generic, comparative, and multi-hop questions (see Figure 5), easier to address with our approach. Indeed, answering these question types generally requires fewer facts with respect to count or enumeration questions. While for Mintaka multi-hop questions, we notice that in some cases they require ordinal reasoning, such as *"Where was the 16th president of the United States born?"*, making them harder for ReFactX.

On Mintaka and WebQSP, the datasets widely covered by the internal knowledge of the tested LLMs, when comparing ReFactX behavior with *LLM-only* on the same questions, ReFactX still demonstrates interesting reasoning patterns. In some cases, it uses facts to prove an answer coming from LLM parametric knowledge. In others, e.g., for *"Where did Rick Santorum attend high school?"*, in which *LLM-only* fails, ReFactX is able to acquire correct facts which lead to correcting model parametric knowledge.

However, ReFactX has some intrinsic limitations. The main one derives from the autoregressive left-to-right nature of LLMs. Indeed, ReFactX requires left-to-right facts, that start with known-information and lead to desired information. Furthermore, while we believe ReFactX is particularly useful to access point-wise factual information, count questions like *"How many movies have been directed by Danny Boyle?"* or *"Which NHL team has the most Stanley Cup wins?"* are particularly challenging, because they require ReFactX to enumerate a large list of facts and to understand when all the required facts have been generated.

This limitation likely explains why our approach is obtaining worse performance on count and superlative question types. Such question types could be better handled by using additional tools like SPARQL engines, which sup-

port counts or other set operations, or by investigating mechanisms to control ReFactX generation similarly as we did for preventing fact repetition.

The obtained results are particularly promising considering that we did not fine-tune the models to improve their ability to leverage ReFactX during reasoning. We plan to address this limitation in future work.

6 Conclusion and Future Work

In this work, we presented ReFactX, a generic knowledge-base-constrained decoding system that can be integrated with any autoregressive LLM and large knowledge bases (KB) of textual facts. In our experiments, using a 800-million-fact KB derived from Wikidata, ReFactX injects evidence during generation via constrained decoding. This mechanism permits only continuations that lead to valid KB facts and is highly efficient. Leveraging a disk-backed prefix tree, it stores 800 million facts in 95 GB, adding only 1.3% latency at generation time. On certain benchmarks, it improves question answering accuracy with gains exceeding 20 percentage points at over 90% precision—compared to relying solely on the LLM's internal knowledge. ReFactX is thus a lightweight, easy-to-integrate solution for question answering with an external KB.

We plan to explore fine-tuning to enhance the reasoning capabilities of LLMs with ReFactX and to extend it to cover counts, long enumerations, and other set operations. To achieve higher accuracy in these cases, the model may first explore using ReFactX and then leverage additional tools—such as a SPARQL engine. We leave this extension for future work.

7 Supplemental Material Statement

Source code for using ReFactX and reproducing our work is available from GitHub at https://github.com/rpo19/ReFactX.

Acknowledgments

This work was supported by the German Federal Ministry of Education and Research (BMBF, SCADS22B) and the Saxon State Ministry for Science, Culture and Tourism (SMWK) by funding the competence center for Big Data and AI "ScaDS.AI Dresden/Leipzig". Additional support came from the EU Horizon Europe programme (grants No. 101189771—DataPACT and No. 101070284—enRichMyData), the European Community – Next Generation EU via the Italian Ministry of Justice, and the Italian PRIN project Discount Quality for Responsible Data Science (202248FWFS). The authors also acknowledge the computing resources provided by the NHR Center at TU Dresden, funded by the German Federal Ministry of Education and Research and participating state governments (www.nhr-verein.de/unsere-partner).

References

1. Abdin, M., Aneja, J., Behl, H., Bubeck, S., Eldan, R., Gunasekar, S., et al.: Phi-4 technical report (2024). https://doi.org/10.48550/arXiv.2412.08905, arXiv preprint arXiv:2412.08905
2. Bevilacqua, M., Ottaviano, G., Lewis, P., Yih, S., Riedel, S., Petroni, F.: Autoregressive search engines: Generating substrings as document identifiers. In: Advances in Neural Information Processing Systems. vol. 35, pp. 31668–31683. Curran Associates, Inc. (2022)
3. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. p. 1247–1250. SIGMOD '08, Association for Computing Machinery (2008). https://doi.org/10.1145/1376616.1376746
4. Cao, N.D., Izacard, G., Riedel, S., Petroni, F.: Autoregressive entity retrieval. In: International Conference on Learning Representations (2021)
5. Cheng, J., Marone, M., Weller, O., Lawrie, D., Khashabi, D., Durme, B.V.: Dated data: Tracing knowledge cutoffs in large language models. In: First Conference on Language Modeling (2024)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT press (2022)
7. Dong, Q., Li, L., Dai, D., Zheng, C., Ma, J., Li, R., et al.: A survey on in-context learning. In: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing. pp. 1107–1128 (2024). https://doi.org/10.18653/v1/2024.emnlp-main.64
8. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., Wang, H.: Retrieval-augmented generation for large language models: A survey (2024), arXiv preprint arXiv:2312.10997
9. Geng, S., Josifoski, M., Peyrard, M., West, R.: Grammar-constrained decoding for structured NLP tasks without finetuning. In: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. pp. 10932–10952. Association for Computational Linguistics (2023). https://doi.org/10.18653/v1/2023.emnlp-main.674
10. Ho, X., Duong Nguyen, A.K., Sugawara, S., Aizawa, A.: Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In: Proceedings of the 28th International Conference on Computational Linguistics. pp. 6609–6625. International Committee on Computational Linguistics (2020). https://doi.org/10.18653/v1/2020.coling-main.580
11. Hongkang Yang, H.Y., Zehao Lin, Z.L., Wenjin Wang, W.W., Hao Wu, H.W., Zhiyu Li, Z.L., Bo Tang, B.T., et al.: Memory^3: Language modeling with explicit memory. Journal of Machine Learning **3**(3), 300–346 (2024). https://doi.org/10.4208/jml.240708
12. Hu, X., Li, W., Lan, X., Wu, H., Wang, H.: Improved beam search with constrained softmax for NMT. In: Proceedings of Machine Translation Summit XV: Papers (2015)
13. Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., et al.: Dense passage retrieval for open-domain question answering. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 6769–6781. Association for Computational Linguistics (2020). https://doi.org/10.18653/v1/2020.emnlp-main.550

14. Lê, M., Fokkens, A.: Tackling error propagation through reinforcement learning: A case of greedy dependency parsing. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers. pp. 677–687 (2017)
15. Lehmann, J., Bhandiwad, D., Gattogi, P., Vahdati, S.: Beyond boundaries: A human-like approach for question answering over structured and unstructured information sources. Transactions of the Association for Computational Linguistics **12**, 786–802 (06 2024). https://doi.org/10.1162/tacl_a_00671
16. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., et al.: Retrieval-augmented generation for knowledge-intensive nlp tasks. In: Advances in Neural Information Processing Systems. vol. 33, pp. 9459–9474. Curran Associates, Inc. (2020)
17. Li, K., Zhang, T., Wu, X., Luo, H., Glass, J., Meng, H.: Decoding on graphs: Faithful and sound reasoning on knowledge graphs through generation of well-formed chains (2024). https://doi.org/10.48550/arXiv.2410.18415, arXiv preprint arXiv:2410.18415
18. Li, M., Zhao, Y., Deng, Y., Zhang, W., Li, S., Xie, W., et al.: Knowledge boundary of large language models: A survey (2024). https://doi.org/0.48550/arXiv.2412.12472, arXiv preprint arXiv:2412.12472
19. Luo, L., Zhao, Z., Gong, C., Haffari, G., Pan, S.: Graph-constrained reasoning: Faithful reasoning on knowledge graphs with large language models. In: Forty-second International Conference on Machine Learning (2025)
20. Maynez, J., Narayan, S., Bohnet, B., McDonald, R.: On faithfulness and factuality in abstractive summarization. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. pp. 1906–1919 (2020). https://doi.org/10.18653/v1/2020.acl-main.173
21. Qwen, Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., et al.: Qwen2.5 technical report (2025). https://doi.org/10.48550/arXiv.2412.15115, arXiv preprint arXiv:2412.15115
22. Schick, T., Dwivedi-Yu, J., Dessi, R., Raileanu, R., Lomeli, M., Hambro, E., et al.: Toolformer: Language models can teach themselves to use tools. In: Advances in Neural Information Processing Systems. vol. 36, pp. 68539–68551. Curran Associates, Inc. (2023)
23. Scholak, T., Schucher, N., Bahdanau, D.: PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. pp. 9895–9901. Association for Computational Linguistics (2021). https://doi.org/10.18653/v1/2021.emnlp-main.779
24. Sen, P., Aji, A.F., Saffari, A.: Mintaka: A complex, natural, and multilingual dataset for end-to-end question answering. In: Proceedings of the 29th International Conference on Computational Linguistics. pp. 1604–1619. International Committee on Computational Linguistics (2022)
25. Shuster, K., Poff, S., Chen, M., Kiela, D., Weston, J.: Retrieval augmentation reduces hallucination in conversation. In: Findings of the Association for Computational Linguistics: EMNLP 2021. pp. 3784–3803. Association for Computational Linguistics (2021). https://doi.org/10.18653/v1/2021.findings-emnlp.320
26. Sun, H., Qiao, Z., Guo, J., Fan, X., Hou, Y., Jiang, Y., et al.: Zerosearch: Incentivize the search capability of llms without searching (2025). https://doi.org/10.48550/arXiv.2505.04588, arXiv preprint arXiv:2505.04588

27. Thoppilan, R., Freitas, D.D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H.T., et al.: Lamda: Language models for dialog applications (2022). https://doi.org/10.48550/arXiv.2201.08239, arXiv preprint arXiv:2201.08239
28. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., et al.: Chain-of-thought prompting elicits reasoning in large language models. In: Advances in Neural Information Processing Systems. vol. 35, pp. 24824–24837. Curran Associates, Inc. (2022)
29. Wu, Y., Zhao, Y., Hu, B., Minervini, P., Stenetorp, P., Riedel, S.: An efficient memory-augmented transformer for knowledge-intensive NLP tasks. In: Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing. pp. 5184–5196. Association for Computational Linguistics (2022). https://doi.org/10.18653/v1/2022.emnlp-main.346
30. Xu, P., Ping, W., Wu, X., McAfee, L., Zhu, C., Liu, Z., et al.: Retrieval meets long context large language models. In: The Twelfth International Conference on Learning Representations (2023)
31. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: React: Synergizing reasoning and acting in language models (2023). https://doi.org/10.48550/arXiv.2210.03629, arXiv preprint arXiv:2210.03629
32. Yih, W.t., Richardson, M., Meek, C., Chang, M.W., Suh, J.: The value of semantic parse labeling for knowledge base question answering. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). pp. 201–206. Association for Computational Linguistics (2016). https://doi.org/10.18653/v1/P16-2033
33. Zheng, L., Chiang, W.L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., et al.: Judging llm-as-a-judge with mt-bench and chatbot arena. In: Advances in Neural Information Processing Systems. vol. 36, pp. 46595–46623. Curran Associates, Inc. (2023)
34. Zhou, J., Chen, L.: Openrag: Optimizing rag end-to-end via in-context retrieval learning (2025). https://doi.org/10.48550/arXiv.2503.08398, arXiv preprint arXiv:2503.08398