# HybridContextQA: A Hybrid Approach for Complex Question Answering using Knowledge Graph Construction and Context Retrieval with LLMs

Ghanshyam Verma[1], Simanta Sarkar[1], Devishree Pillai[1], Hotaka Shiokawa[2], Hamed Shahbazi[2], Fiona Veazey[2], Peter Hubbert[2], Hui Su[2] and Paul Buitelaar[1]

[1]*Insight SFI Research Centre for Data Analytics, Data Science Institute, University of Galway, Ireland*
[2]*Fidelity Investments, USA*

## Abstract

Augmenting domain-specific knowledge with Large Language Models (LLMs) to answer complex conditional questions is an important area of research. LLMs are good at answering general domain questions, however, their performance decreases when applied to a specific domain with complex conditional questions. We hypothesize that extracting context from relevant documents and Knowledge Graphs (KGs), and then feeding this combined knowledge to the LLM prompts, can provide better context to answer the complex conditional questions.

To test our hypothesis, we propose a hybrid approach called **Hybrid Context** for Complex **Q**uestion-**A**nswering (**HybridContextQA**) that can extract relevant context from documents as well as from a KG. To implement this, we create a Retrieval-Augmented Generation (RAG)-based hybrid context retrieval pipeline. This pipeline creates a KG from the provided documents and stores it in a Neo4j graph store. An LLM is used to automatically create a KG from the provided documents. The pipeline also stores the context extracted from the documents in vector form in a vector database. This combined context from KG and vector store can then be used for answering the complex conditional questions of that domain using an LLM.

We perform our experiments on a complex question-answering (QA) dataset called ConditionalQA. This dataset contains complex questions with conditional answers. We also compare the proposed approach with other approaches such as Code Prompt, Text Prompt, and Think-on-Graph. We find that the HybridContextQA approach performs better than the existing approaches for multiple LLMs, including Mistral and Mixtral.

We also conduct comprehensive experiments to analyze the contribution of the context from KG and vector form. We release the code implementing the HybridContextQA approach and the end-to-end pipeline with LLM prompts[1].

[1]https://github.com/GhanshyamVerma/ComplexQA

*KBC-LM'24: Knowledge Base Construction from Pre-trained Language Models workshop at ISWC 2024*

ghanshyam.verma@insight-centre.org (G. Verma); simon.simanta@insight-centre.org (S. Sarkar); devishree.pillai@insight-centre.org (D. Pillai); Hotaka.Shiokawa@fmr.com (H. Shiokawa); hamed.shahbazi@fmr.com (H. Shahbazi); fiona.veazey@fmr.com (F. Veazey); peter.hubbert@fmr.com (P. Hubbert); hui.su@fmr.com (H. Su); paul.buitelaar@universityofgalway.ie (P. Buitelaar)
https://www.universityofgalway.ie/ (G. Verma); https://www.universityofgalway.ie/ (S. Sarkar); https://www.universityofgalway.ie/ (D. Pillai); https://www.fidelity.com/ (H. Shiokawa); https://www.fidelity.com/ (H. Shahbazi); https://www.fidelityinvestments.ie/ (F. Veazey); https://www.fidelityinvestments.ie/ (P. Hubbert); https://www.fidelity.com/ (H. Su); https://www.universityofgalway.ie/ (P. Buitelaar)

# 1. Introduction

The integration of Large Language Models [1] with domain knowledge to answer complex conditional questions of that domain is a crucial area of research. While LLMs excel in answering general domain questions, their performance significantly decreases when applied to complex conditional questions of a particular domain. One of the reasons for their low performance on domain-specific QA tasks is that they have not been trained specifically for those domains [2]. In those cases, LLMs exhibit hallucinations: generation of seemingly plausible but nonsensical or unfaithful answers based on provided sources [3]. In the case of conditional questions, it also becomes hard to answer the question with the right condition if there is no proper context provided to the LLM. These limitations highlight the need for innovative approaches to enhance the performance of LLMs in answering complex conditional questions of a specific domain.

One promising direction is to leverage a domain-specific KG and augment it with an LLM. KGs are powerful tools for organizing and structuring knowledge [4, 5], and their integration with LLMs has shown promising results in various domains [1]. When building a KG from provided documents, incomplete extraction of important triples can negatively impact the performance of the LLM in answering complex questions. This is because the LLM may lack sufficient context or required conditions, particularly for conditional answers. Here, providing additional context in the form of vectors that are extracted from the same documents can complement the context.

We hypothesize that extracting the context in the KG and vector form, and then using that combined context to answer the complex conditional questions, can give better performance than the individual form of context. To test this hypothesis we propose the HybridContextQA approach that extracts the context from the provided documents in the form of a KG and vector and then feeds this combined context to an LLM to answer the complex conditional questions. To implement this we create an end-to-end pipeline that automatically constructs a KG using an LLM on provided documents and store this KG in a Neo4j graph store. The pipeline also extracts the context in vector form and stores it in a vector store. The combined knowledge store is then used to answer the complex questions.

To test this hypothesis, we create our own KG from the documents of the ConditionalQA dataset [6] using the KG creation module of the pipeline. The ConditionalQA dataset has been created using publicly available UK policy documents. We also extract the context in vector form from the same ConditionalQA documents. It is interesting to identify the contribution of the KG and vector context in answering the complex question. For that, we perform comprehensive experiments to analyze the contribution of the KG and vector context.

The main contributions of this paper are as follows:

1. We construct a pipeline for automatic KG creation that exploits the capabilities of LLMs for this task.

2. We propose a hybrid approach called HybridContextQA to extract the context in the form of a KG and vectors and then use this combined context to answer the complex conditional question.

3. We perform comparative analysis and provide insights on the effectiveness of the proposed HybridContextQA approach in comparison to existing approaches, particularly for a domain-specific complex conditional QA task.

4. We release our code that implements HybridContextQA in the form of an end-to-end

pipeline for KG creation, context extraction in vector form, and for performing the downstream task of QA using an LLM [1].

The rest of the paper is structured as follows. In Section 2, we describe related work. Section 3 describes the ConditionalQA dataset. In Section 4, we explain our approach. Section 5 describes the experimental design. In Section 6, we discuss and compare results in detail. Finally, we conclude in Section 7.

## 2. Related Work

LLMs have demonstrated significant performance in simple QA tasks, yet their effectiveness in complex, conditional, domain-specific QA tasks remains an area requiring further investigation. In this context, Sun et al. have introduced ConditionalQA, a dataset specifically designed for complex conditional questions, which serves as a benchmark for testing the capabilities of LLMs in handling such tasks [6].

Puerto et al. proposed a code prompting approach, that enhances LLMs' ability to tackle complex conditional questions by employing a chain of prompts that translate the natural language problem/question into code, which is then used to prompt the LLM [7]. While this code prompting method has demonstrated improved performance over traditional text approaches, it does not incorporate KGs for question answering.

The integration of KGs with LLMs has been actively researched, with multiple studies demonstrating that KGs can enhance the performance of LLMs across various domains, including the legal domain [1]. KGs provide structured representations that can complement the implicit knowledge encoded in LLMs, leading to more accurate and context-aware responses. Several notable approaches have been developed to improve inference by combining the strengths of KGs and LLMs, such as MindMap [8], Think-on-Graph [9], and Knowledge Solver [10] for instance.

MindMap specifically extracts information from a biomedical KG named EMCKG and incorporates this data into LLM prompts, aiming to enhance inference capabilities [8]. This approach was evaluated using a biomedical QA dataset called GenMedGPT-5k, and it has shown that integrating KG-derived information can indeed improve LLM performance.

Think-on-Graph utilizes LLMs as agents that iteratively perform beam searches on a KG to identify relevant reasoning paths, which are then used in LLM prompts to answer questions [9]. This approach has shown promising results, providing evidence in support of utilizing KGs in combination with LLMs to perform QA tasks.

However, these approaches have not yet been tested on QA datasets of a domain-specific complex conditional nature. On the other hand, Retrieval-Augmented Generation (RAG) is a promising approach that incorporates knowledge from external databases or documents with LLMs such that a QA system can be built [11]. However, most of the RAG pipelines are based on keyword and similarity-based searches which can limit the overall performance of the RAG systems [12].

[1] https://github.com/GhanshyamVerma/ComplexQA

These gaps highlight the need for further research to explore the potential of integrating KGs with LLMs using hybrid approaches that exploit both semantic and similarity-based retrievers for enhancing performance in complex, conditional, domain-specific QA tasks.

## 3. Dataset

ConditionalQA [6] is a QA dataset introduced by Sun et al. This dataset challenges existing QA models, by requiring compositional logical reasoning across multiple contexts having many complexly interrelating parts, elements, conditions, or considerations. It includes long context documents with logically complex information, multiple-hop questions, and various question types (extractive, yes/no, multiple answers, not-answerable) [6]. The ConditionalQA dataset aims to stimulate research in understanding complex documents to answer challenging questions.

The documents in ConditionalQA focus on UK public policies. Examples include topics like “Vaccine-Damage-Payment” or “Tax-on-Shopping”. Each document covers a specific policy topic and is organized into sections and subsections. The content within a section is closely related, but there may also be cross-references to other sections [6].

To answer the complex questions of the ConditionalQA data, we need to consider the following three main components: **Document:** It describes a UK public policy. The content is coherent and hierarchical. It is structured into sections and subsections. These policy documents are scraped from UK public policy websites. The content is then processed by serializing the Document Object Model (DOM) trees of web pages into lists of HTML elements (such as <h1>, <li>, <p>,and <tr>). **Question:** It asks about a specific aspect of UK public policy. The questions can be related to a person’s eligibility to apply for something or other aspects, using words like "who," "what," "how," "where," or "when." Even if a question is “not answerable,” it remains relevant to the document content. **User Scenario:** It provides background information or the real-word situation of a person for the question. The scenario may also be simulating real-world information-seeking challenges [6].

A QA model should predict answers with associated conditions, if any. The answers can fall into multiple categories: “Yes” or “no” responses to questions like, “Can I get this benefit?” Extracted text spans for questions asking “how,” “when,” “what,” etc., or “Not answerable” if no relevant answer exists in the document. Since complete information for a definite answer is sometimes lacking, the model must also recognize the conditions necessary for a correct answer. A condition represents additional information that must be satisfied but is not explicitly mentioned in the user’s query.

In the context of ConditionalQA, conditions are present in the content of the documents. A QA model should retrieve the correct answer from the provided documents, along with the conditions that must be met for the answer to be valid. See Figure 1 for an example of a QA pair from ConditionalQA dataset. Figure also shows the answer and condition with parts of the document that are important for answering the question. The model evaluates selected conditions from the document as a retrieval task, aiming for a perfect F1 score at the element level. If no conditions are required, the model should return an empty list.

This dataset challenges QA models by incorporating these conditional dependencies, making

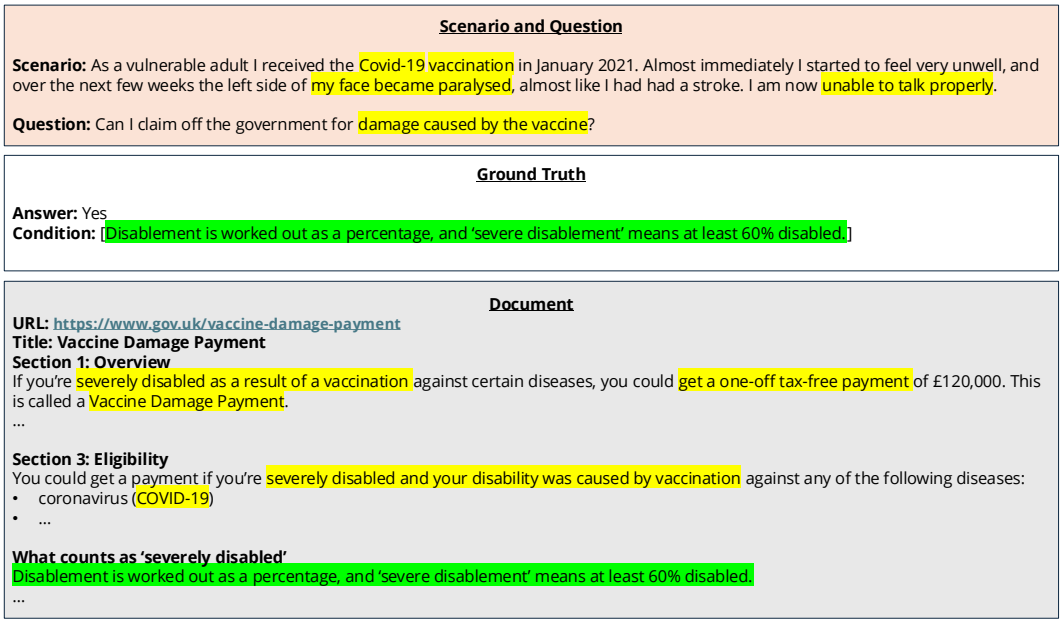


**Figure 1:** An example of a QA pair from the ConditionalQA dataset. The figure also highlights the answer, condition, and relevant parts of the document that are crucial for answering the question.

it more intricate than traditional QA datasets. We use ConditionalQA to explore how well our model handles complex reasoning and context-aware answers.

## 4. Proposed Approach

We propose the HybridContextQA approach that not only extracts the context from the KG, but also from the vector index. It then uses the combined context to answer the complex questions related to UK policies.

To implement the proposed approach, we create a RAG-based hybrid pipeline for KG creation, context extraction in vector form, and perform the complex QA task. The pipeline uses UK policy documents to construct the KG. The pipeline, shown in Figure 2, consists of five modules: loading, indexing, storing, querying, and evaluating. The pipeline modules are described as follows:

### 4.1. Loading

The Loading stage is the initial step in the RAG pipeline, where HTML text documents are ingested into the system as document objects. The SimpleDirectoryReader is employed to read these documents. Subsequently, the documents are divided into nodes based on HTML H1 and H2 tags, with each section loaded as a node. We use HTMLDocsReader for extracting relevant content from the HTML files, ensuring that each node contains meaningful and contextually relevant data.

## 4.2. Indexing

Indexing is a crucial stage where the data is structured to allow efficient querying. In this pipeline, we utilize two types of indexing: KG Indexing and Vector Indexing. The KG Indexing involves using an LLM and a graph store (Neo4j) [13] to create a KG. We use LLM prompts to extract KG triplets from the documents. The LLM extracts the KG triplets in the form of subject, predicate, and object based on the provided instructions and the few shot examples in the prompts. Please refer to Appendix A for the LLM prompts we used for the KG creation. The created KG is then stored in the Neo4j graph store. The KG captures relationships and entities within the data, making it easier to retrieve contextually relevant information. Vector Indexing employs embedding models to generate vector embeddings which are numerical representations of the data's meaning persisted in a vector store for efficient similarity-based retrieval. An embedding model is a machine learning model that transforms text into numerical vector representations. These vectors capture the semantic meaning of the text, enabling efficient similarity comparisons. In our system, we use the Hugging Face Embedding model, specifically the BAAI/bge-large-en-v1.5, to generate vector embeddings of the documents.

## 4.3. Storing

Once the data is indexed, it is essential to store the index and other metadata to avoid re-indexing. This stage ensures that the data structure is persistent and can be efficiently retrieved for future queries. The persistence of the Knowledge Graph Index and Vector Store Index is managed using the StorageContext class. Neo4j is used for the graph store, while the Facebook AI Similarity Search (FAISS) library [14] is used to implement the vector store. The FAISS Vector Store is used for storing vector embeddings, initializing a FAISS index, and persisting it to a storage directory if it does not already exist. Similarly, the Knowledge Graph Index is stored or loaded from a persisted directory, ensuring that the index is available for future queries without needing to be rebuilt. The FAISS not only stores the embeddings but also allows for efficient similarity-based retrieval during the querying phase. When a query is posed, the model also generates a vector embedding for the query and retrieves the most relevant text by comparing similarity scores.

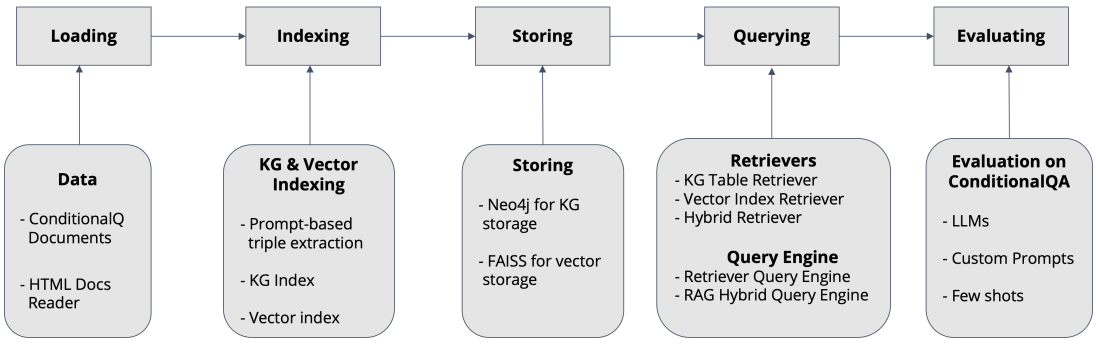


**Figure 2:** RAG-based HybridContextQA pipeline for KG creation, context extraction in vector form, and performing complex QA task using an LLM.

### 4.4. Querying

The Querying stage involves retrieving and processing relevant information based on user queries. This stage includes several components: Retrievers, Routers, Node Postprocessors, and Response Synthesizers. Retrievers fetch relevant data using different strategies, such as the VectorIndexRetriever for similarity-based retrieval and the KGTableRetriever for keyword-based retrieval from the knowledge graph. We have written a custom hybrid retriever that combines these two retrieval methods: the VectorIndexRetriever and the KGTableRetriever, both of which are part of the LlamaIndex library. This hybrid retriever handles both keyword-based searches (from the KG) and similarity-based searches (from the vector database). Routers select the appropriate retriever based on the specified index type. Node Postprocessors handle transformations or re-ranking of nodes before they are used in response synthesis.

The Response Synthesizer works by taking the user query and the retrieved text chunks and KG triples, and synthesizing them into a coherent answer. In our implementation, we use the simple_summarize configuration for response generation, which means all the retrieved text and triples are combined into a single LLM prompt for summarization. The Response Synthesizer generates the response from the LLM. It generates the final response based on the retrieved data and the user's query, with the BaseSynthesizer formatting the final response from the retrieved nodes.

### 4.5. Evaluating

In the Evaluation stage, the retrieved data is passed into the final LLM prompt to answer the complex question. ConditionalQA has various question types. Similar to the code prompting paper [7], we use different prompts to answer the various types of questions. We use three different prompts to answer three types of questions: Yes/No, span/extractive, and questions with conditions. These prompts then process the retrieved data extracted from the previous module and use an LLM to predict the answer. The answer is then matched with the ground truth and performance is measured using the F1 score. We use the exact match policy between ground truth and the predicted answer to compute the F1 score. We produce the results for each question type and also the average F1 score that shows overall results combining all the question types. Please refer to Appendix B for a detailed description of the evaluation criteria.

## 5. Experimental Design

We designed a set of experiments to compare the performance of HybridContextQA with existing approaches. We perform experiments to compare the proposed approach with existing approaches such as code prompt and text prompt [7]. Additionally, we perform experiments to compare the proposed approach with the Think-on-Graph approach [9], which is an existing approach that utilizes a KG to retrieve relevant context and then uses an LLM to answer the question.

We also compare HybridContextQA with the KG retrieval-based approach within the RAG pipeline. This is to evaluate how well HybridContextQA performs in comparison to the context retrieval approaches that solely depend on the KG retrieval method.

To perform all the experiments, we use ConditionalQA data, specifically, the development set provided by [6]. The training set has 2338 QA pairs and the development set has 285 QA pairs, along with the UK policy documents required to answer the questions. We use some of the QA pairs from the training set to perform few-shot learning. We also performed ablation experiments to assess the impact of the coverage of the KG on the performance. We use various LLMs for our experiments: GPT 3.5, GPT 4o, Mistral and Mixtral.

## 6. Results

Table 1 shows the results of Code Prompt, Text Prompt, Think-on-Graph, and the proposed HybridContextQA approach using various LLMs. The LLMs used for answer generation are GPT 3.5, GPT 4o, Mistral, and Mixtral. The results of Code Prompt and Text Prompt are taken from the code prompt paper [7]. To produce the results using the Think-on-Graph approach, we implemented the approach with the help of the provided code in the paper [9] and produced the results on the ConditionalQA dataset.

Based on the results, we can observe that the proposed HybridContextQA approach performs better than the rest of the approaches on three LLMs, which are GPT 4o, Mistral, and Mixtral (see Table 1). In the case of Mistral, the HybridContextQA approach achieved a 45.17% average F1 score, whereas Code Prompt, Text prompt, and Think-on-Graph achieved 28.26%, 28.84%, and 16.24% average F1 scores, respectively, which is 16.91%, 16.33%, and 28.93% lower than that of HybridContextQA, respectively. Similarly, in the case of Mixtral, the HybridContextQA approach achieved a 53.71% average F1 score, whereas Code Prompt, Text prompt, and Think-on-Graph achieved 40.88%, 46.60%, and 17.40% average F1 scores, respectively, which is 12.83%, 7.11%, and 36.31% lower than that of HybridContextQA, respectively. Moreover, the results of HybridContextQA using other LLMs are also comparable with the existing approaches. These results indicate that the use of combined context from a KG and the document, in the form of vectors, can enhance the performance of a complex QA task. The reason for this performance enhancement is that by using the combined context from the KG and the document, we can

**Table 1**
Results of Code Prompt, Text Prompt, Think-on-Graph and the proposed HybridContextQA approach on ConditionalQA dataset.

| **LLM/ Model used** | **Context** | **Approach** | **Avg F1 Score** [[All type answer][]] (271 QA pairs) | **F1 Score (Yes/No)** [[Yes/No answer][]] (106 QA pairs) | **F1 Score (Span)** [[Span answer][]] (102 QA pairs) | **F1 Score (Conditional)** [[All type answer][ Condition]] (63 QA pairs) |
|---|---|---|---|---|---|---|
| Mistral | Document | Code Prompt | 28.26 | 41.74 | 16.30 | 22.76 |
| | Document | Text Prompt | 28.84 | 31.58 | 25.64 | 23.51 |
| | KG | Think-on-Graph | 16.24 | 21.58 | 12.05 | 5.53 |
| | Document + KG | **HybridContextQA (ours)** | **45.17** | **60.00** | **33.55** | **34.39** |
| Mixtral | Document | Code Prompt | 40.88 | 44.80 | **40.99** | 34.62 |
| | Document | Text Prompt | 46.60 | 56.95 | 40.15 | 39.36 |
| | KG | Think-on-Graph | 17.40 | 23.14 | 12.90 | 5.54 |
| | Document + KG | **HybridContextQA (ours)** | **53.71** | **74.13** | 36.45 | **42.59** |
| GPT 3.5 | Document | Code Prompt | **57.64** | **78.64** | **44.40** | **48.02** |
| | Document | Text Prompt | 56.54 | 73.13 | 45.54 | 46.25 |
| | KG | Think-on-Graph | 16.29 | 13.97 | 20.67 | 9.48 |
| | Document + KG | **HybridContextQA (ours)** | 55.03 | 71.21 | 42.97 | 43.52 |
| GPT 4o | Document | Code Prompt | 63.63 | **81.82** | 50.28 | **54.72** |
| | Document | Text Prompt | 59.16 | 81.82 | 40.32 | 48.20 |
| | KG | Think-on-Graph | 20.40 | 21.45 | 21.46 | 7.85 |
| | Document + KG | **HybridContextQA (ours)** | **63.71** | 81.58 | **50.71** | 51.99 |

**Table 2**
Results of context retrieval in different forms using different prompt settings.

| Approach | KG / Hybrid | LLM/model used | Shots | Avg F1 score | F1 Score (Yes/No) | F1 Score (Span) | F1 Score (Conditional) |
|---|---|---|---|---|---|---|---|
| Prompt_v_1 max_triplets = 25 | KG | Mixtral | 0 | 37.57 | 64.73 | 7.14 | 37.5 |
| | Hybrid | Mixtral | 0 | 55.20 | 75.38 | 32.89 | 61.28 |
| | KG | Mixtral | 4 | 45.06 | 75 | 8.9 | 38.85 |
| | Hybrid | Mixtral | 4 | 60.72 | 87.5 | 30.08 | 66.55 |
| Default Prompt max_triplets = 25 | KG | Mixtral | 4 | 45.54 | 75 | 10.06 | 38.88 |
| | Hybrid | Mixtral | 4 | 61.13 | 81.55 | 38.99 | 73.53 |
| Default Prompt max_triplets = 30 | KG | Mixtral | 4 | 40.00 | 68.88 | 4.99 | 51.33 |
| | Hybrid | Mixtral | 4 | 57.35 | 87.5 | 21.9 | 70.01 |
| Prompt_v_2 max_triplets = 25 | KG | Mixtral | 4 | 53.94 | 75.23 | 30.06 | 52.5 |
| | Hybrid | Mixtral | 4 | **64.36** | 93.75 | 30.55 | 67.83 |

provide appropriate context to the LLM prompt to perform better reasoning over the provided context to generate the final answer.

When we use the context from both KG and the document in the form of vectors, a question arises as to how much contribution the KG context has and how much the vector context has in overall performance gain. To answer that question to some degree, we performed an ablation study on 10 documents from the dataset which comprises 30 QA pairs of the development set, with the results shown in Table 2. In this table, the approach column shows the version of the prompt and the number of max triplets used to extract the KG context and Hybrid index per chunk of the document. Each chunk of the document contains 256 tokens. The results show that when we increase the number of max triplets extracted, the performance increases at some level (max triplets =25) then it drops. This is because after a threshold, the information that we extract from the documents can include irrelevant information which can lead to a drop in performance. Therefore, it is important that while extracting the context from the documents, the irrelevant context should not be extracted otherwise it can affect the overall performance.

Table 2 also presents the results of context retrieval using the KG and the HybridContextQA approach within the proposed RAG-based Hybrid pipeline. For the ablation study, we tested three different prompts: Default prompt, Prompt version 1, and Prompt version 2. Please refer to Appendix A for the different prompts. The Default prompt has a parameter, max triplets, to set the number of triplets a user wants to extract. It also includes general in-context examples to implement few-shot learning. Prompt version 1 does not have the max triplets parameter; instead, it asks the LLM to extract all relevant triplets. Unlike the Default prompt, Prompt version 1 does not have general few-shot examples but includes few-shot examples from the ConditionalQA training data. Prompt version 2 has both the max triplets parameter and few-shot examples from the training data. We achieved the best results using Prompt version 2 with max triplets set to 25 (see Table 2). With this setting, KG-based context retrieval managed to get a 53.94% F1 score, whereas the HybridContextQA approach, which uses both KG and vector-based context, achieved 64.36%. The context extracted from the documents in vector form improved the performance by 10.42 percentage points. This demonstrates the contribution of context extracted from documents in vector form. Thus, the HybridContextQA approach is better than the individual context extraction approaches.

Overall, Think-on-Graph performed relatively lower than all the other approaches. This is because it is designed to use only the KG and lacks the necessary relevant context to answer

the complex conditional questions of the ConditionalQA dataset.

The proposed approach is scalable; however, its efficiency depends on the size of the KG and the vector database. For the ConditionalQA dataset, the system performs efficiently due to the use of both FAISS (for fast similarity search) and Neo4j (for graph-based retrieval).

## 7. Conclusion

In this paper, we contribute significantly to the field of complex conditional question answering by introducing the HybridContextQA approach that integrates LLMs with KGs for enhanced context retrieval. Our findings indicate that traditional LLMs struggle with complex conditional questions, particularly in specialized domains where precise contextual understanding is essential. By leveraging both KGs and vector representations of document contexts, we have demonstrated improvement in the performance of LLMs when tasked with answering complex conditional questions. The experimental results validate our hypothesis that a combined context from KGs and vector stores yields superior results compared to using either context type alone. Our approach was rigorously tested against various existing methods, including Code Prompt, Text Prompt, and Think-on-Graph, and showed consistent improvements across multiple LLMs, including Mistral and Mixtral. The comparative analysis not only highlights the effectiveness of our HybridContextQA approach but also provides insights into the specific contributions of KGs and vector contexts in the QA process.

In conclusion, our research underscores the potential of integrating different techniques to enhance the capabilities of LLMs in specialized domains. We believe that our HybridContextQA approach can serve as a foundational framework for future research aimed at improving QA in other complex domains. The open-source code and pipeline we have provided will facilitate further exploration and development in this area, encouraging other researchers to build upon our findings and refine the methodologies for knowledge extraction and context retrieval in legal and other domain-specific applications.

## 8. Acknowledgments

This publication has emanated from research supported in part by a grant from Science Foundation Ireland under Grant number SFI/12/RC/2289_P2 {Insight} and a grant from Fidelity Investments. For the purpose of Open Access, the authors have applied a CC BY public copyright license to any Author Accepted Manuscript version arising from this submission.

We would also like to thank Shyam Subramanian, Priyanka Sahoo, Ares (Xiaoming) Zhang, Ron(yourong) Xu, Venkatesh K, Stephanie Pearson, Alan McClean, Carmel McGroarty Mitchell, Melissa Nysewander, Ping Yao, and Nola Giltinan for attending project meetings and providing advice.

## References


[1] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, X. Wu, Unifying large language models and knowledge graphs: A roadmap, IEEE Transactions on Knowledge and Data Engineering (2024).

[2] N. Kandpal, H. Deng, A. Roberts, E. Wallace, C. Raffel, Large language models struggle to learn long-tail knowledge, in: Proceedings of the 40th International Conference on Machine Learning, ICML'23, JMLR.org, 2023.

[3] Y. Zhang, Y. Li, L. Cui, D. Cai, L. Liu, T. Fu, X. Huang, E. Zhao, Y. Zhang, Y. Chen, et al., Siren's song in the ai ocean: a survey on hallucination in large language models, arXiv preprint arXiv:2309.01219 (2023).

[4] M. Nickel, K. Murphy, V. Tresp, E. Gabrilovich, A review of relational machine learning for knowledge graphs, Proceedings of the IEEE 104 (2015) 11–33.

[5] Q. Wang, Z. Mao, B. Wang, L. Guo, Knowledge graph embedding: A survey of approaches and applications, IEEE Transactions on Knowledge and Data Engineering 29 (2017) 2724–2743.

[6] H. Sun, W. Cohen, R. Salakhutdinov, ConditionalQA: A complex reading comprehension dataset with conditional answers, in: S. Muresan, P. Nakov, A. Villavicencio (Eds.), Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Dublin, Ireland, 2022, pp. 3627–3637. URL: https://aclanthology.org/2022.acl-long.253. doi:10.18653/v1/2022.acl-long.253.

[7] H. Puerto, M. Tutek, S. Aditya, X. Zhu, I. Gurevych, Code prompting elicits conditional reasoning abilities in text+ code llms, arXiv preprint arXiv:2401.10065 (2024).

[8] Y. Wen, Z. Wang, J. Sun, Mindmap: Knowledge graph prompting sparks graph of thoughts in large language models, arXiv preprint arXiv:2308.09729 (2023).

[9] J. Sun, C. Xu, L. Tang, S. Wang, C. Lin, Y. Gong, H.-Y. Shum, J. Guo, Think-on-graph: Deep and responsible reasoning of large language model with knowledge graph, arXiv preprint arXiv:2307.07697 (2023).

[10] C. Feng, X. Zhang, Z. Fei, Knowledge solver: Teaching llms to search for domain knowledge from knowledge graphs, arXiv preprint arXiv:2309.03118 (2023).

[11] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, H. Wang, Retrieval-augmented generation for large language models: A survey, arXiv preprint arXiv:2312.10997 (2023).

[12] K. Sawarkar, A. Mangal, S. R. Solanki, Blended rag: Improving rag (retriever-augmented generation) accuracy with semantic search and hybrid query-based retrievers, arXiv preprint arXiv:2404.07220 (2024).

[13] A. Vukotic, N. Watt, T. Abedrabbo, D. Fox, J. Partner, Neo4j in action, volume 22, Manning Shelter Island, 2015.

[14] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, H. Jégou, The faiss library, arXiv preprint arXiv:2401.08281 (2024).

# A. Appendix A

## A.1. Prompt templates for KG creation

**Default Prompt for KG Triplet Extraction:**

"Some text is provided below. Given the text, extract up to "
"**{max_knowledge_triplets}** "
"knowledge triplets in the form of (subject, predicate, object). Avoid stopwords."
"———————"

"**Example:**"
"Text: Alice is Bob's mother."
"Triplets:(Alice, is mother of, Bob)"
"Text: Philz is a coffee shop founded in Berkeley in 1982."
"Triplets:"
"(Philz, is, coffee shop)"
"(Philz, founded in, Berkeley)"
"(Philz, founded in, 1982)"
"———————"

"Text: **{text}**"
"Triplets:"

**Prompt for KG Triplet Extraction V1:**

"You are a Knowledge Graph creation expert. Some text is provided below. Given the text, extract all the relevant knowledge graph triplets in the form of (subject, predicate, object). Avoid stopwords. which will contain relevant information to answer questions. "
"———————"

**"Example:"**
"Text: Apply by the overseas route if your acquired gender has been … "
"Triplets:(acquired gender, accepted in, approved country or territory)"
…
"Text: You'll get an 'interim certificate' if you or your spouse do not … "
"Triplets:"
"(you, get, interim certificate)"
"(spouse, fill in, statutory declaration)"
"(use, interim certificate, end marriage)"
"Text: You can stay married if you apply for a Gender Recognition Certificate. "
"Triplets:"(apply for, Gender Recognition Certificate, stay married)
"———————"

"Text: **text**"
"Triplets:"

**Prompt for KG Triplet Extraction V2:**

"Some text is provided below. Given the text, extract up to " "**{max_knowledge_triplets}** "
"knowledge triplets in the form of (subject, predicate, object). Avoid stopwords."
"————————"

**"Example:"**
"Text: Apply by the overseas route if your acquired gender has been … "
"Triplets:(acquired gender, accepted in, approved country or territory)"
…
"Text: You'll get an 'interim certificate' if you or your spouse do not … "
"Triplets:"
"(you, get, interim certificate)"
"(spouse, fill in, statutory declaration)"
"(use, interim certificate, end marriage)"
"Text: You can stay married if you apply for a Gender Recognition Certificate. "
"Triplets:"(apply for, Gender Recognition Certificate, stay married)
"————————"

"Text: **text**"
"Triplets:"

# B. Appendix B

## B.1. Evaluation

**Avg F1 score:** The Avg F1 score calculates the average F1 score for all questions of ConditionalQA development set except non-answerable questions. The ground truth and the predicted answer have a specific output format such as: [ [Answer] [ Condition] ]. The Avg F1 score is calculated by evaluating only the [Answer] part of the predicted answer output against the ground truth and not the [Condition] part of the output. There are 271 such QA pairs out of 285 total QA pairs.

**F1 score (Yes/No):** The F1 score (Yes/No) denotes the overall F1 score for the Yes/ No answer type questions of ConditionalQA development set. These may be questions with just Yes/No as answers, with or without any corresponding conditions. The Yes/No F1 score is the average F1 score which is evaluated using just the [Answer] part of the output and not the [Condition] part for each question. There are 143 such QA pairs out of 285 total QA pairs.

**F1 Score (Span/Extractive):** The F1 Score (Span) is the average F1 score for all extractive/span answer type questions of ConditionalQA development set. These questions have span/extractive type answers in their [Answer] part of their output and they may or may not contain any [Condition] in their output. While calculating the average F1 score, only the extractive part generated in the [Answer] part is considered and not the [Condition] part of the output. There are 128 such QA pairs out of 285 total QA pairs.

**F1 Score (Conditional)**: The F1 Score (Conditional) calculates the average F1 score for all questions of ConditionalQA development set which contains conditions in their output. These questions may have Yes/No or extractive answers in the [Answer] part of the output. The F1 Score (Conditional) is calculated by evaluating both the [Answer] part and the [Condition] part of the generated output against the ground truth. There are 63 such QA pairs out of 285 total QA pairs.