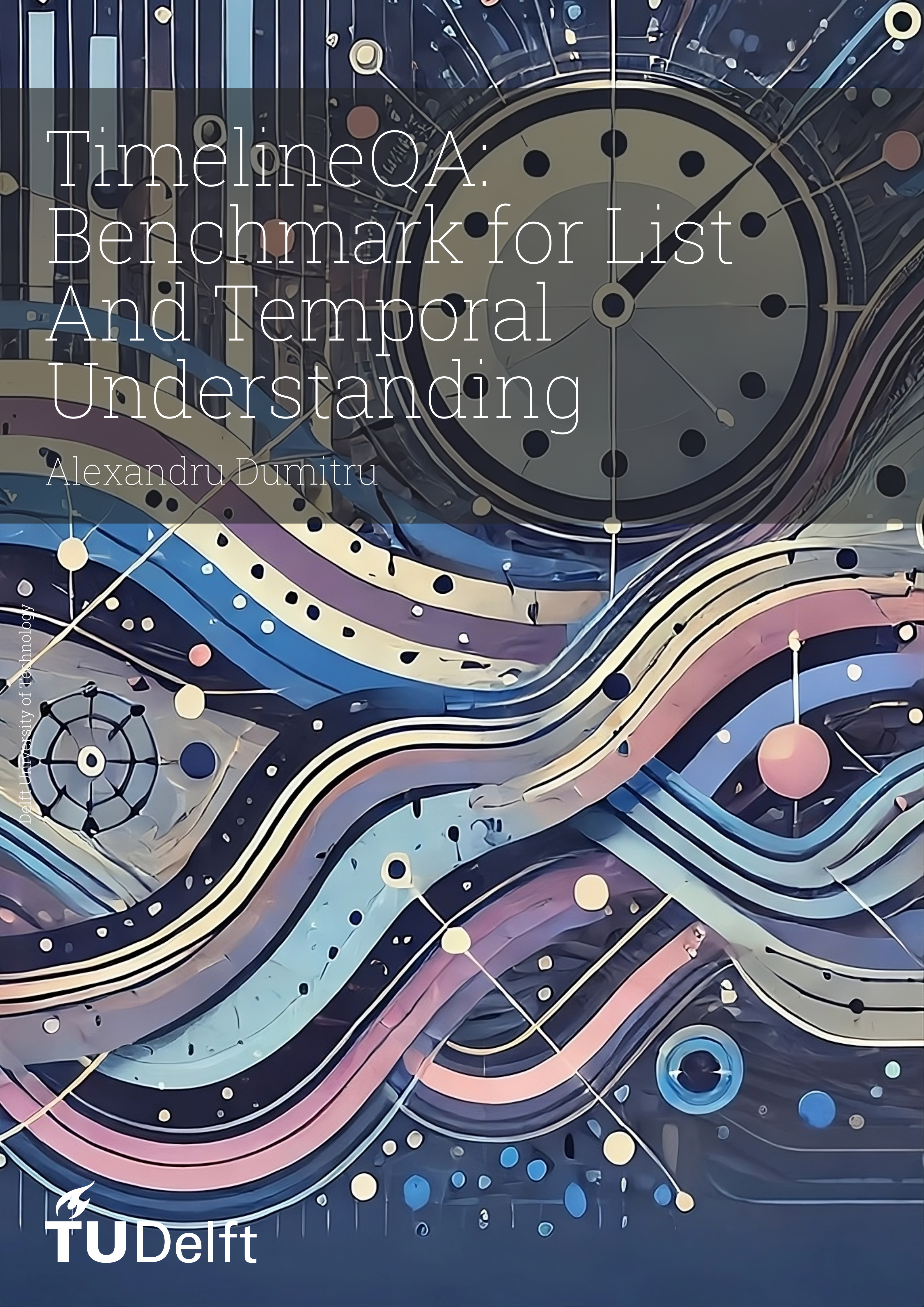


TimelineQA: Benchmark for List And Temporal Understanding

Alexandru Dumitru

Delft University of Technology

TU Delft

TimelineQA: Benchmark for List And Temporal Understanding

by

Alexandru Dumitru

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday July 3, 2024 at 10:30 AM.

Student number:	5074711	
Project duration:	November 24, 2023 – July 3, 2024	
Thesis committee:	Dr A. Anand,	TU Delft, supervisor
	Dr M. Gürel,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Thesis contents were enhanced with the assistance of ChatGPT
Cover Image was created using DALLE-3

TU Delft

Preface

With this thesis, I conclude my 5-year academic journey at TU Delft. During this timeframe, I had the pleasure to meet and interact with many wonderful people.

First and foremost, I would like to express my profound gratitude to my supervisor, Dr. Anand, for his unwavering guidance and advice throughout the thesis process. Moreover, assisting in his two courses, Information Retrieval and Natural Language Processing, was a truly enjoyable experience, and I am grateful for the opportunity.

Next, I would like to thank Venky for his consistent support throughout my thesis. He always provided valuable guidance and was incredibly responsive whenever I had questions or faced challenges. I will miss the Friday meetings, where we regularly discussed new ideas for my thesis.

Additionally, I would like to thank the members of the ELIXIR group for all the interesting talks, presentations, and meetings.

I would also like to thank Chelsea for her unconditional emotional support and the great memories we have shared during our two years of master's studies. I look forward to creating even more in the future.

Special thanks to Bobe, Bogdan, Matteo, and Roy for all the fun activities and lunches we shared. I hope we can continue these in the future as well.

Last but not least, I want to express my deepest gratitude to my family back home. Their unconditional support has always motivated me to keep pushing forward.

Alexandru Dumitru
Delft, June 2024

Abstract

Multiple benchmarks for question answering (QA) systems often under-represent questions that require lists to be answered, referred to in this work as ListQA. This type of question can provide valuable insights into the system's ability to structure its internal knowledge. In this work, we introduce TimelineQA (TLQA), a specialized subset of ListQA that requires both list comprehension and temporal understanding to generate correct answers. We automatically curate a benchmark for TLQA and test it against generative large language models (LLMs) commonly employed in QA systems. Our findings reveal significant shortcomings in current models, particularly in their inability to provide complete answers and accurately align facts temporally. To address these issues, we explore how Retrieval Augmented Generation (RAG) can enhance performance in this task. Our results indicate that while RAG improves performance, especially in retrieving relevant contextual information, there remains considerable room for further improvement. Both the benchmark and the evaluation of the models can be publicly accessed here.

Contents

1

Introduction

Question Answering (QA) is an important and well-explored task [1, 2, 3] that helps address precise information needs of users. Instead of showing a long list of snippets or hyperlinks for a query, QA systems enhance user experience by directly providing an answer to a query with supporting evidence [4]. QA systems are increasingly utilized across various fields, including healthcare, education, and other areas where precise and accurate information is needed [5]. These systems are designed to cater to a wide spectrum of queries, generating answers in forms ranging from single word replies to detailed passages and structured lists.

The significance of questions that require list based answers: Although, numerous solutions for QA systems have been developed, there has been a lack of focus on queries that require structured, listwise answers. Prior works have largely focused on factoid queries or questions with only simple or one possible answer [6]. These questions can be solved, often, by retrieving a small set of sentences from a single evidence source. However, solving questions with multiple possible answers or answers over a timeframe require reasoning over multiple sources, structuring the answers accordingly and ensuring completeness of the final output. In this work, question with list based answers are denoted as **ListQA**.

The ListQA task may include inquiries referring to side effects of certain medication, historical events, priority lists or rankings. In such situations, mistakes can carry considerable risk, such as incorrectly including or omitting the name of a symptom in responses to a healthcare-related question. Furthermore, ListQAs are notably frequent in certain domains. Yoon et al. [7] illustrate this within the biomedical field, where they found that **21.9%** of the queries necessitate answers in a list format. In contrast, only **7%** of naturally posed queries are listwise. Moreover, Chakrabarti et al. [8] report that over 10% of Bing's web queries fall into two types, either superlative or list-of-entity. This reflects the prevalence of ListQAs in web browsing. Therefore, the significant occurrence of ListQAs underscores their importance for study. Advancing ListQA technology can enhance user experience by presenting information in a more structured manner, and improve thoroughness of the response by providing comprehensive lists.

Example: TLQA

Question: *List all sports teams Robert Lewandowski played for from 2010 to 2020.*
[Answer]: Lech Poznań (2010)
Borussia Dortmund (2010-2014)
FC Bayern Munich (2014-2020)

Figure 1.1: An example of question for Timeline based QA

The significance of answering Timeline based questions: In this work, we plan to study timeline based ListQA, where questions require a list of answers over a time period. This reflects real-world queries where users are interested in querying historical events or news [9]. These are common queries issued to search engines which are time-sensitive or involve temporal specifiers [10, 11] such as “What teams did Lebron James play for between 2007-2009?”. These queries are complex to resolve as they need to not only maintain completeness of the answer list but also get accurate estimates of the related time periods. While existing works have focused on questions with temporal markers, they were primarily based on Knowledge Graphs (KG) [11, 12, 13, 14, 15] and not text-based, which limits the understanding of temporal evolution and transitions in text. Additionally, these works do not address questions that require multiple answers, which is common in real-world scenarios involving time-period-specific queries. Hence, in this work, we focus on text based questions that are time-sensitive in nature and require an ordered list as output according to relevant time periods. An example of a question that requires list based answers along with related time periods is shown in Figure 1. We observe that the player has played for 3 different teams over different time periods. Questions similar to the example help evaluate the capability of generative models to track time evolution of facts and answer accurately based on world knowledge encoded in their weights.

Limitations of existing QA benchmarks: Unlike typical QA scenarios such as those in SQuAD [16] where the system extracts a single answer from a single body of text, TLQA systems must find multiple answers from a single text or multiple sources [17] with appropriate time periods. For instance, to respond to the search ”coastal cities in the US that won the green city award between 2007-2010”, the system would have to look in passages about coastal cities in the US [8] as well as cities that won the said award in the specified time frame. This need to not only identify multiple relevant pieces of information, but also to understand how they are interconnected to form a cohesive list increases the complexity of timeline based ListQAs. Another challenge that occurs when dealing with TLQA task is maintaining contextual and relational integrity, especially when dealing with composite queries such as “Who composed film scores for movies written by Robert Alan Aurthur between 1950 to 1958?” [18]. In addition, there is the challenge of list ordering and structuring, relevant in the case of ordered lists such as ranks or guides. Furthermore, another issue is handling list-specific ambiguities, which involves determining the breadth or depth of the list when dealing with vague terms like ”popular” or ”major”.

Challenges of creating benchmark for TLQA task: Finally, the creation of relevant benchmark datasets to evaluate timeline and list based QA represents another complex task. The role of benchmarks in this case is to evaluate how different QA systems perform given the same queries. Establishing well-structured benchmarks is essential in TLQA, as they are under-explored and play an important role in assessing the effectiveness of models, as well as in identifying gaps in performance. Creating such datasets may involve manual sampling, which refers to humans handpicking data examples, or automatic sampling, which implies automating the process through different means. While manual sampling is possible, it is often expensive and time-consuming due to the intricate nature of list questions and the need for comprehensive answer sets. Automatic sampling also faces challenges. Such challenges include selecting semantically correlated candidate answers to avoid producing overly broad or trivial questions, and ensuring that the answer sets are both accurate and complete [19]. The benchmarks should also be formulated in a way as to test the temporal reasoning capabilities of the models under consideration, in addition to their ability to produce complete lists without omission.

The TLQA Benchmark:

We create and release the Timeline List QA (TLQA) benchmark for timeline based list question answering. All the questions in our benchmark necessitate a list based answer. Each answer consists of two components, namely the temporal range component and the answer entities. We also collect and release a **large collection** of Wikipedia articles and corresponding info boxes. Hence, we also formulate evaluation metrics that check for *temporal correctness* and completeness in terms of *coverage of entities* of all answers. We evaluate a range of generative models in few-shot closed book setup and in open-domain setup. Since complex Question Answering is a knowledge intensive task, relying only on model parametric knowledge may not be sufficient. While Large Language Models (LLMs) [20] have demonstrated impressive capabilities on a wide range of tasks, they still suffer from hallucinations [21] where the generative model generates unsubstantiated outputs due to lack of sufficient grounding and limited parametric knowledge. Hence, generative models are usually augmented with context from knowledge sources. This is recently further popularized by Retrieval Augmented

Generation (RAG) frameworks [22, 23]. Since retrieval from open-domain corpus is a challenging task, we evaluate a range of pre-trained retrieval models and demonstrate the performance of generative models employing the resulting context.

In the closed-book setup, the highest performance is achieved by Meta Llama 3, with a listwise F1 score of 0.442 and a Temporal Overlap score of 0.519 using KNN-3shot prompting. Mistral 0.2 also reaches its best performance with the same prompting technique, scoring a listwise F1 of 0.366 and a Temporal Overlap of 0.435. In the open-book setup, augmenting context with BM25 significantly increases model performance. Mistral 0.2 combined with BM25 achieves a listwise F1 score of 0.624 and a Temporal Overlap of 0.651. However, dense retrieval provides only a modest improvement, with a maximum listwise F1 score of 0.441 and a Temporal Overlap of 0.436. This gap can be attributed to the difference in retrieve performance over our corpus. Specifically, BM25 achieves a Mean Reciprocal Rank (MRR) of 0.479, compared to 0.384 for Multi QA, indicating approximately a 20% performance increase.

Research Questions We plan to answer the following research questions through our work:

- **RQ1:** How well do large generative models that are believed to encode world knowledge perform on the task of Timeline based List QA (TLQA) ? What are the common pitfalls when employing such models for TLQA tasks ?
- **RQ2**: To what extent can different paradigms, such as Retrieval Augmented Generation (RAG) improve the performance of LLMs on TLQA tasks?
- **RQ3**: When provided with golden evidence, what are the effects of various distractors on model performance in TLQA tasks?

Contributions: We identify the following contribution from our work:

- A benchmark of timeline based questions that require list based answers. This would help to evaluate the temporal and set based reasoning capabilities of Large generative models that are believed to encode world knowledge.
- Evaluation of large generative models on TLQA task and identification of gaps in their reasoning capabilities on such questions.
- Improving the robustness of large generative models on such questions by infusing temporal reasoning capabilities and relevant context through RAG (Retrieval Augmented Generation) based approaches.

2

Related Works

2.1. Temporal QA Benchmarks

Time is an important aspect in queries issued to answering engines [10]. Information evolves over time [24, 25], and it is critical to provide temporally relevant information to users. Hence, temporal information retrieval [26, 27] and temporal QA [15, 28] has been of immense interest to the research community. These works have focused on queries that have temporal markers or signals or questions that involve temporal signals in the answers. A large body of studies have been dedicated to study evolution of facts through temporal knowledge graphs (TKG) [12, 11, 14, 29]. These works usually involve constructing fact triplets with temporal information.

To exploit the value of temporal information in TKGs, authors have explored the possibility of answering natural language questions over them [30] by releasing a large dataset named CRONQUESTIONS comprising temporal questions. However, [31] observed that such datasets consists of primarily pseudo-temporal questions where the questions could be answered using facts without enforcing temporal constraints. For instance, to answer the question “What award did Carlo Taverna receive in 1863?”, there is no necessity to enforce the temporal constraint as Carlo received only one award. To tackle this [13], proposed a new dataset namely MultiTQ which is relevant to more than one fact triplet and also mandates temporal constraint to be satisfied for answering the question.

However, answering questions over temporal knowledge graphs are limited to the facts contained in the constructed knowledge graph. Additionally, the analysis is limited to transition of facts with time. To study temporal evolution of facts in natural text, the authors [10] proposed a dataset for answering time sensitive questions with textual evidence. The authors accomplish this by collecting time-evolving facts from Wikipedia and mapping them to relevant Wikipedia text. then questions are generated from the resulting collection. Further works have also explored answering time-sensitive questions in natural language [32].

While above works explore temporal QA, they do not study in detail the ability of answering engines to reason over time. Additionally most of these works assume questions have only one answer and do not consider the possibility of multiple answers over time periods. Hence, in our work we propose to curate a benchmark that necessitates answers that are lists which are structured according to related time periods.

2.2. ListQA benchmarks Construction

While the scenario of timeline based list QA has been relatively under-explored, several efforts have been made to create benchmarks which involve questions with list based answers. Both automatic and manual methodologies can be observed in the development of ListQA benchmarks. By manual methodologies, we refer to processes that involve human intervention, like rephrasing or validation, whereas automatic methodologies rely on computational procedures to complete tasks like question generation.

2.2.1. Fully Manual Methodologies

Benchmarks exclusively developed through manual efforts are relatively rare due to the extensive time and resources required. However, notable instances do exist. One such example is **BioASQ** [33], an annual challenge that assesses QA and information retrieval systems in the medical domain. In one of its tasks, the challenge evaluates models across various question types, including list-based questions. The latest edition of this challenge featured 967 questions categorized as list questions. These questions in BioASQ are manually curated, drawing from a wide range of biomedical literature and databases using experts' feedback as guidance.

2.2.2. Fully Automated Methodologies

Fully automated methodologies in the creation of ListQA benchmarks rely exclusively on computational procedures without human intervention. This approach significantly accelerates the dataset generation process and can handle large volumes of data efficiently. Two notable contributions in this area are the LIQUID framework [19] and the application of Set Expansion (SE) in QA systems [34].

The LIQUID framework describes how ListQA datasets can be generated from unlabeled corpora, such as Wikipedia or PubMed. The process consists of 4 stages: answer extraction, question generation, iterative filtering and answer expansion. Initially, answers are extracted by applying named entity recognition to summarized passages, enhancing the likelihood of selecting correlated candidate answers and avoiding trivial questions. Subsequently, a Question Generator (QC) model, fine-tuned for $< c, q, a >$ triplets, creates initial questions using a context and a concatenated set of answers. The third stage involves filtering incorrect answers through a validation process, using a QA model to iteratively remove answers with confidence scores below a set threshold, typically 0.1, ensuring better alignment between the question and the refined answers. The final stage expands the answer set, seeking additional answer spans with a confidence score surpassing the lowest score in the initial set.

While the study is older than the others presented, the work described by Wang et al. [34] showcases an idea which could still be applicable, namely the application of set expansion in the scope of ListQA. This process starts by selecting initial seed answers from a QA system. These seeds are then used to form queries for retrieving a wide array of documents through a search engine. The method applies a two-seed query technique, sending seed pairs to gather documents rich in contextual information. From these documents, the algorithm identifies potential new answers based on the proximity and co-occurrence of seeds within the text, employing context-based heuristics. Subsequently, in the refinement stage, a probabilistic model evaluates each new candidate answer for relevance. This model considers the frequency and patterns of co-occurrence with the seeds, filtering out less relevant answers, resulting in an expanded set finely tuned to closely match the intent and content of the original query.

2.2.3. Hybrid Methodologies

ComplexWebQuestions [35], **RoMQA** [36] and **Qampari** [18] are benchmarks that exemplify the use of both automatic and manual methodologies in ListQA dataset creation, utilizing knowledge bases for generating question entries and human workers for rephrasing.

ComplexWebQuestions starts with the WEBQUESTIONSSP dataset [37], which contains simple questions paired with SPARQL queries for Freebase. These queries are subsequently expanded, through four different types of operation, namely conjunctions, superlatives, comparatives and compositions. For conjunctions, the process involves traversing the knowledge base (KB) and adding SPARQL triplets to yield a valid subset of answers. Comparatives and superlatives are formed by finding a numerical property common to all elements in the answer set and adding a triplet and restrictor to the query. Compositions are created by replacing an entity in the original query with a variable and adding a triplet that defines the entity. Following the creation of these complex queries, the system then automatically generates understandable machine-generated questions, which are paraphrased into natural language by Amazon Mechanical Turk (AMT) workers.

In RoMQA, the dataset is constructed by first sampling constraints from Wikidata, transforming subject-proposition-object triples into entity-relation constraints. An example given in the paper is how the constraint "Gilbert Amy occupation pianist" would be transformed into "Gilbert Amy occupation obj" and "pianist occupation sub". These are then clustered based on shared answer entities, with clusters adjusted for uniformity by resampling constraints, particularly down-sampling over-represented ones like ”country”. Logical queries are generated from these clusters, creating up to five per cluster. In this process, constraints are either removed or negated with a probability of 0.1 and redundant constraints are eliminated. The resulting logical queries combine multiple constraints using conjunctions. Finally, these logical queries are transformed into natural language questions using AMT crowd-workers.

In Qampari, the dataset creation begins with generating questions with multiple answers from Wikipedia’s knowledge graph and tables, using Wikidata and Wikipedia tables. This involves formulating a question, identifying a set of relevant answers, and linking them to corresponding evidence passages from Wikipedia. The process starts with simple questions involving a single entity and relation, which are then expanded to create more complex questions using intersection and composition operations. After generating the questions and identifying evidence passages, a filtering process is applied, keeping only those queries with a high percentage of validated answers. The final phase involves two steps: correctness validation and paraphrasing. Crowdsourcing workers validate the correctness of each question-answer pair against the evidence passages. Additionally, they rephrase the questions, initially in pseudo-language, into natural language, ensuring they are understandable and relatable.

2.3. Language Models in QA systems

As stated previously, the development of Large Language Models have contributed to the improvement of QA systems. Models including GPT-4 [38], PaLM [39] and LLaMA [40], have demonstrated expertise in multiple NLP tasks, including QA. The performance of these models have been studied in different settings, including both specific and general domains. For example, Kamalloo et al. [41] showcase the performance of LLMs in an open-domain setting, by achieving state-of-the-art results on NQ-OPEN with InstructGPT, using more nuanced evaluation techniques, which include manual assessments. Moreover, in the medical field, studies such as [5] and [42] prove that LLMs are capable of adapting to domain-specific tasks, and perform considerably well.

Despite their capabilities, these models have been shown to suffer from hallucinations [43]. The term hallucination refers to the phenomena in which LLMs generate text that is nonsensical, unsubstantiated or inconsistent with the input text. Such behavior hinders the applicability of LLMs, especially in the context of QA, where factual answers are expected. In TLQA tasks, hallucinations could impact aspects such as the completeness and accuracy of responses, as LLMs might generate excessive or insufficient information in addition to incorrect information. Additionally, studies have shown that when LLM based dialogue systems are tasked with generating a long output text, the later part may contradict with the beginning [44]. This phenomenon may also appear in different cases of TLQAs. Despite the existence of studies focused on mitigating such hallucinations [44], a more thorough exploration into the performance of LLMs in settings similar to TLQAs remains a significant area for research, especially in terms of maintaining accuracy, completeness and consistency in longer responses. In TLQAs it is also challenging to maintain temporal accuracy as LLMs may make up dates or years due to popularity bias and other issues.

3 Benchmark Construction

In this chapter, we present the main methodology behind curating the TLQA benchmark. The goal of this benchmark is to evaluate the ability of generative language models to answer queries that require both list comprehension and temporal knowledge. We present our inspiration, data source and procedure to create the TLQA dataset.

3.1. TempLAMA

The dataset TempLAMA [45] was used as a building block to construct the benchmark for TLQA tasks. TempLAMA represents a benchmark to test the capability of language models to answer factual knowledge that varies over time. This dataset contains fill-in-the-blank or cloze style queries that depend on the time context specified in the query. TempLAMA was developed by first selecting facts from Wikidata with start or end dates after 2010 and up to 2020, ensuring the dataset covers recent temporal changes. The criteria for inclusion required that both the subjects and objects of these facts are linked to Wikipedia pages. The authors focused on identifying subject-relation pairs with objects that vary over time. To determine which relations to include, they analyzed the frequency of subjects associated with varying objects across different time periods. This analysis allowed them to identify nine relations with the highest number of subjects demonstrating temporal variation. These relations include “member of sports team”, “position held”, “employer”, “political party”, “head coach”, “educated at”, “chairperson”, “head of government” and “owned by”. For each of these relations, the authors wrote template cloze queries (such as in 3.1) and populated them with the 1,000 most frequent subjects per relation. For each subject and relation pair, all objects with their associated time intervals were gathered, and separate queries were constructed for each year in those intervals. When intervals overlapped, all valid objects for those years were included in the list of correct answers. This resulted in a dataset of 50,310 queries spanning 11 years, from 2010 to 2020. The goal of the benchmark is to showcase potential problems that language models may face when encoding temporal knowledge, such as averaging conflicting information from different time periods, where the model may have low confidence in any of the correct answers due to conflicting data. Another issue is forgetting, where models might not memorize facts valid during underrepresented periods of time, leading to poor performance on questions about the more distant past. Additionally, there is the problem of poor temporal calibration, where models become less accurate when predicting facts outside the temporal scope of their training data.

Year	Input	Target
	TempLAMA	
2012	Cristiano Ronaldo plays for _X_.	Real Madrid
2019	Cristiano Ronaldo plays for _X_.	Juventus FC

Table 3.1: Examples taken from TempLAMA [45]

3.2. Limitations of TempLAMA

TempLAMA has several notable limitations related to its evaluation method, reliance on Wikidata, and temporal depth.

Firstly, the evaluation method used on TempLAMA is limited. While the benchmark considers listwise answers during its creation by including multiple answers that overlap within the same year, the evaluation process relies on the maximum token-level F1 score between the predicted and ground truth answers for queries with multiple targets. This approach does not account for the completeness of the responses, as it only recognizes the highest-scoring answer, rather than evaluating whether all correct answers are provided. Our analysis shows that approximately **25.3%** of entries in TempLAMA have in their ground truth more than 1 entity. However, the current evaluation method does not adequately measure the language models' ability to return all correct answers. This limitation means that the evaluation does not fully capture the models' knowledge and accuracy in handling multiple valid responses, leading to an incomplete assessment of their performance.

Secondly, TempLAMA's reliance on Wikidata introduces certain artifacts into the benchmark. The authors acknowledge this issue in their limitations section, noting decisions such as assuming that a missing start date implies the fact is valid from the beginning of the time period of interest. Additionally, about 7.9% of the questions have subjects with only one word from their original fullname, which can be problematic when entities share names, such as 'Cristiano Ronaldo' and 'Ronaldo,' or when entities have generic names that require more specificity, such as 'Xavi.' These ambiguities and assumptions can affect the clarity and precision of the queries, potentially leading to inconsistent or ambiguous evaluation results.

Lastly, the temporal depth of TempLAMA is restricted to the period from 2010 to 2020. This limited historical range constrains the benchmark's ability to evaluate language models on their knowledge and reasoning over a broader historical context. A more extended temporal coverage would allow for a more comprehensive assessment of the models' capabilities to handle temporal knowledge spanning a wider range of years and ability to extrapolate to current time period.

These limitations suggest that while TempLAMA provides valuable insights into the temporal reasoning capabilities of language models, improvements in evaluation methods, data sources, and temporal coverage are necessary to achieve a more accurate and complete assessment.

3.3. Timeline QA

TempLAMA serves as inspiration for our TLQA generation, while keeping in mind its limitations mentioned previously. Before we detail the creation of the Timeline QA dataset, we formally define Timeline QA entries as follows:

Definition 3.3.1 *A timeline-based list question Q is a query that, given a subject entity s and a temporal context t, requests a comprehensive list of entities or facts e associated with s over the time period t. The answer A to Q is a set of pairs (e, τ) where e is an entity or fact, and τ is the time interval during which e is relevant to s.*

An example can be seen in Figure ... . We move from the fill-in-the-gaps query style of TempLAMA to a list-based query style, which requires models to return all relevant entities or facts over a specified time period. Moreover, the model is required to specify the correct time interval applicable for each answer in the list. This type of query aims to assess the model's ability to accurately retrieve and contextualize both temporal information and listwise answers.

3.4. TLQAs generation

We propose an automated solution to generate TLQAs. Our focus is on generating TLQAs from subjects that appear in the first two Wikidata relations mentioned in TempLAMA, namely P54 (member of a sports team) and P39 (position held). The main intuition behind this decision is that their Wikipedia articles may contain **infoboxes** that include timelines relevant to these relations.

3.4.1. Infoboxes

An infobox is a structured box in a Wikipedia article that presents key information about the subject in a standardized format. In Wikipedia's wikitext, the markup language used by Wikipedia, infoboxes follow specific templates designed for different types of subjects. These templates ensure that information is presented consistently across articles. As of today, there are more than 1000 types of infoboxes, which can be found on Wikipedia[1]. Examples of how infoboxes look can be seen in... . These structures often contain timeline-related elements, which can be collected to form timeline-based questions.

3.4.2. Infobox collection

We first focus on collecting the relevant infobox per TempLAMA query. Multiple steps are involved in the infobox collection stage. Given that TempLAMA entries can share identical queries but yield different answers across various timeframes, we first aggregate these entries by query, and group their answer set, which results to 5825 entries. Then, for each aggregated query, the subject of each query is extracted by using the Wikidata relation as a semantic marker. For example, for relation P54, we know that each query will be the form of `subject plays for _X_`, meaning that we can split each query of this type based on the words `plays for`, and select the first position as our subject.

For each subject, the correct Wikipedia page is required to collect the corresponding infobox. Using only the subject's name may be insufficient, as multiple named entities may share the same name. Moreover, as seen previously, TempLAMA subjects may inherently have ambiguous names. Therefore, to resolve each subject to its correct Wikipedia page, an unique identifier is needed. The Wikidata ID serves this purpose, providing a one-to-one relationship between entities and Wikipedia articles.

To accurately find the Wikidata ID for each subject, we utilize the subject's name, the relation type specified in TempLAMA, and the aggregated answer array. TempLAMA provides the Wikidata IDs for each answer, which removes ambiguity in this aspect. We first search for entities with the subject's name on Wikidata, limiting the search to the top 50 results to manage the scope. Next, we filter these results to retain only those entities that have the specified relation type (e.g., P54 for sports teams) present in TempLAMA. Finally, we verify that the object IDs in these relations match the answer IDs provided in TempLAMA. If a match is found, the corresponding Wikidata ID is assigned to the subject.

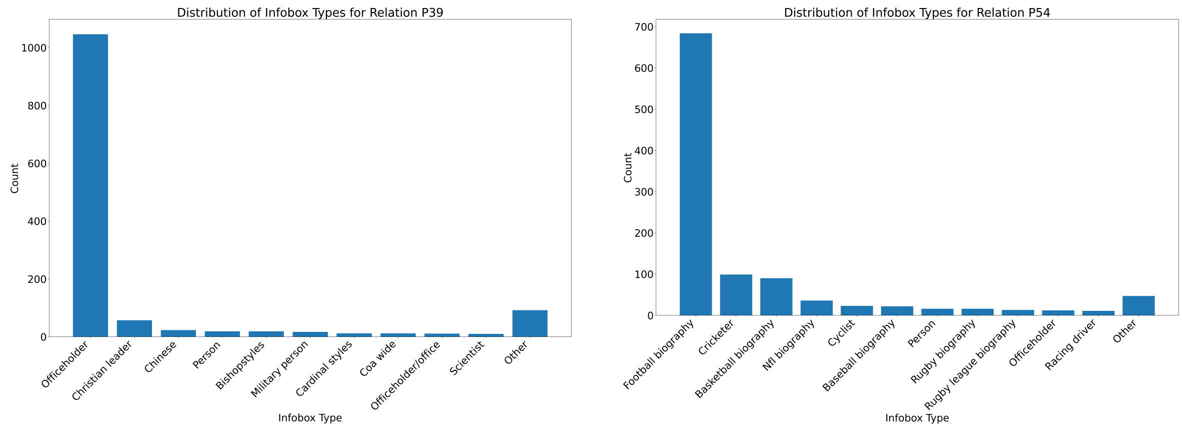
After assigning Wikidata IDs, we perform a verification step to ensure accuracy. For this, we implemented a verification function that checks the assigned Wikidata IDs against the subject names and their aliases. The function fetches the label and aliases for each Wikidata ID and compares them with the normalized subject name. If a match is not found among the names, the entry is removed from the aggregated data set. This process resulted in reducing the initial 5825 entries to 5711 verified entries.

3.4.3. Distribution over Wikidata relatios in TempLAMA

After obtaining the Wikidata IDs, we fetch the corresponding infobox for each subject in the aggregated queries. For each of the nine Wikidata relations mentioned in TempLAMA, we analyze and plot the distribution of infobox types. Infobox types that appear fewer than 10 times are grouped under 'Other' to maintain clarity in the visual representation. The plots for each relation are available in Appendix A.

We focus on relations P39, and P54 as those reveal significant concentrations of certain infobox types, whereas other relations from TempLAMA did not display such bias towards particular infoboxes. The infobox distribution for these two relations can be seen in Figure 3.1.

[1] https://en.wikipedia.org/wiki/Wikipedia:List_of_infoboxes



(a) Distribution of Infobox Types for Relation P39

(b) Distribution of Infobox Types for Relation P54

Figure 3.1: Infobox Type Distribution for Relations P39 and P54

As it can be observed, for the relation P39, **infobox officeholder** represent the most common infobox type, whereas for P54, the most frequent one is **footbal biography**. We can assume that this type of infoboxes exhibit a high amount of temporal diversity, given their inclusion in TempLAMA, which selects relations with high temporal variability. Therefore, these infobox types will be the primary focus in our TLQA benchmark creation. Additionally, to increase diversity, we include the **cricketer** infobox as well. Examples of how these infoboxes look on Wikipedia, and their templates in Wikitext, can be seen in Appendix B.

3.4.4. TLQA from infoboxes

As mentioned previously, each type of infobox follows a specific template designed in wikitext. These templates enforce a structured presentation of information that is consistent across Wikipedia pages. Therefore, we design a parsing function to extract timeline-based questions and answers from these infoboxes. For football players and cricketers, infoboxes include the teams where they played and the corresponding years. Similarly, for officeholders, all political roles are listed with their years. We use the template descriptions for each infobox type to extract, per each subject, their corresponding timeline. The fields used for each category can be seen in Table 3.2.

Infobox Type	**Temporal Markers**	**Answer Fields**
Football Biography	youthyears years nationalyears	youthclubs clubs nationalteam
Officeholder	termstart termend subterm	office suboffice jr/sr statesenate stateassembly order
Cricketer	year internationalspan	club country

Table 3.2: Fields used for extracting temporal information from Infoboxes

To form TLQA queries, we construct a question per each subjects that explicitly requests a list of entities or facts associated with the subject that are known to this day. The mapping of how we design the questions, given an infobox type can be seen in 3.3.

Wikidata Relation ID	Infobox Type	Query
P54	Football biography Cricketer	`List all teams` *subject* `played for to this day.`
P39	Officeholder	`List all political positions` *subject* `held to this day.`

Table 3.3: Mapping from subject to query based on Wikidata relation type and infobox type

To create our answer set, given a generated query, we parse the infobox corresponding to each subject and query to extract the relevant information. We utilize the temporal markers mentioned in Table 3.2 to compile comprehensive answers. For sports-related queries, we extract the teams and the corresponding years the subject played for each team. For political-related queries, we extract the political positions held and the associated time periods. If the end year for a position or team association is not specified, we interpret this as an indication that the subject currently holds the position or remains with the team, aligning with the convention used on Wikipedia. The extracted data is then structured into a dictionary, where each key represents a unique entity (team or political position) and the value is a list of time intervals during which the subject was associated with that entity. This approach allows us to account for subjects who may have multiple non-consecutive associations with the same entity, such as athletes returning to previous teams or politicians re-elected to positions after a hiatus.

Category	Split	Nr. of entries	Mean Nr. of Answers	Median Nr. of Answers
Political	Training	630	5.473	5
	Test	251	5.438	5
	Whole Dataset	881	5.463	5
Sports	Training	528	11.775	11
	Test	246	11.915	11
	Whole Dataset	774	11.820	11

Table 3.4: Mean and median number of answers for political and sports queries, reported separately for training, test, and the whole dataset splits.

The process of generating questions and answers is applied to the extracted TempLAMA subjects marked with the specified infobox types. This results in a collection of **1655** TLQAs, with an average number of answers of **8.641**. We perform a stratified train-test split based on the question's topic (either political or sports-related). For each split and question type, we report the number of entries, the average number of answers, and the median number of answers. The results, shown in Table 3.4indicate that sports-related questions tend to have more answers on average compared to political questions, reflecting greater variability in sports team memberships. In addition, the consistent median values across splits suggest that the distribution of answers remains stable between the training and test sets.

3.5. Corpus Creation

In addition to the benchmark, we curate a corpus for use in conjunction with the TLQA dataset for open-domain QA task. The purpose of this corpus is to provide evidence to answer the timeline-based queries in the benchmark.

To construct the corpora, we parse Wikipedia and extract the title, infobox, and summary from each document. Given that the TLQA benchmark is created based on infoboxes, the infoboxes represent the most important evidence which should be used when answering the queries.

We download the latest English Wikipedia dump[2] with a knowledge cutoff at April 2024. Parsing Wikipedia XML represents a significant challenge due to its custom markup language. However, multiple public libraries are available which deal with this task [3]. We decide to use dumpster-dip [46], a NodeJS library that handles parsing the Wikipedia dump and extracts article attributes such as infoboxes into more digestible formats such as JSON.

The collection contains around 6.7 million articles, from which we filter out the articles without an infobox, resulting in approximately **4.5 million** articles. For each article, we save the title, infobox, and summary. The infoboxes are written as JSON objects. For the summary, we extract the 'lead'[4] section of each article, which provides an overview of the most important points of the article, allowing readers to understand the basics of the topic without having to read the entire article.

[2] https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2
[3] https://www.mediawiki.org/wiki/Alternative_parsers
[4] https://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style/Lead_section

4 Experimental Setup

This chapter details the experimental setup and methodology employed to evaluate the performance of language models on the TLQA benchmark. We describe the models used, their configurations, the inference procedures, and the evaluation metrics. All experiments are conducted on a remote server running Arch Linux, equipped with 2 NVIDIA GeForce RTX 3090 GPUs with 24 GB VRAM each, 256 GB RAM and a 16-core, 32-thread 2nd Gen AMD EPYC™ 7302P CPU.

4.1. Models Used

The experiments in this work were run on two popular open-source generative language models, namely **Mistral 0.2** [47] and **Meta Llama 3** [48].

Mistral 0.2 is a 7-billion-parameter language model based on the transformer architecture [49], developed by Mistral AI. It features a context window of 32,000 tokens and a vocabulary size of 32,000 tokens. The model uses Grouped Query Attention (GQA) [50] to improve efficiency and performance during inference. Released on March 23, 2024, Mistral 0.2 has demonstrated superior performance compared to other open-source models with a similar or higher number of parameters. It is trained on publicly available data, although the specific datasets used have not been disclosed.

Meta Llama 3 is an 8-billion-parameter language model that likewise uses the transformer architecture. It has a context window of 8,000 tokens and a significantly larger vocabulary size of 128,000 tokens. Similar to Mistral 0.2, Meta Llama 3 adopts GQA to improve inference efficiency. Although the specific training datasets are not openly accessible, the authors indicate that the model was pre-trained on over 15 trillion tokens from publicly available sources. Additionally, previous Llama models were utilized to identify and generate high-quality training data. The model was publicly released on April 18, 2024, and was shown to achieve state-of-the-art results on different benchmarks, outperforming models with a similar parameter count.

In the experiments, we use the instruct versions of both Mistral 0.2 and Meta Llama 3 models. Instruct models are fine-tuned versions of generative language models specifically designed to follow instructions given user prompts. They are trained on datasets that include various instructions and their corresponding responses. The authors of Instruct Mistral 0.2 mention that they use the instructions datasets publicly available on Huggingface for fine-tuning. Meta Llama 3 instruction tuning relies on a combination of supervised fine-tuning (SFT), rejection sampling, proximal policy optimization (PPO), and direct preference optimization (DPO), in addition to using human-annotated feedback.

For efficiency in runtime during our experiments, we use low-quantized versions of the models. Quantization reduces the model size while aiming to maintain minimal performance difference compared to the standard versions. We download the GPT-Generated Unified Format (GGUF) of the models from Huggingface. Specifically, we choose Q5[1] for Mistral 0.2 and Q6[2] for Meta Llama 3 based on the quan-

[1] `https://huggingface.co/TheBloke/Mistral-7B-Instruct-v0.2-GGUF`
[2] `https://huggingface.co/bartowski/Meta-Llama-3-8B-Instruct-GGUF`

tization authors' recommendations. These quantized versions allow us to conduct experiments in less time while maintaining the integrity and performance of the original models.

During our experiments, we keep the temperature parameter at 0.3. Temperature in the context of language models controls the randomness of predictions by scaling the logits before applying the softmax function. A lower temperature value, such as 0.3, makes the model's predictions more deterministic by reducing the probability of less likely outcomes and focusing more on the higher probability predictions. This approach ensures that the responses generated by the model are consistent, allowing us to evaluate the results more easily.

4.2. Evaluation Methodology

In this section, we highlight the steps taken to evaluate the performance of the language models. The objective is to quantify both listwise and temporal performance. To achieve this, we first perform an **entity matching** step, which aligns the entities from the model-generated answers with those in our benchmark.

4.2.1. Entity Matching

Given the nature of our TLQAs, the answers will always be in the form of entity names and years. The issue behind this is that language models may generate valid entity names that differ from those in the ground truth due to variations in their training data. These differences can include variations in naming conventions, abbreviations, or alternative spellings. Therefore, to accurately assess the model's performance, we design a pipeline which match entity names that are written differently but denote the same concept.

The pipeline contains three stages, namely a *subsequence* matching, a *fuzzy* matching and an *embedding* matching. We run this pipeline sequentially, gradually removing pairs of generated entries and ground truth entries that match. The criteria for matching become progressively less stringent at each stage. Examples of matched entities from different stages of our pipeline can be seen in Table 4.1.

Matching Type	**Expected Answer**	**Generated Answer**
Subsequence	`mayor of new Taipei city`	`mayor of new Taipei`
Fuzzy	`leader of the opposition in the legislative assembly of manitoba`	`leader of the opposition in manitoba`
Embedding	`leader of syriza`	`president of syriza`

Table 4.1: Examples of matched entities for different types of matching. Highlighted in red are the differences between the expected answer from our benchmark and the generated answer from the model. Examples extracted from prompting Meta Llama 3 on our benchmark.

During subsequence matching, we first check if the generated entity name exactly matches any remaining ground truth entries. If no exact match is found, we then check if the generated entity name is a subsequence of any remaining ground truth entries, or vice versa. This stage primarily catches small differences, such as variations in football club names such as "FC Barcelona" versus "Barcelona," or situations similar to the ones shown in Table 4.1.

For fuzzy matching, we use the Token Set Ratio algorithm provided by the FuzzyWuzzy library. This algorithm tokenizes both the generated and ground truth entity names into sets of unique tokens (words), removes the common tokens from each set, and then computes a similarity score based on the Levenshtein distance. The Levenshtein distance is defined as the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one string into another. We match entries only if their Token Set Ratio score is greater than 90. This step catches mistakes such as slight variations in naming conventions.

During the embedding matching stage, we use an embedding model to generate vector representations of the entity names. We then compute the cosine similarity between each pair of vectors from the generated and expected answer sets, removing pairs with the highest similarities first. We match entries only if the cosine similarity is greater than 0.85. We use the all-MiniLM-L6-v2 model, a small embedding model trained to produce semantically meaningful embeddings for sentences and phrases. In this part of the pipeline, entity names that have different words, but the same meaning, are matched, as seen in Table 4.1.

4.2.2. Listwise Metrics

To evaluate the results of the language model in terms of listwise performance, we apply the standard definitions of $Precision$, $Recall$, and $F_1 Score$. Given a pair of generated and expected answer sets, we count the true positives (TP) as the number of matched entities. We define False negatives (FN) as the entities in the expected set that are not matched, and false positives (FP) as the entities in the generated set that are not matched. Based on this, we define the three metrics as follows:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F_1 Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

In our context, $Precision$ denotes the proportion of correctly identified entities among all entities identified by the model, quantifying how correct the model's answers are. $Recall$, on the other hand, denotes the proportion of correctly identified entities among all entities that should have been identified, reporting how complete the model's answers are. The $F_1 Score$ balances these two metrics, providing a single measure that reflects both the quality and quantity of the generated answer.

We calculate these metrics for each set of generated and expected answers, and then average them across all examples to obtain an overall evaluation of the model's performance.

4.2.3. Temporal Metrics

To evaluate temporal performance for our given setup, we define two different metrics: $Temporal\ Overlap$ and $Temporal\ Jaccard$.

The Temporal Overlap score measures the extent to which the generated date ranges align with the expected date ranges. It is computed as the ratio of the intersection of the generated and expected date ranges to the total length of the expected date ranges. Formally, given the set of years covered by the expected date ranges Y_{expected} and the set of years covered by the generated date ranges $Y_{\text{generated}}$, we define Temporal Overlap as:

$$Temporal\ Overlap = \frac{|Y_{\text{expected}} \cap Y_{\text{generated}}|}{|Y_{\text{expected}}|}$$

This score ranges from 0 to 1, where 1 indicates a perfect match between the generated and expected date ranges, and 0 indicates no overlap.

The Temporal Jaccard measures the similarity between the generated and expected date ranges, considering both the intersection and union of the date ranges. Unlike Temporal Overlap, Temporal Jaccard penalizes the model for hallucinating years outside the expected range. We compute the metric as the ratio of the intersection of the generated and expected date ranges to the union of both date ranges. Formally, we define it as:

$$Temporal\ Jaccard = \frac{|Y_{\text{expected}} \cap Y_{\text{generated}}|}{|Y_{\text{expected}} \cup Y_{\text{generated}}|}$$

This score also ranges from 0 to 1, where 1 indicates a perfect match and 0 indicates no similarity.

To compute these scores, we first parse and normalize the date ranges from both the generated and expected answers. For each matched entity, we represent the date ranges as set of years, which we further use to calculate the Temporal Overlap Score and Temporal Jaccard Similarity. We then average these scores across all matched entities to obtain the overall temporal performance of the model.

4.3. Prompting Techniques

To assess the model's performance on the TLQA benchmark, we employ different **prompting** techniques to guide the model's answer generation. Prompting, in the context of generative language models, refers to the action of providing a structured input that orients the model toward a specific task or style of response. A prompt may include different components. Given our QA setup, we consider three different types of messages that may be included in the prompt:

- **System Message:** This message sets the context or behavior for the model. It provides overarching instructions on how the model should behave during the interaction. For our scope, we instruct the model to answer TLQAs, and structure the output similar to our benchmark, to ease parsing.
- **User Message:** These are the questions or commands provided by the user that the model needs to respond to. In our case, user messages will represent the questions from our benchmark.
- **Assistant Message:** These represent the messages generated by the model. However, if appended to the prompt, they serve as examples responses that demonstrate how the model should reply.

The way prompts are constructed may highly affect the models' performance. Therefore, we study different approaches to create meaningful prompts.

4.3.1. Manual few-shot

Brown et al. [20] showcase how large language models can achieve impressive performance on different Natural Language Processing tasks, by simply providing demonstrations of how the task should be performed. In the case of QA tasks, this involves presenting the model with examples of questions and their corresponding answers. In our approach, we select question-answer pairs from our training set to serve as few-shot examples and evaluate the model on the test set. To maintain consistency, we provide the same examples for all questions in the test set.

4.3.2. KNN few-shot

Instead of manually selecting the examples, Liu et al. [51] propose an automatic method to sample the most relevant examples for the query using K-Nearest Neighbors (KNN). The idea is to embed the query and perform the search to find similar examples. In our case, we embed all questions in the training set, and for each question in the test set, we find the top k questions in the training set that are most similar. These similar questions, along with their corresponding answers, are then provided as few-shot examples. For embedding, we use the all-mini-lm-v2 model, as it is lightweight and has good performance.

4.3.3. Manual Chain of Thought

Wei et al. [52] introduce the concept of Chain of Thought (CoT) prompting, a technique that involves providing the model with a sequence of intermediate reasoning steps that lead to the final answer, in conjunction with the few-shot examples. This method helps the model to break down complex problems into more manageable sub-problems, thereby enhancing its ability to perform tasks that require logical reasoning and multi-step problem-solving. We implement this approach by manually selecting and writing rationales for two different queries of different topics from the training set. These examples, along with the questions from the test set, are provided to the model as a prompt.

4.3.4. Auto Chain of Thought

Another implementation of chain of thought prompting, Auto Chain of Thought (AutoCoT), was developed by Zhang et al. [53]. AutoCoT automates the generation of intermediate reasoning steps, by using the model's capabilities to create these steps rather than relying on manually written rationales.

This technique uses a few-shot prompting approach, where the model is given examples of reasoning chains and then asked to generate similar chains for new questions. In our implementation, we combine AutoCoT with the KNN seen previously. For each query from the test set, we select the top k closest questions from the training set. For each selected question, we ask the model to first generate rationales by thinking 'step-by-step'. We then append the training questions, along with their generated rationale chain and the answers from the training set, to the prompt. This is provided to the model in a few-shot manner. Similarly, to answer the question from the test set, the model is first tasked with generating a rationale and then extracting its answer from the chain.

4.4. Retrieval Augmented Generation

All settings mentioned in the previous section operate in a closed-book manner, where the language model generates answers based solely on its pre-existing knowledge without access to external information. In this section, we further consider an open-book setup, in which we append additional context to our prompts that the model can use to answer the query. To achieve this, we implement a simple Retrieval Augmented Generation (RAG) approach [22]. The core idea behind RAG is to retrieve additional documents from an external source as supporting evidence for the language model to use in generating answers.

In our implementation, we treat each question in the test set as a retrieval query and perform a search on the proposed corpus derived from Wikipedia, outlined in Section 3.5. Given our corpus, we consider two scenarios: one where the documents contain title-infobox-summaries and another where the documents contain only title-summaries. This differentiation is made to study the effect of the infobox on retrieval performance, as the infobox may include highly structured information that could impact retrieval performance.

From the retrieved documents, we select only the infobox to be included as context to the query. As previously mentioned, the answers in our dataset are purely extracted from infoboxes, therefore they should provide the required information to give the correct answer to the query. In the title-infobox-summary setup, we extract the infobox from each document and append it to the prompt as context. In the title-summary setup, we create a mapping between all Wikipedia titles and their corresponding infoboxes. When fetching documents, we use this mapping to find and append the relevant infobox to the query as additional context. We transform the infobox content from a JSON format to a more simplistic format by unfolding it into a single $key : value$ pair per line. This transformation is done to avoid the verbosity of the JSON structure and make the information more digestible for the model.

Two main paradigms are typically considered in the design and implementation of information retrieval systems based on their document representation techniques: sparse retrieval and dense retrieval. We implement both approaches and discuss how they work in the following sections.

4.4.1. Sparse Retrieval

Sparse retrieval refers to traditional information retrieval methods where documents and queries are represented by sparse vectors. Each dimension of the vector corresponds to a unique term in the vocabulary of the entire corpus. Since any given document or query usually contains only a small subset of the terms in the vocabulary, most dimensions will have a value of zero, resulting in a sparse vector. This sparsity is advantageous for storage and computational efficiency, as operations can ignore the zero values. These vectors are typically derived from term frequency-inverse document frequency (TF-IDF) representations or other bag-of-words models.

TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus). The term frequency (TF) is the number of times a term appears in a document, while the inverse document frequency (IDF) is a measure of how much information the word provides, based on how commonly it appears across the entire corpus. The TF-IDF score for a term in a document is calculated as follows:

$$TF - IDF(t, d) = TF(t, d) \times IDF(t)$$

where:

- $TF(t, d)$ is the term frequency of term t in document d, calculated as the number of times t appears in d divided by the total number of terms in d.
- $IDF(t)$ is the inverse document frequency of term t, calculated as $\log(\frac{N}{1+df(t)})$, where N is the total number of documents in the corpus and $df(t)$ is the number of documents containing the term t.

One of the most well-known sparse retrieval algorithms, which is still considered a strong baseline to this day, is BM25. BM25, or Best Matching 25, is a ranking function used to estimate the relevance of documents to a given search query. The algorithm is based on the probabilistic retrieval framework and refines TF-IDF by incorporating term saturation and document length normalization. The BM25 score for a document D and query Q is calculated as follows:

$$BM25(D, Q) = \sum_{t \in Q} IDF(t) \cdot \frac{f(t, D) \cdot (k_1 + 1)}{tf(t, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})}$$

where:

- $tf(t, D)$ is the term frequency of term t in document D,
- $|D|$ is the length of document D,
- $avgdl$ is the average document length in the corpus,
- k_1 and b are free parameters, with k_1 typically set to 1.2 and b set to 0.75,
- $IDF(t)$ is the inverse document frequency of term t.

In our implementation, we use Elasticsearch[3] to index and retrieve documents based on the BM25 algorithm. First, we load the Wikipedia corpus, parsing each to extract the title, infobox, and summary. These parsed documents are then indexed in Elasticsearch, where each document is stored with its associated terms and their frequencies. We create two indices: one for documents containing title-infobox-summary and another for documents containing title-summary. For each test query, Elasticsearch uses the BM25 algorithm to rank and retrieve the most relevant documents based on their indexed terms. From the retrieved documents, the relevant infobox content is extracted and formatted to be appended as context to the query.

4.4.2. Dense Retrieval

Dense retrieval is a modern information retrieval method that involves representing both documents and queries as dense vectors in a continuous vector space. Both the documents and the queries are encoded in the same vector space using an embedding model. An embedding model is a type of neural network trained to transform text into numerical vector representations, where semantically similar texts are mapped to nearby points in the vector space. The similarity between the query vector and each document vector can be computed using metrics such as cosine similarity, which measures the cosine of the angle between two vectors. Documents are ranked based on their similarity scores, with higher scores indicating greater relevance to the query.

In our implementation, we use ChromaDB[4] to create and store embeddings based on our Wikipedia corpus. Similar to before, we create two different vector stores given our Wikipedia corpus: one where documents include title-infobox-summary, and another where documents include only title-summary. We experiment with two different embedding models: all-MiniLM-L6-v2 and multi-qa-mpnet. Chroma recommends using all-MiniLM for its balance of performance and efficiency. Additionally, we use the multi-qa-mpnet model as it is reported to have been pre-trained on question-answer pairs, making it highly suitable for our TLQA benchmark. For similarity computations, Chroma uses cosine similarity to measure the relevance between vectors. At prompting, we embed the question and use it to query the Chroma vectorstore. From the documents returned, we extract their infoboxes, appending it to the prompt as additional context.

[3] `https://www.elastic.co/`
[4] `https://www.trychroma.com/`

4.4.3. Retrieval Metrics

To evaluate the performance of our retrieval systems, we consider multiple commonly used metrics in information retrieval.

Mean Reciprocal Rank (MRR) evaluates the retrieval system based on the rank position of the first relevant document. Given our setup, for each question, there exists only one infobox that was used to generate the answer. Therefore, we can assume that there is only one relevant document per query. Thus, we consider this metric to be most relevant for our user case. The metric is calculated as the average of the reciprocal ranks of the first relevant document across all queries:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

where $|Q|$ is the total number of queries and $rank_i$ is the rank position of the first relevant document for the i-th query.

Normalized Discounted Cumulative Gain (NDCG) measures the usefulness of a document based on its position in the result list. It accounts for the graded relevance of documents and the rank of the documents in the retrieved list. The higher the position of a relevant document, the greater the contribution to the overall gain. The NDCG is calculated as follows:

$$NDCG@k = \frac{DCG@k}{IDCG@k}$$

where $DCG@k$ (Discounted Cumulative Gain) is:

$$DCG@k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

and $IDCG@k$ (Ideal Discounted Cumulative Gain) is the DCG score of the ideal ranking.

Precision measures the fraction of relevant documents retrieved out of the total documents retrieved. It is a measure of the accuracy of the retrieval system and is defined as:

$$Precision = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{retrieved\ documents\}|}$$

where the numerator is the number of relevant documents retrieved, and the denominator is the total number of documents retrieved.

Recall measures the fraction of relevant documents retrieved out of the total relevant documents available. It is a measure of the completeness of the retrieval system and is defined as:

$$Recall = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{relevant\ documents\}|}$$

4.4.4. Golden Evidence and Distractors

Another setup we experiment with assumes perfect retrieval, always returning the correct infobox per query. We refer to the correct infobox in this context as the golden evidence. For each question in our test set, we collect the golden infobox and append it to the prompt as context. This setup allows us to estimate the upper bound of how well Retrieval Augmented Generation (RAG) can improve the performance of models on our TLQA benchmark.

Additionally, we investigate the impact of distractors on performance. We define distractors as any infoboxes other than the golden one. The goal is to study how much these distractors affect performance. To do this, we sample three different types of distractors:

- **Sparse retrieval distractors**: These are selected using the BM25 algorithm.
- **Dense retrieval distractors**: These are selected using ChromaDB.
- **Random distractors**: These are randomly selected from our corpus.

We append a varying number of distractors, along with the golden infobox, to the query during prompting. This allows us to study the model's performance in the presence of noise, thereby understanding how much distractors impact the effectiveness of the model.

5

Experimental Results

In this chapter, we present and discuss the main results obtained from executing the experimental setup described previously. Three different setups are considered during the inference of models, namely classical prompting techniques, retrieval augmented generation, and golden evidence with the effect of different distractors.

5.1. Standard Prompting

To answer **Research Question 1**, we study the performance of two open source large language models, namely Mistral 0.2 and Meta LLama 3, against the TLQA benchmark. This is done in a closed-book setup, meaning that there is no evidence provided. The model should answer the queries from its own knowledge and respect the guidelines mentioned in the system message. The outcome from the different prompting techniques presented previously can be seen in Table 5.1.

Model	Precision	Recall	F1	Time Overlap	Time Jaccard
Mistral 0.2 7b					
Manual 3 shot	0.558	0.257	0.330	0.384	0.317
KNN 3 shot	0.532	0.310	**0.366**	0.435	0.360
Manual COT	0.529	0.276	0.338	0.412	0.330
Auto COT	0.523	0.311	0.365	0.346	0.282
Meta Llama 3 8b					
Manual 3 shot	0.582	0.345	0.414	0.491	0.408
KNN 3 shot	0.548	0.399	**0.442**	0.519	0.428
Manual COT	0.595	0.326	0.399	0.515	0.423
Auto COT	0.276	0.199	0.218	0.314	0.260

Table 5.1: Performance of Mistral 0.2 and Meta Llama 3 models on the benchmark using different prompting techniques. The first three metrics evaluate **listwise completeness** (how complete the answer list compared to the ground truth), whereas the last two assess the **temporal correctness** of the answers provided

The results indicate certain trends that are consistent among the two language models.

Low Recall: All settings exhibit substantially lower listwise recall scores compared to precision. This indicates that models often provide answers that are correct with respect to the ground truth but incomplete. Several factors inherent to these language models can explain this low recall. One reason could be that instruction-tuned models typically become more 'conservative' with their answers, focusing on the quality of answers returned. Therefore, they are less inclined to generate answers with lower token probability distributions, which results in fewer overall responses. This trend is showcased in Figure 5.1, which shows that the average number of expected answers is substantially larger than the number of answers generated by the language models, using different prompting techniques. This behavior stems from the fact that instruct models are fine-tuned using reinforcement learning from human feedback (RLHF) [54], a process which emphasizes high precision by aligning the model outputs closely with human-provided instructions and feedback, frequently at the expense of recall. As a result, these models prioritize generating highly accurate responses, which leads to a more selective approach, providing fewer but more reliable answers. These findings align with the ones by Le et al. [55], who define precision as the proportion of generated outputs that fall within the support of the reference distribution and recall as the proportion of the reference distribution covered by the generated outputs. While their definitions apply to the quality and diversity of text generation, the observed behavior is similar: models fine-tuned to follow human instructions tend to lose completeness in their answers.

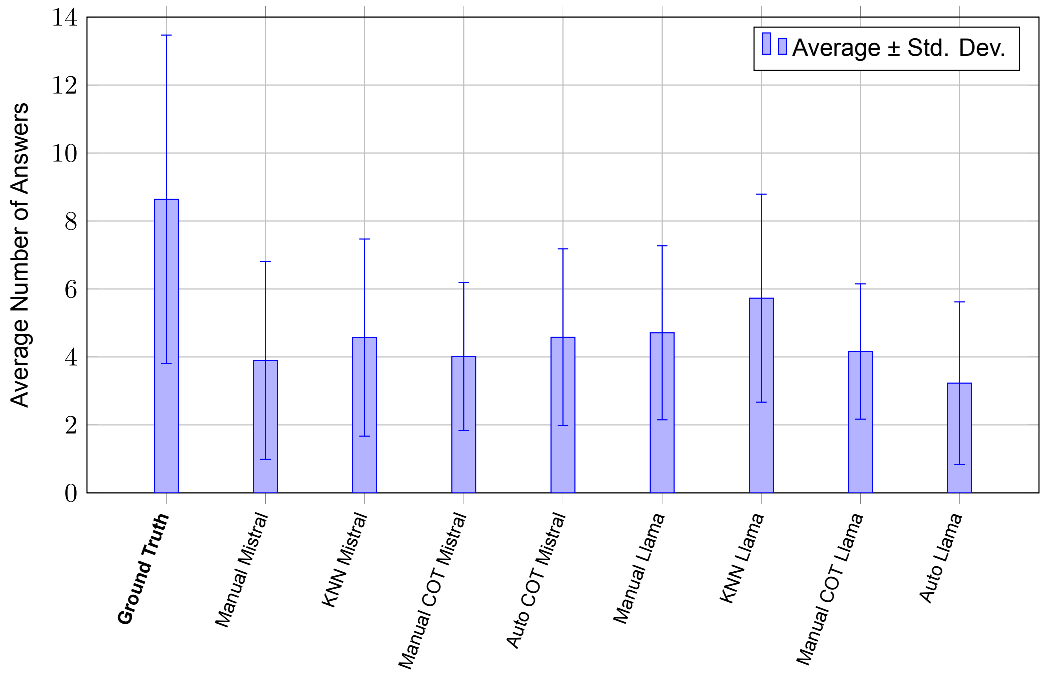


Figure 5.1: Average number of answers with standard deviations for the different prompting strategies for Mistral and Meta LLama

KNN performance: For both models, Table 5.1 indicates a slight increase in performance in listwise and temporal scores, when using inference with the KNN few-shot technique compared to manually selecting examples from the training set. This implies that few-shot examples that are more relevant to the query topic improve the models' ability to answer it. In our case, we expect the KNN algorithm to always select samples that have a similar area of interest (either sports or politics in our benchmark) with the query. This could be because selecting contextually similar examples activates the relevant parts of the model's pre-trained knowledge. When examples are closely related to the query, the model's attention mechanisms are better aligned with the specific patterns of the domain, such as common terminology and contextual associations.

COT performance: The performance of the two Chain-of-Thought (COT) techniques varies between the two models. For Mistral, both manual and auto COT show slight improvements in listwise F1 scores over standard few-shot prompting. However, neither technique outperforms KNN prompting, and auto COT performs worse in temporal metrics compared to manual few-shot. For Meta Llama 3, both COT techniques underperform compared to manual and KNN few-shot prompting, with auto COT exhibiting a significant drop in performance across both listwise and temporal metrics. One explanation for the suboptimal performance of COT in this setup could be the limitations imposed by the context window size. Mistral has a 32k token window, whereas Meta Llama has a 8k token window [56, 48]. The creation of rationales and the subsequent extraction of the final answer may consume too much context, particularly because TLQAs require a substantial amount of tokens to generate correct answers. The context limitation could explain why AutoCOT performs significantly worse than Manual COT in Meta Llama, whereas in Mistral, it performs better in listwise metrics.

Temporal performance: Both models generally perform suboptimally in terms of temporal scores on TLQA tasks. It can be observed that the Time Overlap scores are consistently higher than the Time Jaccard scores, indicating that while the models are not very capable of identifying significant overlapping time periods, they struggle even more with accurately bounding the timelines. This discrepancy likely arises because the models can detect relevant time frames but have difficulty precisely delineating the start and end years, resulting in either overshooting or undershooting the correct periods.

Mistral VS Meta Llama3: Both model's cut off knowledge is estimated to be around 2023, with Meta Llama s being in March, while Mistrals not being publicly available. This suggests that both models should, in theory, be able to answer most queries from TLQA benchmark with information up to date, with similar performance. Nevertheless, in most settings, Meta Llama 3 performs considerably better, both timewise and listwise, compared to Mistral. This difference in performance can be attributed to several factors, including training data quality and model architecture. While both models claim to use publicly available data during training, neither disclosed their training datasets. Therefore, we hypothesize that Meta Llama 3 may benefit from a better-curated and more diverse training set, potentially providing a broader knowledge base. Architecturally, Mistral has a larger context window size of 32k tokens compared to Meta Llama 3's 8k tokens [56, 48]. However, Meta Llama 3 has a significantly larger vocabulary size of 128k tokens compared to Mistral's 32k tokens [47, 48]. The larger vocabulary size enables the model to recognize and generate more words and terminlogy, which may help with providing more complete answers when dealing with TLQAs. This could be another reason why Llama 3 performs substantially better than Mistral, and why, on average, it generates more answers, as seen in Figure 5.1.

5.1.1. Impact of Varying Top K Elements on Few-Shot Prompting

We further study the effect of the number of samples provided in the context of manual few-shot and KNN few-shot. The performance can be seen in Figure 5.2.

For the listwise F1 score, the performance shows a rapid convergence, indicating that increasing the number of examples beyond a certain point yields diminishing returns. Specifically, for both KNN and manual few-shot techniques, there is minimal performance improvement as the number of examples (K) increases. This suggests that the models quickly learn the necessary patterns with just a few examples, and adding more examples does not significantly enhance the F1 score.

However, for the temporal metrics (Temporal Overlap and Temporal Jaccard), there is a notable difference in the trends between KNN and manual prompting. In the case of KNN few-shot, the temporal performance improves consistently as K increases. This implies that providing more contextually similar examples helps the model better align temporally relevant information, leading to improved temporal accuracy. The KNN approach benefits from having more examples that are contextually similar, which aids in better temporal alignment of events.

Conversely, in manual few-shot prompting, the temporal metrics do not show the same level of improvement and even slightly decrease in some cases. This could be due to the manually selected examples not being as contextually aligned with the query, leading to potential noise and distractions that hinder temporal performance. In manual few-shot, the quality and relevance of the examples are critical, and increasing K without maintaining contextual similarity might introduce irrelevant information, thereby impacting the model's ability to temporally align the answers accurately.

Take-Away 1 : *Both Mistral and Meta Llama 3 perform suboptimally on the TLQA benchmark, in terms of listwise and temporal performance. They exhibit substantially lower recall, indicating a struggle with providing complete answers. Temporally, the models score poorly on both Time Overlap and Time Jaccard metrics, suggesting that they struggle with providing the correct time intervals and with bounding the correct timelines as well.*

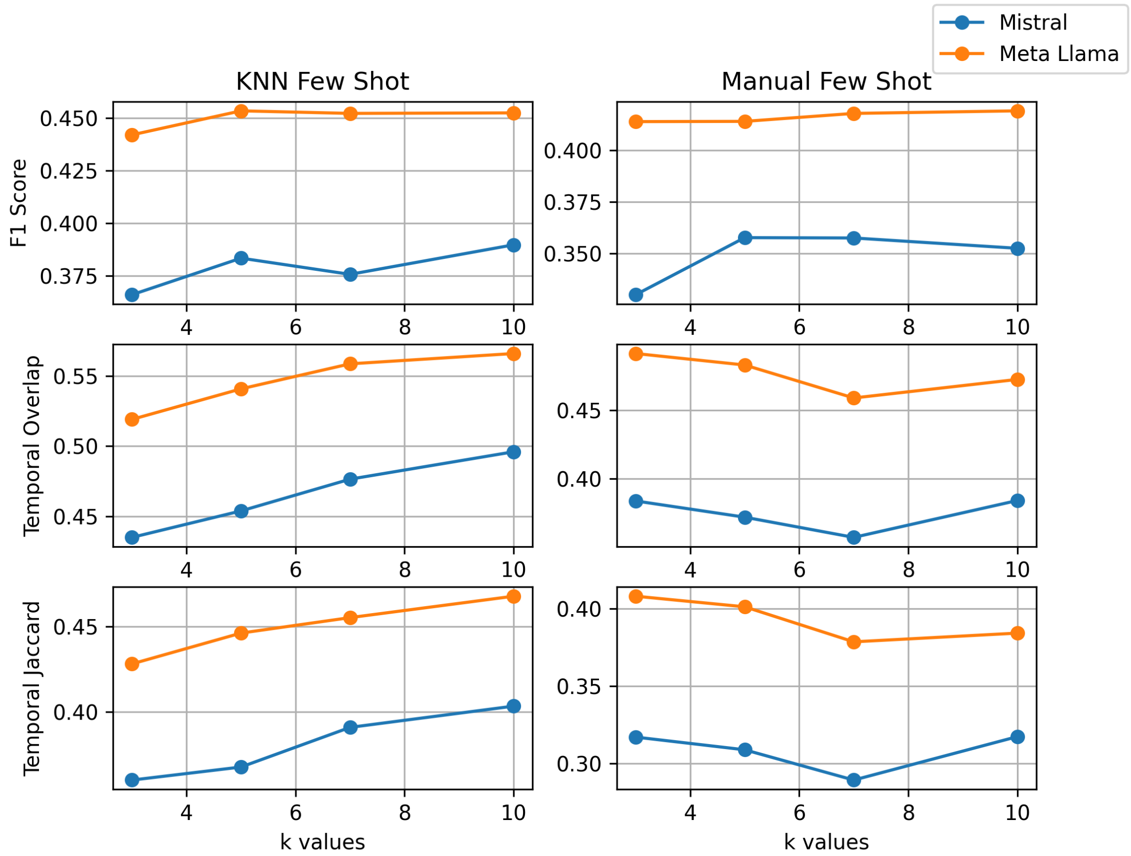


Figure 5.2: Performance of KNN and Manual Few shot over different k shots.

5.2. Retrieval Augmented Generation

To address **Research Question 2**, we study the impact of augmenting the query with additional context to determine if this improves the model's performance on TLQA tasks. The context is fetched with the different retrieval setups previously described and then added to the prompt. This experiment is conducted exclusively with Mistral 0.2, as Meta Llama 3's context window is insufficient to accommodate the majority of context passages effectively.

5.2.1. Retrieval Performance

We first evaluate the performance of the different retrieval settings on the two custom Wikipedia corpora. This evaluation is conducted to understand how retrieval performance impacts the overall performance of the RAG model. We hypothesize that setups with better retrieval performance will lead to improved overall performance of the RAG model on TLQA tasks.

The performance of the different retrieval systems in terms of Mean Reciprocal Rank (MRR) is shown in Figure 5.3. Additionally, the plots for other retrieval metrics, such as Precision, Recall, and NDCG, are provided in Appendix C. We focus on MRR because it directly measures the rank of the first relevant item. As described previously, the answer for each timeline question in the benchmark is extracted from one unique infobox. Therefore, we are interested in retrieving the specific document that contains this infobox, making MRR the most representative metric to gauge the performance of the retrievers.

Based on the given plot, several observations can be made:

Dense vs Sparse: Sparse retrieval, represented by BM25, performs substantially better than dense retrieval methods on both the title-summary and title-infobox-summary settings. This can be attributed to several factors. Firstly, Sciavolino et al. [57] prove that dense retrieval systems perform considerably worse than BM25 in simple entity-centric questions. They show that when dense retrievers are tasked

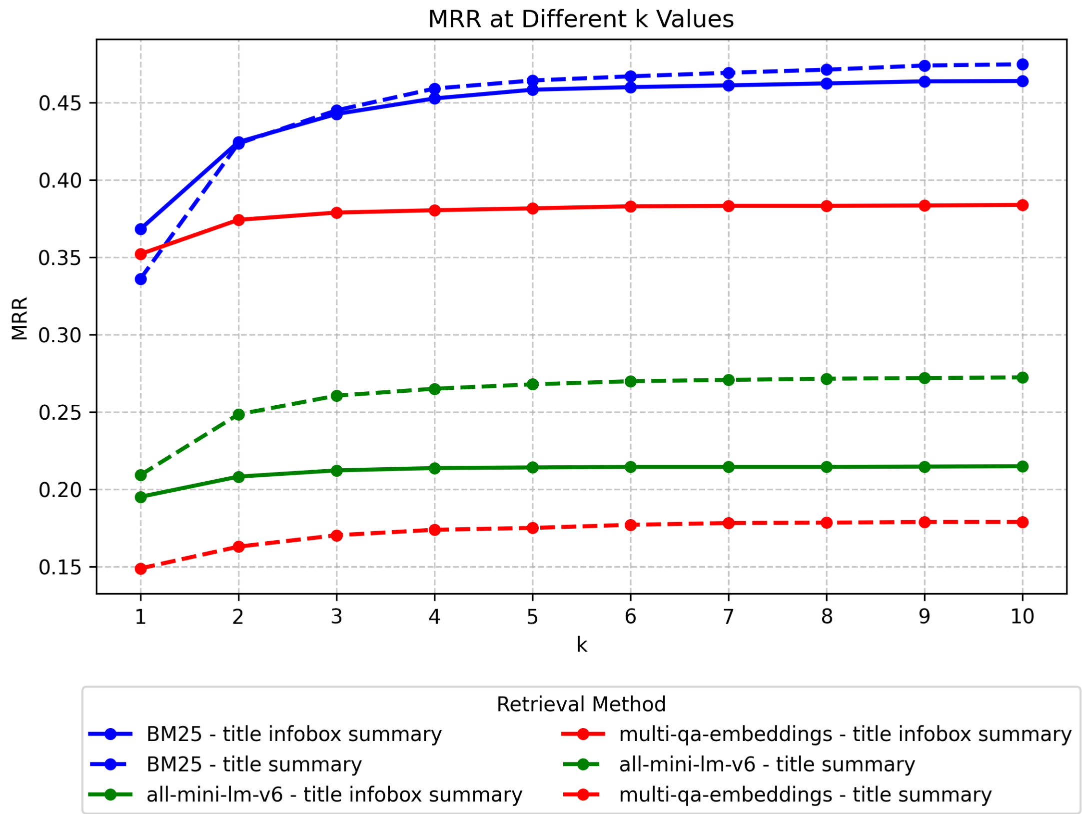


Figure 5.3: Performance of different retrieval settings over the two Wikipedia corpora, title-summary and title-infobox-summary. Dashed line represents title-summary, whereas full line represents title-infobox-summary.

with entity-based questions, they suffer from issues such as popularity bias, where models are biased towards entities seen more frequently, leading to poorer performance on rarer entities. BM25, on the other hand, does not rely on the frequency of entities and instead uses exact term matching, which ensures that specific terms in queries are matched directly to documents, making it more effective for entity-centered data. Given that the TLQA benchmark and the collected corpora are based on named entities, BM25 is likely to perform better on our corpora due to these characteristics.

Another reason why sparse retrieval performs better could be that dense retrievers require extensive context to perform optimally. When documents contain structured or summarized information, such as infoboxes or summaries, the lack of contextual information can hinder the performance of dense retrievers. Dense models rely on understanding relationships within the text, which can be challenging when encoding infoboxes and summaries. In contrast, BM25 does not depend on contextual embeddings and can effectively utilize the available structured data, making it more suitable for infobox and summary retrieval tasks.

Title-Summary vs Title-Infobox-Summary: The retrieval performance of the dense and sparse retrievals differs between the title-summary and title-infobox-summary corpora. For each retriever, we can make the following observations:

- **BM25:** The retrieval performance of BM25 shows minimal difference between the title-summary and title-infobox-summary corpora. At k=1, BM25 scores 0.368 on the title-infobox-summary corpus and 0.336 on the title-summary corpus. As k increases to 10, BM25's performance on the title-summary corpus converges to 0.478, while the title-infobox-summary corpus converges to 0.467. This minimal difference can be explained by how BM25 operates. In both cases, the goal is to find the page that best matches the query terms, with the entity name being the most relevant term that points to the correct Wikipedia page. The entity name is typically present in the summary or title, ensuring a match. Although the infobox may also contain the entity name, it could introduce additional named entities, potentially adding noise. This noise could slightly affect performance, leading to different convergence values.

- **all-mini-lm-v6:** The retrieval performance of all-mini-lm-v6 shows a more noticeable difference between the title-summary and title-infobox-summary corpora. At k=1, the MRR scores are 0.195 for the title-infobox-summary corpus and 0.210 for the title-summary corpus. By k=10, these values converge to 0.215 for title-infobox-summary and 0.275 for title-summary, indicating a widening performance gap. The generally suboptimal performance, particularly under the title-infobox-summary setup, can be attributed to how the embeddings were created. All-mini-lm-v6 was primarily trained to distinguish between semantically similar and dissimilar sentences using a contrastive learning objective. Therefore, it may not be able to encode structured data such as infoboxes properly, adding more noise to the vector representations of the documents, which impacts its performance.
- **multi-qa-mpnet:** Multi-qa-mpnet shows a significant performance disparity between the title-summary and title-infobox-summary corpora. At k=1, the MRR scores are 0.352 for the title infobox summary and 0.149 for the title summary. In both settings, the model's performance converges fast, maintaining a similar difference in performance. By k=10, the MRR scores are 0.385 for title-infobox-summary and 0.180 for title-summary. The reason for this discrepancy in performance on the two corpora could be due to how the model was trained on a large corpus of question-answer pairs, which optimizes it for encoding and retrieving answers. As infoboxes contain the structured information needed to answer queries in the TLQA dataset, the model could be able to better encode them as possible answers to queries. In contrast, title-summaries corpus lacks the full answer of the query, as the infobox is not included, resulting in lower performance.

The order of how the different retrievers rank on other metrics, such as Precision, Recall, and NDCG, is consistent with the observations made from the MRR scores, as it can be seen in Appendix C. BM25 consistently outperforms the dense retrieval models across both corpora. Given these results, we expect BM25 on both corpora to lead to the highest listwise and temporal scores in our RAG setup.

5.2.2. RAG Performance

We perform our RAG setup with Mistral 0.2 using the three different retrievers on the TLQA benchmark. The number of top-ranked documents retrieved that are being appended to the query as additional context, denoted as k, was varied to assess its impact on performance. We use the best prompting method previously found in Section 5.1, namely KNN 3-shot. We report the listwise F1 score, Time Overlap and Time Jaccard for each k. The experiments were conducted on both title-summary and title-infobox-summary corpora. The results can be seen in Table 5.2.

Retrieval Setup	**Corpus**	**F1@k**			**Time overlap@k**			**Time Jaccard@k**		
		k=1	**k=3**	**k=10**	**k=1**	**k=3**	**k=10**	**k=1**	**k=3**	**k=10**
all-mini-lm-v6	Title - Summary	0.367	0.430	0.417	0.372	0.414	0.400	0.323	0.366	0.351
	Title - Infobox - Summary	0.394	0.398	0.368	0.410	0.398	0.340	0.360	0.341	0.293
BM25	Title - Summary	0.502	0.594	0.618	0.503	0.594	0.627	0.458	0.556	0.584
	Title - Infobox - Summary	0.536	0.607	**0.624**	0.558	0.623	**0.651**	0.513	0.581	**0.607**
multi-qa-mpnet	Title - Summary	0.353	0.379	0.376	0.373	0.375	0.354	0.310	0.318	0.300
	Title - Infobox - Summary	0.441	0.426	—	0.436	0.407	—	0.375	0.352	—

Table 5.2: Performance Evaluation of Different Retrieval Setups at k = 1, 3, and 10, including Time overlap and Time Jaccard KNN 3 shot setup on mistral

It can be observed that, generally, the performance improvement among the different RAG setups corresponds to the effectiveness of the retrievers, as seen in Section 5.2.1. BM25, on both corpora, consistently achieves the highest scores in both listwise and timewise evaluations across all k values. However, there are instances where the performance of retrievers does not directly translate to downstream performance. For example, BM25 performs better retrieval on the title-summary corpus, but slightly better results are obtained in the RAG setup with the title-infobox-summary corpus. Nevertheless, the difference in performance is negligible and can be attributed to the variations in the documents fetched, besides the one containing the relevant infobox. These variations can affect performance, depending on how 'distracting' the additional documents are.

Furthermore, we observe that only BM25 demonstrates consistent performance improvements across all k values. In contrast, dense retrieval methods, such as all-mini-lm-v6 and multi-qa-mpnet, exhibit a decline in performance from $k = 3$ to $k = 10$. This suggests that the additional documents retrieved by dense methods introduce more 'noise.' 'Noise' in this context could mean documents that are either too similar to the ground truth infobox, causing confusion for the model, or documents that occupy a significant portion of the model's context window, hindering its ability to generate accurate responses. It is also notable that for multi-qa-mpnet, there are no values at $k = 10$ for title-infobox-summary setup. This is because the model ran out of context window space for k values greater than 3, likely due to the retrieval of larger infoboxes that rapidly fill the model's context capacity.

Moreover, as shown in Table 5.1, the best performance of Mistral 0.2 was achieved with KNN prompting, scoring $0.366, 0.435, 0.360$ for F1, Time Overalp and Time Jaccard metrics. Comparing these scores with the results obtained from the RAG setup from Table 5.2, we notice that augmenting the prompt with context fetched by BM25 does improve the performance, across all three metrics, achieving up to $0.624, 0.651, 0.607$ respectively. However, the performance gains are less prominent when using Dense Retrieval approaches, scoring, at most $0.441, 0.436, 0.375$, when using the multi-qa-mpnet embedding over title-infobox-summary corpora. Additionally, comparing with Meta Llama 3's results which scored $0.442, 0.519, 0.428$, we notice that Mistral surpasses this performance only with BM25, and not with dense retrievals. This highlights that high-performing retrieval models can improve the performance of language models that otherwise underperform. Conversely, an underperforming model coupled with suboptimal retrieval performance fails to outperform a higher-performing model.

5.2.3. Golden Evidence and Performance at different K

We further analyze how the performance of RAG systems varies when changing the number of documents retrieved as evidence to be augmented to the query. To ensure the TLQA benchmark is solvable with the provided context, we assume a perfect retrieval system that always retrieves the correct infobox, referred to as golden evidence. This approach allows us to determine the upper-bound performance of RAG setups in our scenario. To assess this, we augment each query with the infobox containing the correct answer and prompt Mistral 0.2 using both manual few-shot and KNN-shot methods. We compare the golden evidence performance of Mistral to the best-performing RAG and prompting setups. The results are summarized in Table 5.3.

Model	Precision	Recall	F1	Time Overlap	Time Jaccard
Closed Book					
Mistral 0.2 KNN	0.532	0.310	0.366	0.435	0.360
Meta Llama 3 KNN	0.548	0.399	0.442	0.519	0.428
Open Book					
Mistral 0.2 + BM-25	0.744	0.575	0.624	0.651	0.607
Mistral 0.2 + all-mini-lm	0.570	0.378	0.430	0.414	0.366
Mistral 0.2 + mutli-qa-mpnet	0.574	0.392	0.441	0.436	0.375
Golden Evidence					
Manual few-shot	0.918	0.770	0.818	0.747	0.715
KNN few-shot	0.941	0.850	0.882	0.857	0.820

Table 5.3: Performance of Mistral 0.2 and Meta Llama 3 models on the benchmark using different prompting techniques. The first three metrics evaluate **listwise completeness** (how complete the answer list compared to the ground truth), whereas the last two assess the **temporal correctness** of the answers provided

We notice that, with both Manual and KNN few-shot prompting, the golden evidence setup achieves the

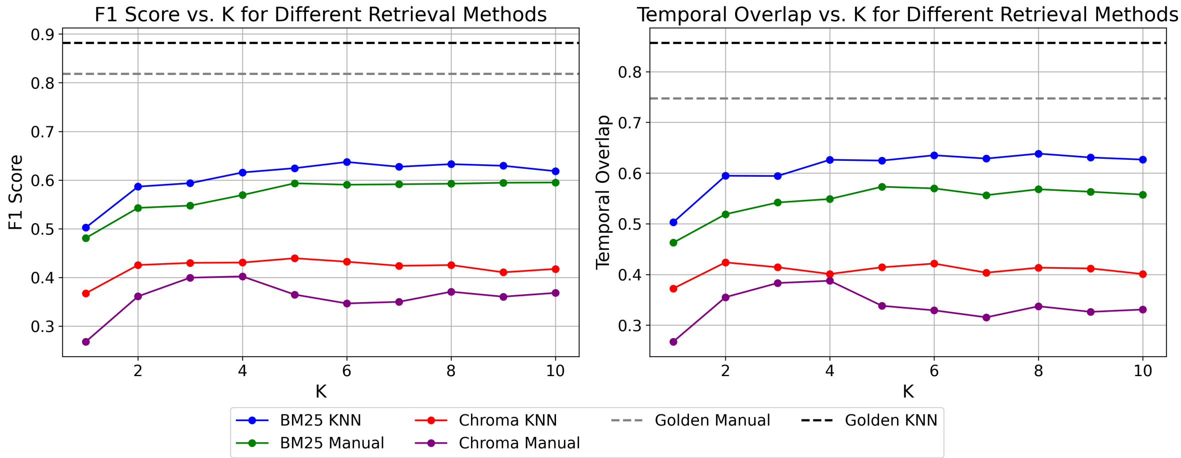


Figure 5.4: Comparison of RAG performance with different retrieval methods on the title-summary corpora. The left plot shows the F1 Score for different retrieval methods as the number of documents fetched (k) varies, while the right plot shows the Temporal Overlap for the same methods. The dotted lines represent the upper-bound performance using perfect retrieval (Golden Manual and Golden KNN).

highest scores. This substantial improvement compared to other setups indicates that the TLQA benchmark is solvable with perfect retrieval, demonstrating that the benchmark is not excessively challenging. Nevertheless, current RAG setups struggle to achieve such performance. Figure 5.4 highlights how the performance of different RAG setups with different retrievers converge compared to the golden evidence. The figure presents the F1 score and Temporal Overlap, while the plot for Time Jaccard, which follows similar trends, can be found in Appendix C. We observe that after $k = 3$, performance improvements are minimal, and a significant gap remains between golden evidence and the retrieval-based setups. This underscores that there is still considerable room for improvement in the retrievers or the models themselves to bridge this performance gap.

Take-Away 2: *Augmenting the query with additional context does seem to improve Mistral's 0.2 performance on the TLQA benchmark. We observe that fetching the infoboxes using BM25 leads to a better performance improvement compared to using dense retrieval. Dense retrievers struggle with our corpora, likely due to their underperformance when dealing with entity-centric queries, or corpora with lacking context. Furthermore, when using a perfect retrieval system that retrieves only the correct infobox, we observe a significant performance increase, surpassing the scores achieved with BM25.*

5.3. Effects of Distractors over Golden Evidence

To address **Research Question 3**, we evaluate how different types of distractors impact the performance of the golden evidence RAG setup. We use three different methods for selecting distractors: randomly selecting k infoboxes from our corpora, retrieving the top k documents using BM25, or retrieving the top k documents using dense retrieval with all-mini-lm-v2 embeddings. We then append the golden evidence along with these k distractors as additional context for the language model to answer the query. The evaluation is conducted using BM25 and dense retrieval with all-mini-lm-v2 on the title-summary corpora. The results are presented in Figure 5.5.

The following observations can be made:

- **Performance Decrease:** It can be observed that performance drops noticeably with the addition of even a single distractor ($k = 1$), indicating that noise affects the model's ability to extract the correct information to answer the query. This effect becomes more evident as we increase the number of distractors (k). While the performance of the model on both F1 and Temporal Overlap can still be considered high' even at $k = 10$, especially compared to the highest results obtained previously (RAG + BM25), there is a clear trend that adding noise decreases performance. This consistent performance degradation can be attributed to the model's capacity limitations; as more distractors are introduced, the cognitive load on the model increases, making it harder

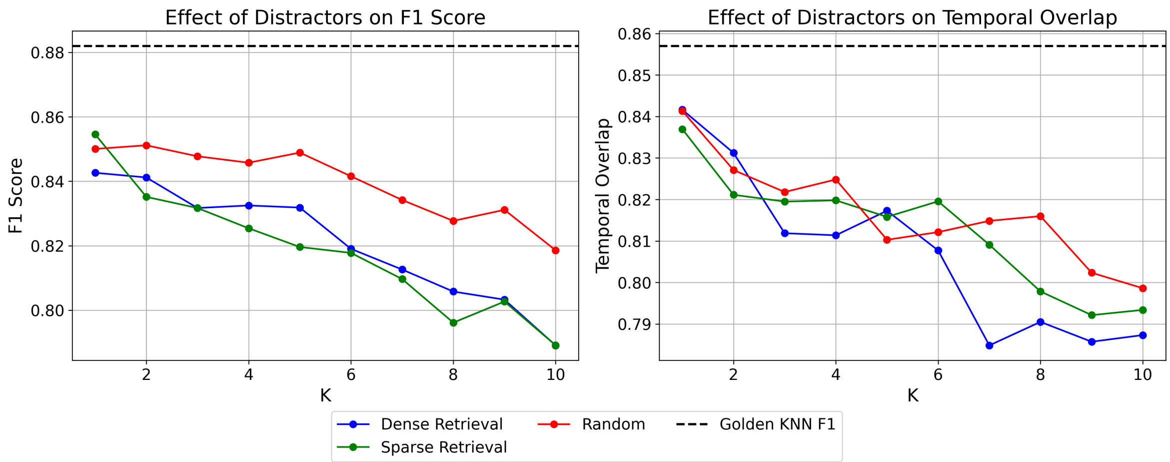


Figure 5.5: Comparison of F1 Score and Temporal Overlap for RAG methods with golden evidence augmented with varying numbers of distractors (k). Distractors are selected either Randomly, using Sparse Retrieval, or using Dense Retrieval. The dashed line represents the score of RAG with golden evidence without any distractors. Evaluation was conducted using KNN 3-shot prompting on Mistral 0.2

to integrate multiple pieces of context effectively. The model's attention mechanism, designed to prioritize relevant information, becomes less effective when faced with numerous distractors. This is because the attention mechanism distributes focus across all provided context, and with an increasing number of distractors, the relevant information receives less attention.

- **Impact of Random vs. Retrieved Distractors:** Randomly selecting infoboxes as distractors appears to decrease performance less significantly than using retrieved distractors. This can be attributed to the fact that randomly chosen infoboxes are less likely to contain information that is contextually similar to the relevant infobox. As a result, the model can more easily distinguish between relevant and irrelevant information. In contrast, retrieved distractors, whether dense or sparse, are selected based on their relevance to the query. This makes them more contextually similar to the correct infobox, thereby making it harder for the attention mechanism to prioritize the correct information.

Take-Away 3: *Distractors do affect the performance of a RAG system that has the golden evidence for each query. We observe a drop in performance both in the listwise F1 score and Temporal Overlap, as we increase the number of distractors added to the context, in addition to the golden infobox. In addition, we notice that selecting random distractors leads to a less significant performance decrease compared to using distractors fetched by a dense or sparse retrieval. Random distractors are less likely to be contextually similar to the relevant infobox, which leads to the model being less affected by them.*

5.4. Question Type-Specific Performance and Error Analysis

We further analyze the results we obtained in depth to find and highlight certain patterns in the mistakes made by the language modes when answering TLQAs.

5.4.1. Performance Based on Question Type

We first evaluate how the performance of our best setups changes depending on the type of questions asked. We split our test data based on the question topic. Given that our TLQA benchmark focused exclusively on political figures and sports players, we categorize the questions accordingly as either sports-related or political-related. We then evaluate the models separately on these subsets. The results are presented in Table 5.4.

We can observe that, generally, the performance, both in terms of listwise and timewise scores, is better for queries of a political nature. This discrepancy can be attributed to the differing number of answers required for sports and political TLQA queries. As shown in Table 3.4, the average number of answers for a sports query is 11.775, while for political queries it is only 5.473 (based on the test set).

Model	Type	Precision	Recall	F1	Temporal Overlap	Time Jaccard
Closed Book						
Mistral KNN	Sports	0.469	0.245	0.308	0.420	0.322
	Political	0.593	0.374	0.423	0.459	0.402
Meta Llama KNN	Sports	0.512	0.357	0.408	0.467	0.358
	Political	0.583	0.440	0.477	0.568	0.493
Open Book						
Mistral BM25	Sports	0.800	0.545	0.630	0.730	0.675
	Political	0.690	0.606	0.619	0.571	0.535
Mistral all-mini-lm	Sports	0.516	0.294	0.358	0.419	0.350
	Political	0.628	0.463	0.503	0.424	0.393
Mistral Multi QA	Sports	0.517	0.312	0.373	0.433	0.348
	Political	0.634	0.474	0.511	0.445	0.411
Golden Evidence						
Mistral Golden Manual	Sports	0.941	0.661	0.763	0.862	0.818
	Political	0.898	0.876	0.874	0.634	0.613
Mistral Golden KNN	Sports	0.935	0.760	0.828	0.897	0.847
	Political	0.949	0.940	0.935	0.816	0.792

Table 5.4: Performance of various models on the benchmark using different prompting techniques, categorized by question type (Sports and Political)

Consequently, the high number of required answers in sports queries introduces more opportunities for errors, making it challenging to achieve completeness, which is reflected in the consistently lower Recall scores for sports compared to political queries. Additionally, the higher number of answers requires managing more timelines, as each answer has at least one timeline, increasing the complexity and potential for mistakes in temporal alignment.

However, there are instances where the results of political queries show lower temporal scores compared to sports queries results, specifically in the cases of Mistral BM25, Golden Manual, and Golden KNN. One possible explanation could be that political timelines may inherently be more complex than those of sports. Political careers often involve overlapping roles, non-linear career progressions, and a variety of positions held simultaneously or in quick succession, contributing to a higher temporal complexity. For instance, a politician might serve in multiple capacities such as a minister, a party leader, and a parliament member simultaneously, each with distinct timelines. In contrast, sports timelines are generally more straightforward, as players typically switch teams less frequently and often play for a single club per year.

5.4.2. Qualitative Analysis

We present and analyze common mistakes that language models exhibit when tasked with TLQA queries.

A frequent issue is the model's inability to answer the query, as exemplified in Table 5.5. In this instance, Meta Llama 3, prompted in a manual 3-shot setting, explicitly states that it cannot find any information on Alan Kirby. This refusal occurs because the model does not recognize the entity, indicating lack of knowledge. In contrast, when using a KNN 10-shot setup for the same query, another type of problem can be noticed. The model attempts to provide an answer but hallucinates all entries, as shown in Table 5.6. This behavior likely occurs because the KNN setup selects only TLQA samples containing football players for few-shot learning, which strongly suggests that Alan Kirby is a football player. Moreover, all ten players selected as examples are either English or Scottish and played for English teams. In addition, five teams from these examples appear in the model's response. This indicates that KNN

➤ **Query**: `List all teams Alan Kirby played to this day.`	
Expected Answer	**Generated Answer**
• `Aston Villa F.C.: 1996-1998, 1994-1996` • `Johnville F.C. (Waterford): 1994` • `Republic of Ireland under-21 national football team: 1998` • `Waterford United F.C.: 1999-2001` • `League of Ireland XI: 2010` • `Longford Town F.C.: 2001-2006, 2012` • `St Patrick's Athletic F.C.: 2007-2008` • `Sporting Fingal F.C.: 2009-2010` • `Sligo Rovers F.C.: 2011`	`I apologize, but I couldn't find any information on a person named Alan Kirby playing for any football teams. If you meant someone else or provided incorrect information, please let me know and I'll be happy to help.`

Table 5.5: Example of query that the model is not able to answer. Generated with Meta Llama 3 with manual 3-shot prompting.

➤ **Query**: `List all teams Alan Kirby played to this day.`	
Expected Answer	**Model Answer**
• `Aston Villa F.C.: 1996-1998, 1994-1996` • `Johnville F.C. (Waterford): 1994` • `Republic of Ireland under-21 national football team: 1998` • `Waterford United F.C.: 1999-2001` • `League of Ireland XI: 2010` • `Longford Town F.C.: 2001-2006, 2012` • `St Patrick's Athletic F.C.: 2007-2008` • `Sporting Fingal F.C.: 2009-2010` • `Sligo Rovers F.C.: 2011`	• `Leeds United F.C.: 1995-2001, 2000` • `England national under-21 football team: 1999` • `Huddersfield Town A.F.C.: 2001-2004` • `Rotherham United F.C.: 2004-2006` • `Doncaster Rovers F.C.: 2006-2008` • `Grimsby Town F.C.: 2008-2010` • `Bradford City A.F.C.: 2010-2011` • `Gainsborough Trinity F.C.: 2011-2012` • `Harrogate Railway Athletic F.C.: 2012-2013` • `Farsley Celtic F.C.: 2013-2014` • `York City F.C.: 2014-2015` • `Brighouse Town A.F.C.: 2015-2016` • `AFC Emley F.C.: 2016-2017` • `Garforth Town A.F.C.: 2017-2018` • `Liversedge F.C.: 2018`

Table 5.6: Example of model answer with incorrect details. Same query and model as in Table 5.5, however with a knn-10 shot setting.

influences the model, causing it to hallucinate and create plausible but incorrect answers. For example, `Leeds United F.C.` appears three times in the examples given, and `England national under-21 football team` appears four times, which could be a reason why those appear first in model's answers. Furthermore, the model's answer timeline appears plausible, as it generally follows the pattern of a single active club per year with incremental years, suggesting a continuous career. This pattern likely results from the model detecting trends in the ten provided examples. However, the first entry contains an inconsistency where it lists "1995-2001" followed by "2000," even though 2000 falls within the 1995-2001 range, which could suggest that the model may struggle to understand time.

Another common type of mistake the models display is providing incomplete answers. As previously mentioned in Section 5.1, we observe that the models scored significantly lower in listwise Recall compared to other metrics, indicating that they may face challenges in providing complete responses. An example of this case is seen in Table 5.7. We observe that out of the six answers in the expected answers, the model only generates two out of them. This issue can be attributed to both **popularity**

bias and **recency bias**. Popularity bias refers to the model's tendency to favor more well-known or frequently mentioned entities or facts in the training data, whereas recency bias refers to the model's inclination to focus on more recent information, neglecting older data. In the selected examples, recency bias is be observed as the model's answer did not include older political positions, such as `Chairman of the Moderate Youth League` or `Minister for Social Security`. Moreover, popularity bias is evident as the model omits the position `Leader of the Opposition`, which has similar timelines to the positions it did generate. Both these biases likely appear due to artifacts from the model's training data, which often contains a disproportionate amount of information on more recent and well-known entities and facts.

➤ **Query**: `List all political positions Ulf Kristersson held to this day.`	
Expected Answer	**Model Answer**
• `Prime Minister of Sweden: 2022-2024` • `Leader of the Moderate Party: 2017-2024` • `Leader of the Opposition: 2017-2022` • `Minister for Social Security: 2010-2014` • `Chairman of the Moderate Youth League: 1988-1992` • `Member of the Riksdag: 2014-2024, 1991-2000`	• `Leader of the Moderate Party: 2017-2024` • `Member of the Riksdag: 2006-2021, 2022-2024`

Table 5.7: Example of incomplete answer

Temporally, a recurrent issue that language models face when addressing TLQA queries is aligning the time intervals accurately. This can be observed in the same example in Table 5.7. While the first entry correctly matches the timeline, the second entry contains partially incorrect timelines. We observe that the model struggles with temporally bounding the timeline of the political position, completely missing the earlier interval (1991-2000), starting the other timeline earlier (2006 instead of 2014) and creating gaps where there should not be any (the gap between 2021-2022). This behavior highlights that models struggle with reasoning over time. This may occur due to several factors, including the model's difficulty in understanding and integrating non-linear career paths and its reliance on fragmented information from the training data. Furthermore, the training data might lack comprehensive temporal annotations, leading the model to generate imprecise timelines.

Take-Away 4: *Performance on political queries is, on average, higher than on sports-related queries due to the higher number of possible answers in sports queries. Moreover, we highlight several common pitfalls in TLQA models. Firstly, the model may refuse to answer or hallucinate due to a lack of knowledge. Secondly, the model often fails to provide complete answers, exhibiting popularity or recency bias. Finally, the model struggles to correctly align timelines, indicating difficulty with temporally bounding facts.*

6 Discussions and Conclusion

In this chapter, we reflect upon the work and the results presented, highlighting limitations, and potential future research directions.

6.1. Discussion

The goal of this thesis is to propose a new QA task and benchmark that evaluates how well models perform when dealing with queries requiring both listwise and timewise comprehension. We propose an automatic pipeline for generating questions and answers, using entities that are either office holders, football players, or cricketers, according to Wikipedia's infobox categorizations. While we use only entities selected from TempLAMA, our method of using infoboxes to generate TLQAs is applicable to any entity with the mentioned types of infoboxes.

Our findings indicate that generative language models often struggle with the benchmark. Particularly, we observe that both Mistral 0.2 and Meta Llama 3 8b consistently score low in listwise Recall, suggesting that they struggle to provide complete answers. We further notice that the models have difficulties generating correct timelines, often hallucinating outside the expected year range, or not covering the whole expected year set.

We observe that incorporating RAG does improve performance, but only if the retriever performs well. We notice how sparse retrieval, represented by BM25, does increase the performance of Mistral 0.2 substantially. On the other hand, dense retrieval shows suboptimal performance, likely due to the embedding's inability to encode entity names, aligning with the findings of Sciavolino et al. [57]. Interesting to highlight is how Meta Llama 3 performs better in KNN few-shot closed-book setting compared to Mistral 0.2 augmented with context fetched with dense retrieval.

Finally, when provided with golden evidence, Mistral 0.2 achieves exceptionally high scores, both listwise and timewise, demonstrating that the TLQA benchmark is indeed solvable. Additionally, introducing distractors affects performance both temporally and listwise. Random distractors have a lesser impact than those selected via retrieval, indicating that the model is more distracted by infoboxes that closely resemble the golden one.

6.2. Limitations

Different types of limitations can be highlighted in our work.

Benchmark limitations: The benchmark is relatively small compared to other publicly available ListQA datasets such as Liquid [19] or Qampari [18]. It contains only 1655 entries in total, with the test set used for experiments comprising only 497 entries. Moreover, the benchmark lacks diversity, covering only two topics: sports and politics. This restricts the generalizability of our findings to a broader range of subjects. Finally, since our benchmark is created through a fully automated methodology, there is a lack of quality control. This makes it challenging to guarantee the accuracy and validity of the questions and answers.

Evaluation limitation: The accuracy of the entity matching pipeline is not evaluated, making it difficult to determine if this is the optimal approach. The pipeline may introduce false positives or false negatives in the matched entities set, which could affect the reliability of the evaluation results.

Experimental limitation: Our experiments are limited to ”small” (7-8 billion parameters) open-source generative language models. However, open-source models with more parameters and some closed-source models such as GPT-4 [38] have demonstrated superior performance in multiple settings compared to 7-8b parameters models. Additionally, since generative models are used, more runs may be required to achieve reliable results, even with low-temperature settings. Finally, we do not quantify the frequency of different types of mistakes made by the models, which limits the analysis of error patterns and their implications.

6.3. Future Works

There are several potential directions for future research to improve this work.

Proposed Solutions: Firstly, while our TLQA benchmark demonstrates that generative language models struggle with this type of QA task, we do not propose specific solutions to enhance their performance. Although a brute-force approach involving training larger models with improved data quality and pre-training processes is an evident option, alternative strategies should also be explored. In an open-book scenario, one idea could be to first prompt the model to identify the infobox that could serve as the golden one, and then extract the answer from there. In a closed-book scenario, a multi-branch inference approach could be implemented, where multiple potential answers are generated using probabilistic sampling techniques such as nucleus sampling or diverse beam search. Applying set operations such as intersection and union to the generated answer sets could lead to more complete or factually accurate answers. This approach could especially improve the performance of models with small context window, such as Meta Llama 3, that cannot otherwise employ RAG based solutions.

Benchmark improvement: Secondly, our benchmark can be improved in several ways. Instead of using TempLAMA as a seed, we could use the Wikipedia corpus we curated to generate a larger set of QAs, by applying our designed parser to all applicable infoboxes (officeholder, football biography, and politics). A study assessing the infoboxes with the most temporal diversity could also be a valuable future direction, as we currently rely on TempLAMA's findings to select temporally diverse infobox types. Another potential improvement could involve implementing different temporal splits. Instead of listing information up to the present day, we could request information for specific time frames to evaluate how well the model handles temporal information. This would better assess the model's understanding of time, as it would need to interpret partial time intervals mentioned in the training data. Finally, manual annotation and quality checking should be performed to ensure the accuracy of autogenerated answers and queries.

Improved Wikipedia corpus: Regarding our Wikipedia corpus, we currently use the lead section as a representative summary of each article. However, we should create summaries that encompass all information from the infoboxes. By extracting sentences from the Wikipedia articles that contain entities mentioned in the infoboxes, we could write custom summaries to ensure that all answers are included. This approach would allow us to test the model's ability to find the correct answer within unstructured text, as opposed to relying solely on infoboxes.

Entity Matching: Future iterations of this study should explore alternative entity matching approaches. Entity matching remains a challenging task, and an efficient, definitive solution is not yet clear [58]. A promising approach is presented by Bulian et al. [59], where the authors train language models to predict whether two answers are equivalent. A similar method could be developed specifically for named entities.

Extensive experimets: As mentioned in Section 6.1, future iterations of this study should experiment with larger models to determine if the performance issues observed in smaller models persist. In addition, more complex RAG setups could be considered, as illustrated in Figure 6.1.

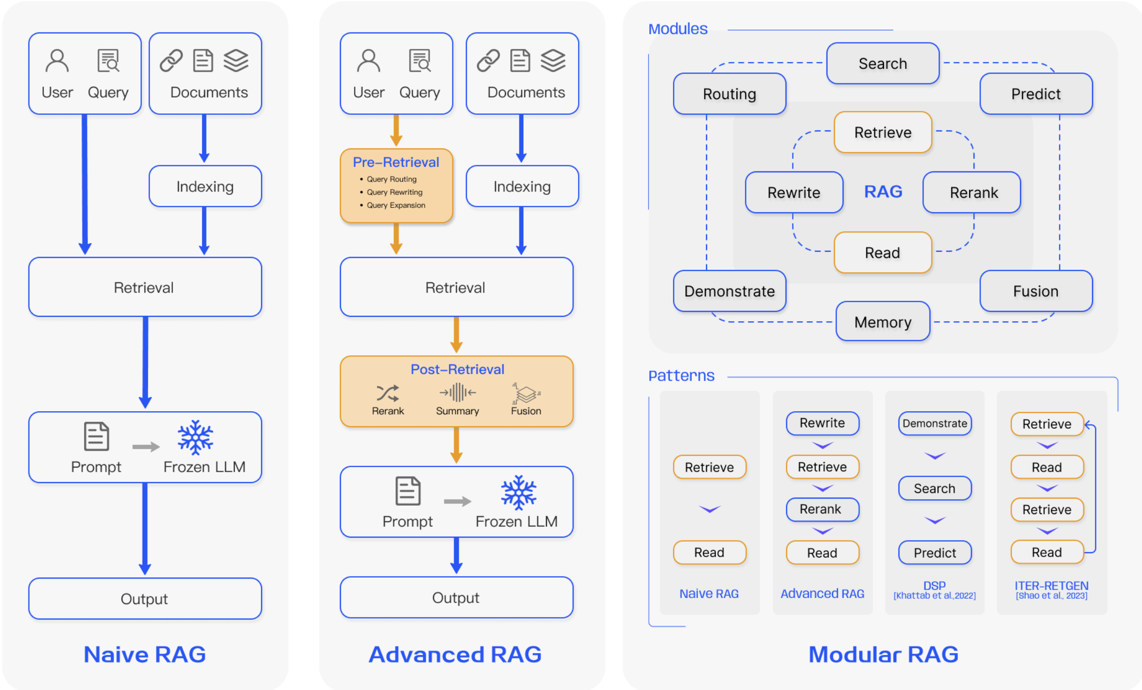


Figure 6.1: Examples of different RAG paradigms. Currently, our work falls into the 'Naive RAG' category. Future research should aim to enhance the setup to align with more advanced RAG paradigms. Figure extracted from Gao et al [23]

6.4. Conclusion

Our proposed TLQA task and benchmark highlight performance issues in generative language models when dealing with timeline questions. We observe that the models struggle to achieve completeness, often generating only more well-known answers. Additionally, the models have difficulty generating accurate timelines, frequently hallucinating and failing to correctly time-bound information. These findings underscore the need for further research into improving the temporal reasoning and list comprehension capabilities of generative models.

References

[1] Bert F. Green et al. "Baseball: an automatic question-answerer". In: *Papers Presented at the May 9-11, 1961, Western Joint IRE-AIEE-ACM Computer Conference*. IRE-AIEE-ACM '61 (Western). Los Angeles, California: Association for Computing Machinery, 1961, pp. 219–224. ISBN: 9781450378727. DOI: 10.1145/1460690.1460714. URL: https://doi.org/10.1145/1460690.1460714.

[2] Pranav Rajpurkar et al. "SQuAD: 100,000+ Questions for Machine Comprehension of Text". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Ed. by Jian Su, Kevin Duh, and Xavier Carreras. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2383–2392. DOI: 10.18653/v1/D16-1264. URL: https://aclanthology.org/D16-1264.

[3] Xiaodong Liu et al. "Stochastic Answer Networks for Machine Reading Comprehension". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Iryna Gurevych and Yusuke Miyao. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 1694–1704. DOI: 10.18653/v1/P18-1157. URL: https://aclanthology.org/P18-1157.

[4] Fengbin Zhu et al. *Retrieving and Reading: A Comprehensive Survey on Open-domain Question Answering*. 2021. arXiv: 2101.00774 [cs.AI].

[5] Karan Singhal et al. "Towards Expert-Level Medical Question Answering with Large Language Models". In: *CoRR* abs/2305.09617 (2023). DOI: 10.48550/ARXIV.2305.09617. arXiv: 2305.09617. URL: https://doi.org/10.48550/arXiv.2305.09617.

[6] Vaibhav Mavi, Anubhav Jangra, and Adam Jatowt. *A Survey on Multi-hop Question Answering and Generation*. 2022. arXiv: 2204.09140 [cs.CL].

[7] Wonjin Yoon et al. "Sequence tagging for biomedical extractive question answering". In: *Bioinform.* 38.15 (2022), pp. 3794–3801. DOI: 10.1093/BIOINFORMATICS/BTAC397. URL: https://doi.org/10.1093/bioinformatics/btac397.

[8] Kaushik Chakrabarti et al. "TableQnA: Answering List Intent Queries With Web Tables". In: *CoRR* abs/2001.04828 (2020). arXiv: 2001.04828. URL: https://arxiv.org/abs/2001.04828.

[9] Jiexin Wang, Adam Jatowt, and Masatoshi Yoshikawa. *ArchivalQA: A Large-scale Benchmark Dataset for Open Domain Question Answering over Historical News Collections*. 2022. arXiv: 2109.03438 [cs.CL].

[10] Wenhu Chen, Xinyi Wang, and William Yang Wang. *A Dataset for Answering Time-Sensitive Questions*. 2021. arXiv: 2108.06314 [cs.CL].

[11] Zhen Jia et al. "TempQuestions: A Benchmark for Temporal Question Answering". In: *Companion Proceedings of the The Web Conference 2018*. WWW '18. Lyon, France: International World Wide Web Conferences Steering Committee, 2018, pp. 1057–1062. ISBN: 9781450356404. DOI: 10.1145/3184558.3191536. URL: https://doi.org/10.1145/3184558.3191536.

[12] Zhen Jia et al. "Complex Temporal Question Answering on Knowledge Graphs". In: *Proceedings of the 30th ACM International Conference on Information amp; Knowledge Management*. CIKM '21. ACM, Oct. 2021. DOI: 10.1145/3459637.3482416. URL: http://dx.doi.org/10.1145/3459637.3482416.

[13] Ziyang Chen, Jinzhi Liao, and Xiang Zhao. "Multi-granularity Temporal Question Answering over Knowledge Graphs". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 11378–11392. DOI: 10.18653/v1/2023.acl-long.637. URL: https://aclanthology.org/2023.acl-long.637.

[14] Costas Mavromatis et al. *TempoQR: Temporal Question Reasoning over Knowledge Graphs*. 2021. arXiv: `2112.05785 [cs.CL]`.

[15] Chao Shang et al. "Improving Time Sensitivity for Question Answering over Temporal Knowledge Graphs". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 8017–8026. DOI: `10.18653/v1/2022.acl-long.552`. URL: `https://aclanthology.org/2022.acl-long.552`.

[16] Pranav Rajpurkar et al. "SQuAD: 100, 000+ Questions for Machine Comprehension of Text". In: *CoRR* abs/1606.05250 (2016). arXiv: `1606.05250`. URL: `http://arxiv.org/abs/1606.05250`.

[17] Chen Zhang et al. "How Many Answers Should I Give? An Empirical Study of Multi-Answer Reading Comprehension". In: *CoRR* abs/2306.00435 (2023). DOI: `10.48550/ARXIV.2306.00435`. arXiv: `2306.00435`. URL: `https://doi.org/10.48550/arXiv.2306.00435`.

[18] Samuel Joseph Amouyal et al. "QAMPARI: : An Open-domain Question Answering Benchmark for Questions with Many Answers from Multiple Paragraphs". In: *CoRR* abs/2205.12665 (2022). DOI: `10.48550/ARXIV.2205.12665`. arXiv: `2205.12665`. URL: `https://doi.org/10.48550/arXiv.2205.12665`.

[19] Seongyun Lee, Hyunjae Kim, and Jaewoo Kang. "LIQUID: A Framework for List Question Answering Dataset Generation". In: *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*. Ed. by Brian Williams, Yiling Chen, and Jennifer Neville. AAAI Press, 2023, pp. 13014–13024. DOI: `10.1609/AAAI.V37I11.26529`. URL: `https://doi.org/10.1609/aaai.v37i11.26529`.

[20] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[21] Ziwei Ji et al. "Survey of Hallucination in Natural Language Generation". In: *ACM Computing Surveys* 55.12 (Mar. 2023), pp. 1–38. DOI: `10.1145/3571730`. URL: `https://doi.org/10.1145%2F3571730`.

[22] Patrick S. H. Lewis et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. 2020. URL: `https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html`.

[23] Yunfan Gao et al. "Retrieval-Augmented Generation for Large Language Models: A Survey". In: *CoRR* abs/2312.10997 (2023). DOI: `10.48550/ARXIV.2312.10997`. arXiv: `2312.10997`. URL: `https://doi.org/10.48550/arXiv.2312.10997`.

[24] Johannes Hoffart et al. "YAGO2: exploring and querying world knowledge in time, space, context, and many languages". In: *Proceedings of the 20th International Conference Companion on World Wide Web*. WWW '11. Hyderabad, India: Association for Computing Machinery, 2011, pp. 229–232. ISBN: 9781450306379. DOI: `10.1145/1963192.1963296`. URL: `https://doi.org/10.1145/1963192.1963296`.

[25] Simon Gottschalk and Elena Demidova. *EventKG: A Multilingual Event-Centric Temporal Knowledge Graph*. 2018. arXiv: `1804.04526 [cs.CL]`.

[26] Nattiya Kanhabua and Avishek Anand. "Temporal Information Retrieval". In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '16. Pisa, Italy: Association for Computing Machinery, 2016, pp. 1235–1238. ISBN: 9781450340694. DOI: `10.1145/2911451.2914805`. URL: `https://doi.org/10.1145/2911451.2914805`.

[27] Ricardo Campos et al. "Survey of Temporal Information Retrieval and Related Applications". In: *ACM Comput. Surv.* 47.2 (Aug. 2014). ISSN: 0360-0300. DOI: `10.1145/2619088`. URL: `https://doi.org/10.1145/2619088`.

[28] Jungbin Son and Alice Oh. “Time-Aware Representation Learning for Time-Sensitive Question Answering”. In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 70–77. DOI: 10.18653/v1/2023.findings-emnlp.6. URL: https://aclanthology.org/2023.findings-emnlp.6.

[29] Jiabao Chen and Yongquan Fan. “Improving temporal question answering using temporal knowledge graph embedding”. In: *2023 4th International Conference on Computer Engineering and Application (ICCEA)*. 2023, pp. 570–575. DOI: 10.1109/ICCEA58433.2023.10135342.

[30] Apoorv Saxena, Soumen Chakrabarti, and Partha Talukdar. “Question Answering Over Temporal Knowledge Graphs”. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Ed. by Chengqing Zong et al. Online: Association for Computational Linguistics, Aug. 2021, pp. 6663–6676. DOI: 10.18653/v1/2021.acl-long.520. URL: https://aclanthology.org/2021.acl-long.520.

[31] Ziyang Chen et al. “Temporal knowledge graph question answering via subgraph reasoning”. In: *Knowledge-Based Systems* 251 (2022), p. 109134. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2022.109134. URL: https://www.sciencedirect.com/science/article/pii/S0950705122005603.

[32] Jungo Kasai et al. “RealTime QA: What’s the Answer Right Now?” In: *ArXiv* abs/2207.13332 (2022). URL: https://api.semanticscholar.org/CorpusID:251105205.

[33] Anastasios Nentidis et al. “Overview of BioASQ 2023: The Eleventh BioASQ Challenge on Large-Scale Biomedical Semantic Indexing and Question Answering”. In: *Experimental IR Meets Multilinguality, Multimodality, and Interaction - 14th International Conference of the CLEF Association, CLEF 2023, Thessaloniki, Greece, September 18-21, 2023, Proceedings*. Ed. by Avi Arampatzis et al. Vol. 14163. Lecture Notes in Computer Science. Springer, 2023, pp. 227–250. DOI: 10.1007/978-3-031-42448-9_19. URL: https://doi.org/10.1007/978-3-031-42448-9%5C_19.

[34] Richard C. Wang et al. “Automatic Set Expansion for List Question Answering”. In: *2008 Conference on Empirical Methods in Natural Language Processing, EMNLP 2008, Proceedings of the Conference, 25-27 October 2008, Honolulu, Hawaii, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2008, pp. 947–954. URL: https://aclanthology.org/D08-1099/.

[35] Alon Talmor and Jonathan Berant. “The Web as a Knowledge-Base for Answering Complex Questions”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*. Ed. by Marilyn A. Walker, Heng Ji, and Amanda Stent. Association for Computational Linguistics, 2018, pp. 641–651. DOI: 10.18653/V1/N18-1059. URL: https://doi.org/10.18653/v1/n18-1059.

[36] Victor Zhong et al. “RoMQA: A Benchmark for Robust, Multi-evidence, Multi-answer Question Answering”. In: *CoRR* abs/2210.14353 (2022). DOI: 10.48550/ARXIV.2210.14353. arXiv: 2210.14353. URL: https://doi.org/10.48550/arXiv.2210.14353.

[37] Wen-tau Yih et al. “The Value of Semantic Parse Labeling for Knowledge Base Question Answering”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*. The Association for Computer Linguistics, 2016. DOI: 10.18653/V1/P16-2033. URL: https://doi.org/10.18653/v1/p16-2033.

[38] OpenAI. “GPT-4 Technical Report”. In: *CoRR* abs/2303.08774 (2023). DOI: 10.48550/ARXIV.2303.08774. arXiv: 2303.08774. URL: https://doi.org/10.48550/arXiv.2303.08774.

[39] Rohan Anil et al. “PaLM 2 Technical Report”. In: *CoRR* abs/2305.10403 (2023). DOI: 10.48550/ARXIV.2305.10403. arXiv: 2305.10403. URL: https://doi.org/10.48550/arXiv.2305.10403.

[40] Hugo Touvron et al. “LLaMA: Open and Efficient Foundation Language Models”. In: *CoRR* abs/2302.13971 (2023). DOI: 10.48550/ARXIV.2302.13971. arXiv: 2302.13971. URL: https://doi.org/10.48550/arXiv.2302.13971.

[41] Ehsan Kamalloo et al. “Evaluating Open-Domain Question Answering in the Era of Large Language Models”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*. Ed. by Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki. Association for Computational Linguistics, 2023, pp. 5591–5606. DOI: 10.18653/V1/2023.ACL-LONG.307. URL: https://doi.org/10.18653/v1/2023.acl-long.307.

[42] Harsha Nori et al. “Capabilities of GPT-4 on Medical Challenge Problems”. In: *CoRR* abs/2303.13375 (2023). DOI: 10.48550/ARXIV.2303.13375. arXiv: 2303.13375. URL: https://doi.org/10.48550/arXiv.2303.13375.

[43] Joshua Maynez et al. “On Faithfulness and Factuality in Abstractive Summarization”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 1906–1919.

[44] Ziwei Ji et al. “Survey of hallucination in natural language generation”. In: *ACM Computing Surveys* 55.12 (2023), pp. 1–38.

[45] Bhuwan Dhingra et al. “Time-Aware Language Models as Temporal Knowledge Bases”. In: *Transactions of the Association for Computational Linguistics* 10 (Mar. 2022), pp. 257–273. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00459. eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00459/2004543/tacl_a_00459.pdf. URL: https://doi.org/10.1162/tacl%5C_a%5C_00459.

[46] Spencer Kelly. *Dumpster-dip: wikipedia dump parser*. https://github.com/spencermountain/dumpster-dip. 2024.

[47] Albert Q Jiang et al. “Mistral 7B”. In: *arXiv preprint arXiv:2310.06825* (2023).

[48] Meta AI. *Meta Llama 3: Advancing AI Technology*. 2024. URL: https://ai.meta.com/blog/meta-llama-3/.

[49] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon et al. 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[50] Joshua Ainslie et al. “GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Association for Computational Linguistics, 2023, pp. 4895–4901. DOI: 10.18653/V1/2023.EMNLP-MAIN.298. URL: https://doi.org/10.18653/v1/2023.emnlp-main.298.

[51] Jiachang Liu et al. “What Makes Good In-Context Examples for GPT-3?” In: *Proceedings of Deep Learning Inside Out: The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures, DeeLIO@ACL 2022, Dublin, Ireland and Online, May 27, 2022*. Ed. by Eneko Agirre, Marianna Apidianaki, and Ivan Vulic. Association for Computational Linguistics, 2022, pp. 100–114. DOI: 10.18653/V1/2022.DEELIO-1.10. URL: https://doi.org/10.18653/v1/2022.deelio-1.10.

[52] Jason Wei et al. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”. In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. Ed. by Sanmi Koyejo et al. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

[53] Zhuosheng Zhang et al. “Automatic Chain of Thought Prompting in Large Language Models”. In: *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL: https://openreview.net/pdf?id=5NTt8GFjUHkr.

[54] Long Ouyang et al. “Training language models to follow instructions with human feedback”. In: *Advances in neural information processing systems* 35 (2022), pp. 27730–27744.

[55] Florian Le Bronnec et al. “Exploring Precision and Recall to assess the quality and diversity of LLMs”. In: *62nd Annual Meeting of the Association for Computational Linguistics*. 2024.

[56] Hugging Face. *Mistral 7B Instruct v0.2*. 2024. URL: https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2.

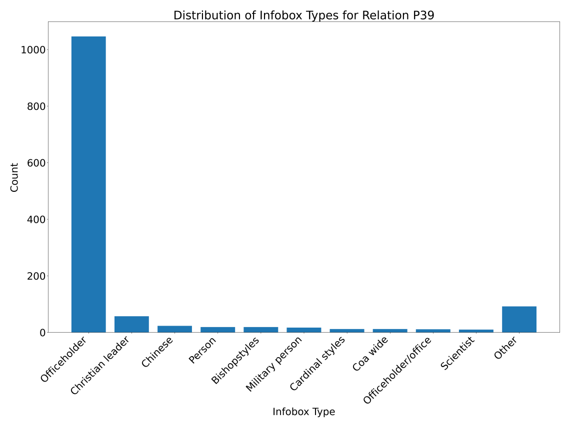
[57] Christopher Sciavolino et al. “Simple Entity-Centric Questions Challenge Dense Retrievers”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021, pp. 6138–6148.

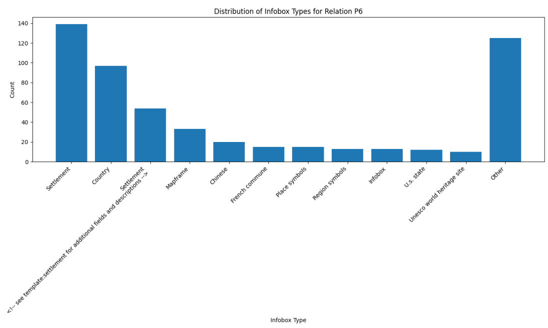
[58] Nils Barlaug and Jon Atle Gulla. “Neural networks for entity matching: A survey”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15.3 (2021), pp. 1–37.

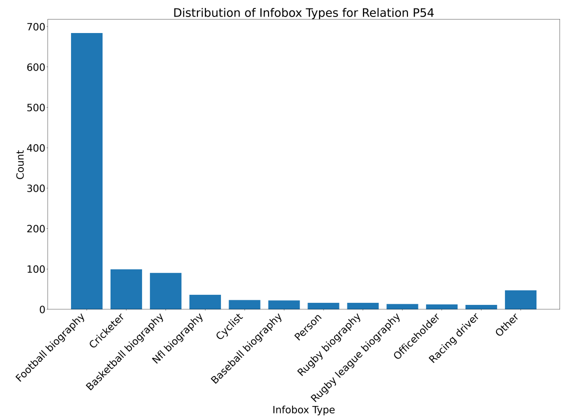
[59] Jannis Bulian et al. “Tomayto, Tomahto. Beyond Token-level Answer Equivalence for Question Answering Evaluation”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*. Ed. by Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang. Association for Computational Linguistics, 2022, pp. 291–305. DOI: 10.18653/V1/2022.EMNLP-MAIN.20. URL: https://doi.org/10.18653/v1/2022.emnlp-main.20.

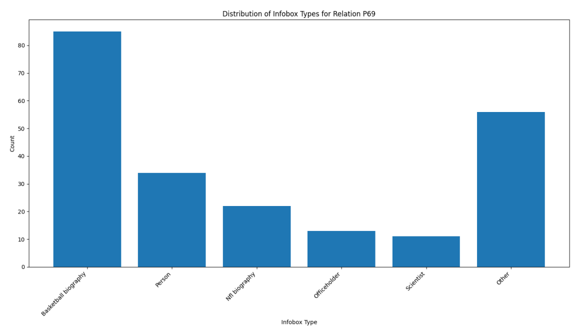
A

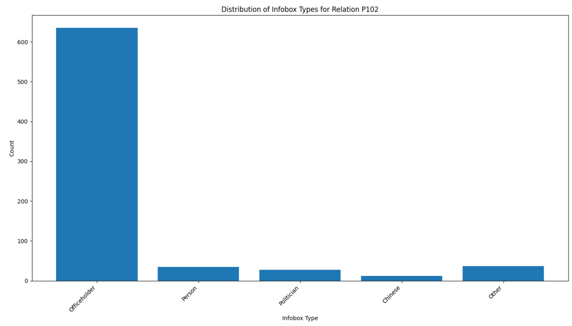
TempLAMA infobox distributions

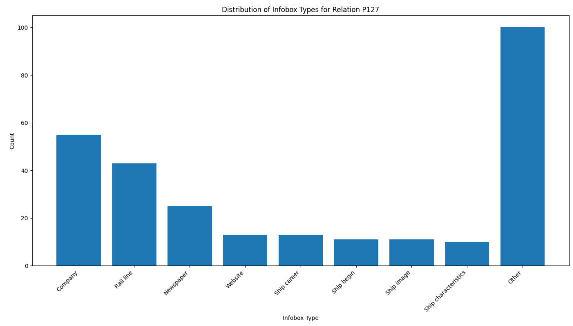


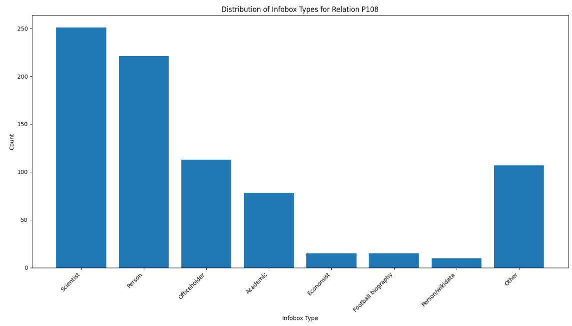


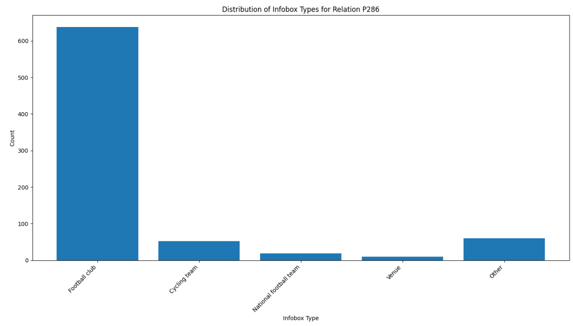


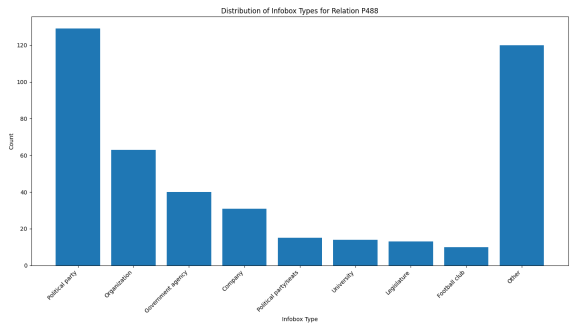



Distribution of Infobox Types for Relation P102
Count
0
100
200
300
400
500
600
Officeholder
Person
Politician
Chinese
Other
Infobox Type


Distribution of Infobox Types for Relation P127
Count
0
20
40
60
80
100
Company
Rail line
Newspaper
Website
Ship career
Ship begin
Ship image
Ship characteristics
Other
Infobox Type


Distribution of Infobox Types for Relation P108
Count
0
50
100
150
200
250
Scientist
Person
Officeholder
Academic
Economist
Football biography
Person/wikidata
Other
Infobox Type


Distribution of Infobox Types for Relation P286
Count
0
100
200
300
400
500
600
Football club
Cycling team
National football team
Venue
Other
Infobox Type


Distribution of Infobox Types for Relation P488
Count
0
20
40
60
80
100
120
Political party
Organization
Government agency
Company
Political party/seats
University
Legislature
Football club
Other
Infobox Type

B

Infoboxes

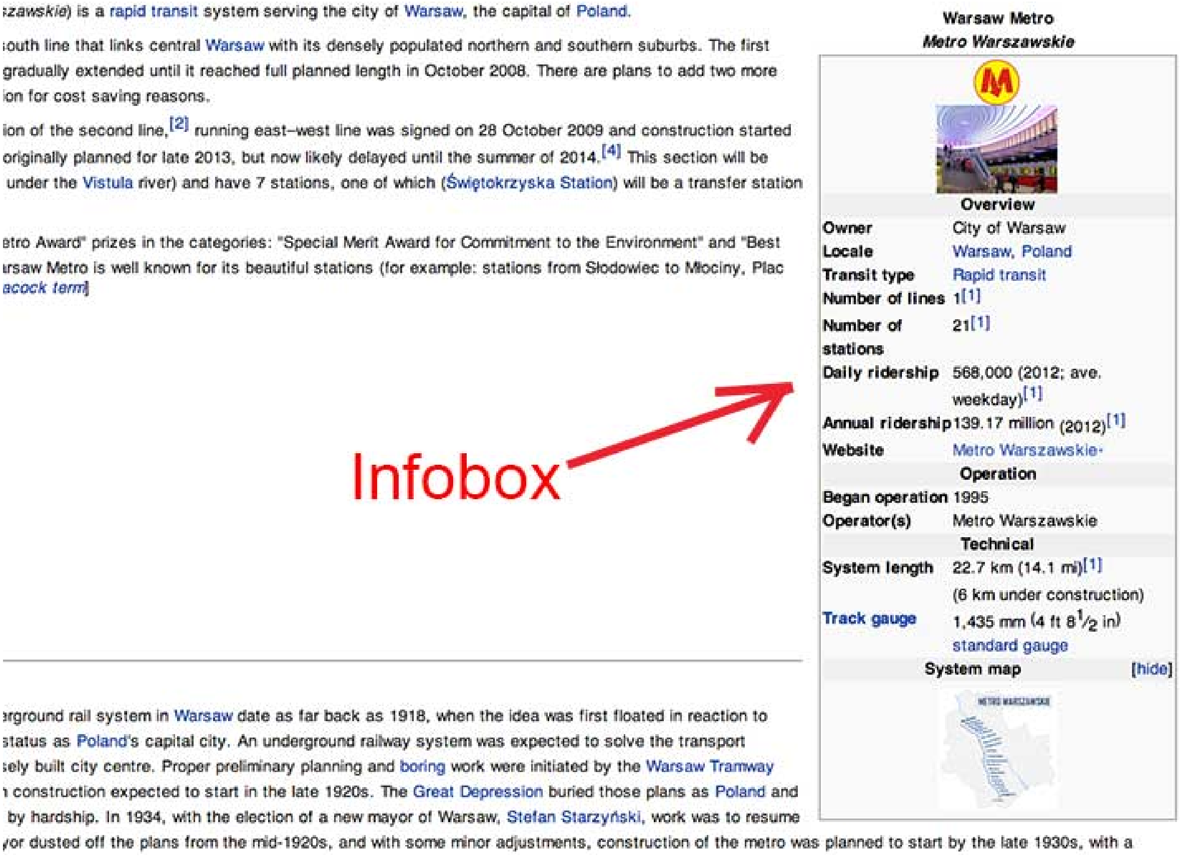
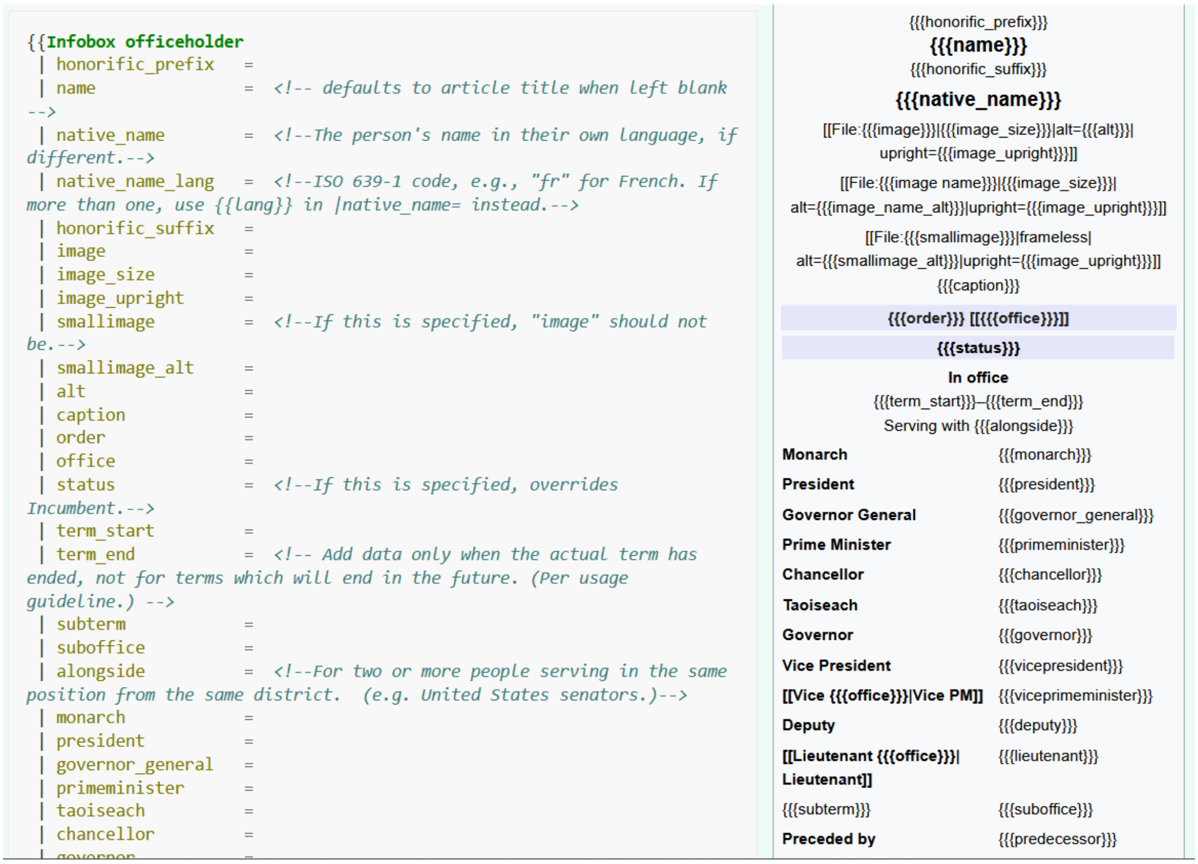


Figure B.1: Example of an infobox ***Source***

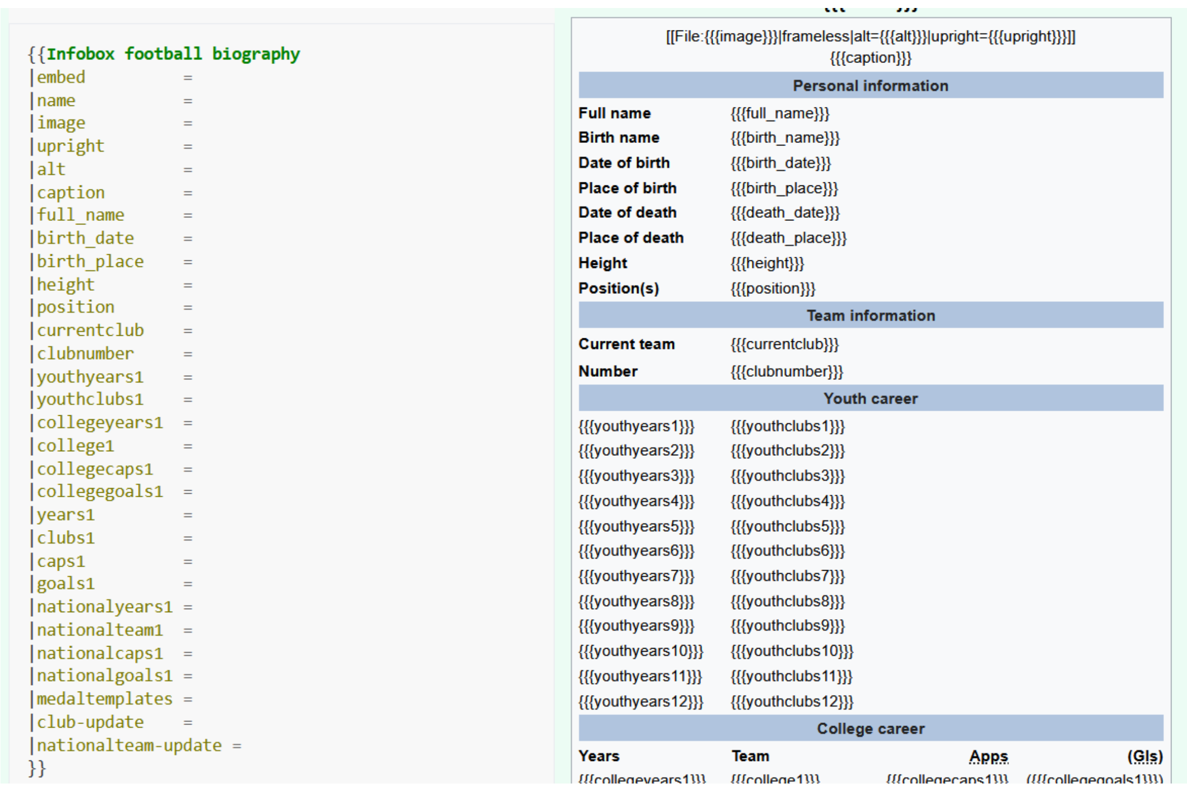


```
{{Infobox officeholder
 | honorific_prefix   =
 | name               = <!-- defaults to article title when left blank
-->
 | native_name        = <!--The person's name in their own language, if
different.-->
 | native_name_lang   = <!--ISO 639-1 code, e.g., "fr" for French. If
more than one, use {{lang}} in |native_name= instead.-->
 | honorific_suffix   =
 | image              =
 | image_size         =
 | image_upright      =
 | smallimage         = <!--If this is specified, "image" should not
be.-->
 | smallimage_alt     =
 | alt                =
 | caption            =
 | order              =
 | office             =
 | status             = <!--If this is specified, overrides
Incumbent.-->
 | term_start         =
 | term_end           = <!-- Add data only when the actual term has
ended, not for terms which will end in the future. (Per usage
guideline.) -->
 | subterm            =
 | suboffice          =
 | alongside          = <!--For two or more people serving in the same
position from the same district.  (e.g. United States senators.)-->
 | monarch            =
 | president          =
 | governor_general   =
 | primeminister      =
 | taoiseach          =
 | chancellor         =
```

{{{honorific_prefix}}}
{{{name}}}
{{{honorific_suffix}}}
{{{native_name}}}
[[File:{{{image}}}|{{{image_size}}}|alt={{{alt}}}|upright={{{image_upright}}}]]
[[File:{{{image name}}}|{{{image_size}}}|alt={{{image_name_alt}}}|upright={{{image_upright}}}]]
[[File:{{{smallimage}}}|frameless|alt={{{smallimage_alt}}}|upright={{{image_upright}}}]]
{{{caption}}}
{{{order}}} [[{{{office}}}]]
{{{status}}}
In office
{{{term_start}}}–{{{term_end}}}
Serving with {{{alongside}}}

Monarch	{{{monarch}}}
President	{{{president}}}
Governor General	{{{governor_general}}}
Prime Minister	{{{primeminister}}}
Chancellor	{{{chancellor}}}
Taoiseach	{{{taoiseach}}}
Governor	{{{governor}}}
Vice President	{{{vicepresident}}}
[[Vice {{{office}}}|Vice PM]]	{{{viceprimeminister}}}
Deputy	{{{deputy}}}
[[Lieutenant {{{office}}}|Lieutenant]]	{{{lieutenant}}}
{{{subterm}}}	{{{suboffice}}}
Preceded by	{{{predecessor}}}

Figure B.2: Template of infobox officeholder



```
{{Infobox football biography
|embed            =
|name             =
|image            =
|upright          =
|alt              =
|caption          =
|full_name        =
|birth_date       =
|birth_place      =
|height           =
|position         =
|currentclub      =
|clubnumber       =
|youthyears1      =
|youthclubs1      =
|collegeyears1    =
|college1         =
|collegecaps1     =
|collegegoals1    =
|years1           =
|clubs1           =
|caps1            =
|goals1           =
|nationalyears1   =
|nationalteam1    =
|nationalcaps1    =
|nationalgoals1   =
|medaltemplates   =
|club-update      =
|nationalteam-update =
}}
```

[[File:{{{image}}}|frameless|alt={{{alt}}}|upright={{{upright}}}]]
{{{caption}}}

Personal information

Full name	{{{full_name}}}
Birth name	{{{birth_name}}}
Date of birth	{{{birth_date}}}
Place of birth	{{{birth_place}}}
Date of death	{{{death_date}}}
Place of death	{{{death_place}}}
Height	{{{height}}}
Position(s)	{{{position}}}

Team information

Current team	{{{currentclub}}}
Number	{{{clubnumber}}}

Youth career

{{{youthyears1}}}	{{{youthclubs1}}}
{{{youthyears2}}}	{{{youthclubs2}}}
{{{youthyears3}}}	{{{youthclubs3}}}
{{{youthyears4}}}	{{{youthclubs4}}}
{{{youthyears5}}}	{{{youthclubs5}}}
{{{youthyears6}}}	{{{youthclubs6}}}
{{{youthyears7}}}	{{{youthclubs7}}}
{{{youthyears8}}}	{{{youthclubs8}}}
{{{youthyears9}}}	{{{youthclubs9}}}
{{{youthyears10}}}	{{{youthclubs10}}}
{{{youthyears11}}}	{{{youthclubs11}}}
{{{youthyears12}}}	{{{youthclubs12}}}

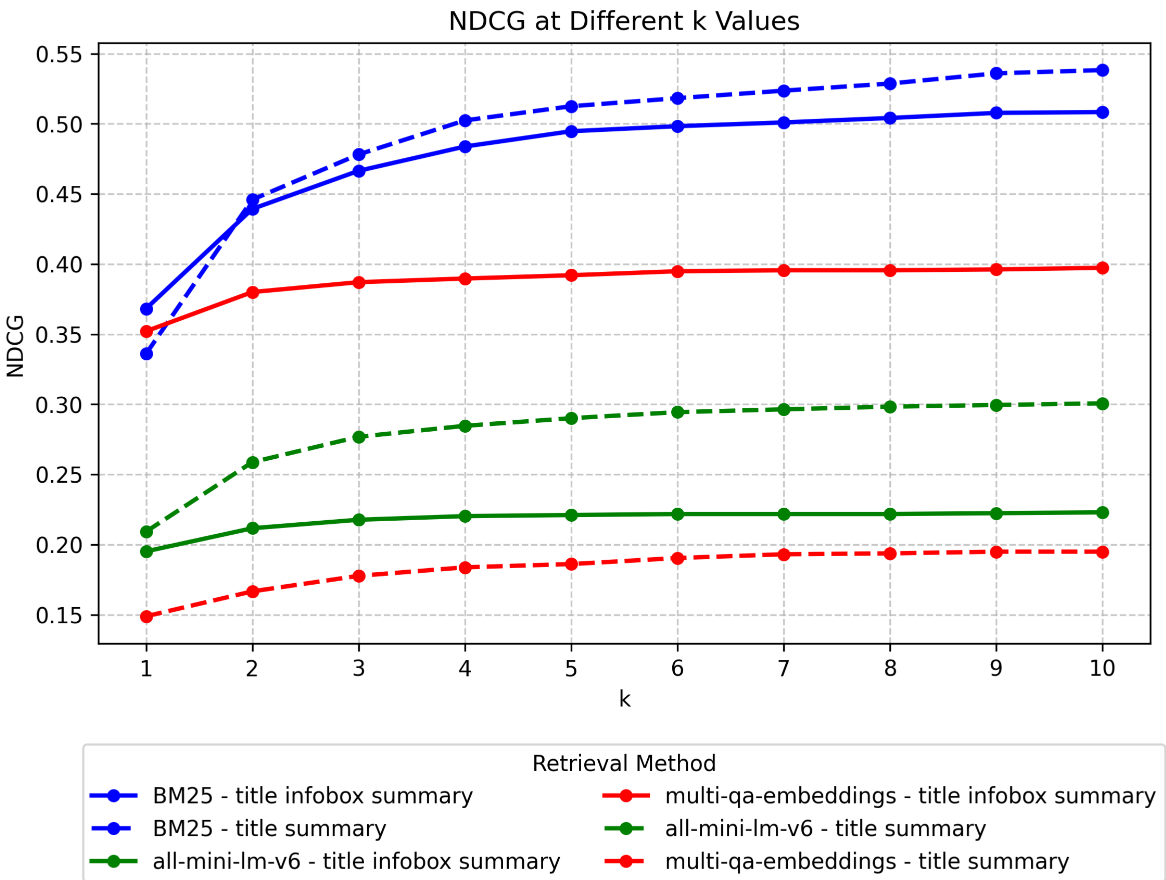
College career

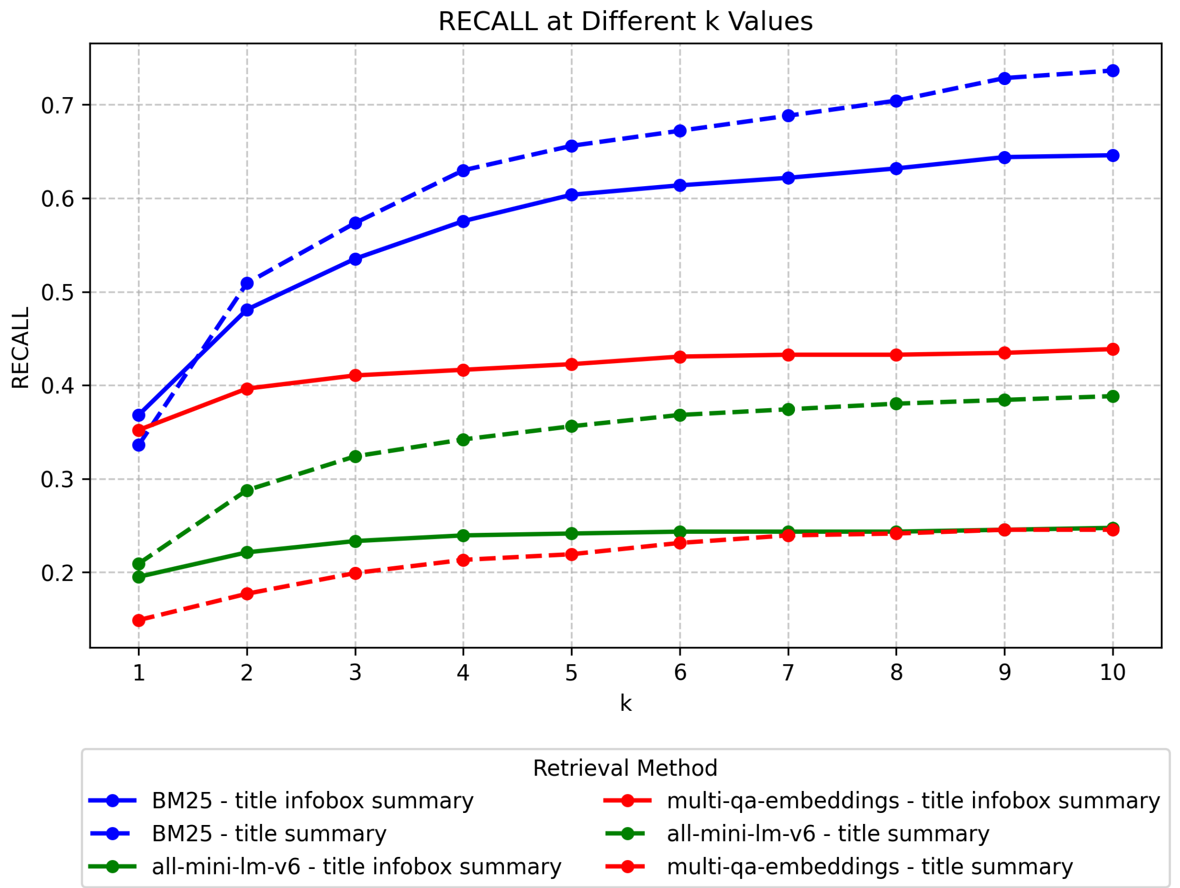
Years	Team	Apps	(Gls)

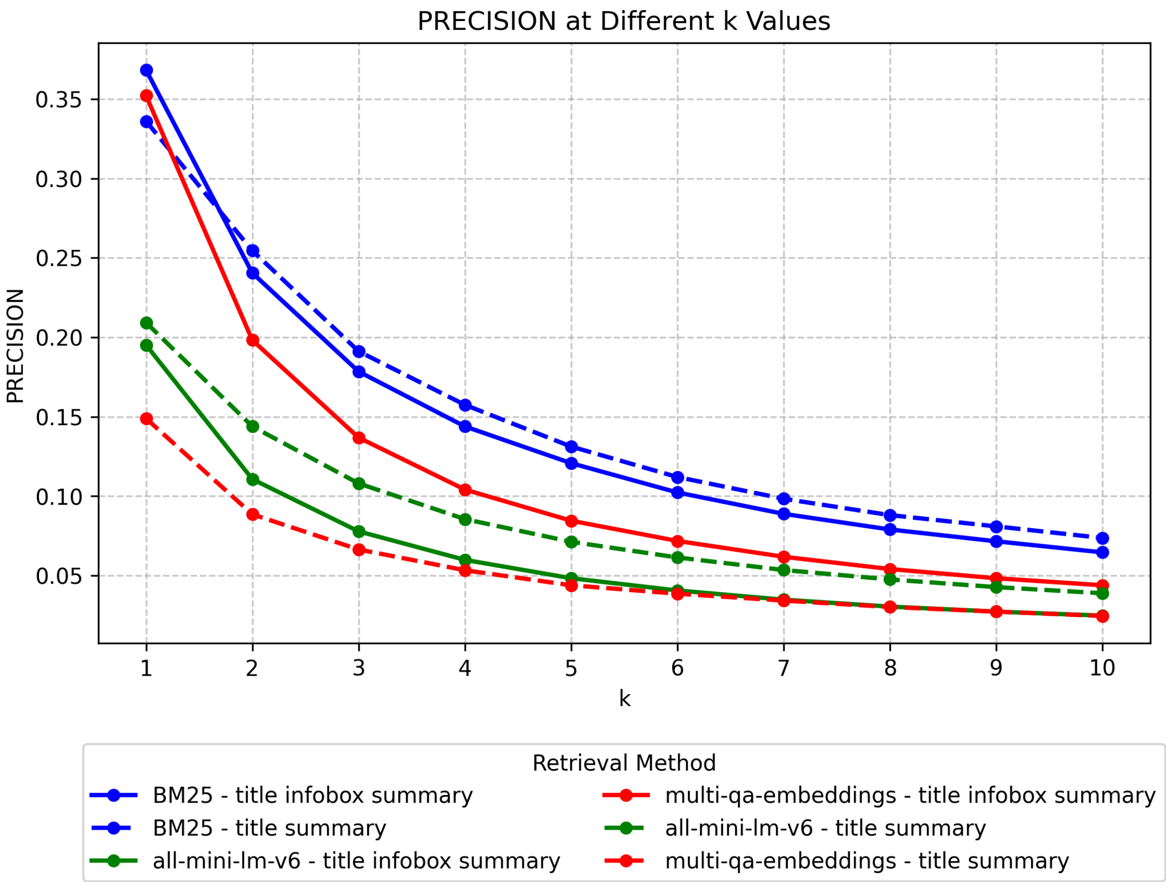
Figure B.3: Template of infobox officeholder footballplayer

C

Retriever Performance Plots




RECALL at Different k Values
RECALL
0.2
0.3
0.4
0.5
0.6
0.7
1
2
3
4
5
6
7
8
9
10
k
Retrieval Method
BM25 - title infobox summary
BM25 - title summary
all-mini-lm-v6 - title infobox summary
multi-qa-embeddings - title infobox summary
all-mini-lm-v6 - title summary
multi-qa-embeddings - title summary


PRECISION at Different k Values
PRECISION
0.35
0.30
0.25
0.20
0.15
0.10
0.05
1
2
3
4
5
6
7
8
9
10
k
Retrieval Method
BM25 - title infobox summary
BM25 - title summary
all-mini-lm-v6 - title infobox summary
multi-qa-embeddings - title infobox summary
all-mini-lm-v6 - title summary
multi-qa-embeddings - title summary