

Project Summary Report

COS300018 – Intelligent Systems

Project C – Team 5

Written by:

| | |
|-------------------|-----------|
| Alexander Small | 104007704 |
| Khanh Toan Nguyen | 104180605 |
| Matthew Crick | 104905950 |

Supervised by:

Mr. Hy Nguyen

Executive Summary

Large Language Models (LLMs) can generate incorrect responses due to inadequate training datasets [1, 2, 3], and struggle to answer questions requiring current world knowledge because their training data is fixed at one point in time [4]; it is expensive and time consuming to update LLM knowledge because this requires re-training the entire model. Our RAG solution addresses this issue by providing our LLM with an external knowledge base [5] which it can use to access factual and up-to-date information about the world. Our RAG can retrieve *context* relevant to a user's question from its knowledge base before answering the question, leading to more factual answers.

Our implementation employs a number of optimisations to improve performance, including dense retrieval of text embeddings [6, 7] for efficient semantic search [8], and Hierarchical Navigable Small Worlds (HNSW) graph [9] for optimised dense retrieval. One of the most impactful features was query decomposition [10, 11, 8]; the ability to break complex questions requiring many stages of reasoning into simpler questions which could each have context retrieved independently; this improved accuracy of complex queries by considering the context of each subquery to formulate a single coherent answer. This report outlines our RAG system's creation process, architecture, installation and usage guide, and limitations.

Table of Contents

| | |
|-----------------------------------------------------------------|-------------------------------------|
| Introduction..... | 3 |
| Project Requirements..... | 4 |
| LLM Requirements..... | 4 |
| Hardware Requirements | 4 |
| System Research..... | 4 |
| Hallucination Mitigation Technique | 4 |
| Prompt Engineering..... | 5 |
| Large Language Model | 6 |
| Parameter Size..... | 7 |
| Computational Consideration | 7 |
| Knowledge Base | 7 |
| System Workflow..... | 8 |
| System Architecture | 11 |
| Overview | 11 |
| Libraries Used..... | 12 |
| Models Used..... | 12 |
| Meta Llama 3.1..... | 12 |
| NoInstruct Small Embedding..... | 12 |
| System Installation Guide..... | 13 |
| Prerequisites..... | 13 |
| Installing GPU Drivers..... | 13 |
| Installing Miniconda (Python Virtual Environment Manager) | 14 |
| Creating a Virtual Environment | 14 |
| Installing Dependencies | 15 |
| Installing PyTorch with GPU..... | 15 |
| Requesting Access to Meta Llama 3.1 | 16 |
| Create a Hugging Face Account | 16 |
| Share Your Contact Information with Meta | 16 |
| Create a Hugging Face Token..... | 16 |
| Sign in to Hugging Face on Python | 17 |
| System Usage Guide | 19 |
| Deploying the RAG App Locally | 19 |
| Using the RAG Interface | 20 |
| Output Settings..... | 20 |
| RAG Usage and Configuration | 20 |
| Creating a Knowledge Base from Wikipedia | 23 |
| Demonstrations..... | Error! Bookmark not defined. |
| Evaluation | Error! Bookmark not defined. |
| HaluEval..... | Error! Bookmark not defined. |
| RAGAS..... | Error! Bookmark not defined. |
| Limitations of the RAG..... | 26 |
| Results | Error! Bookmark not defined. |
| Conclusion | 26 |
| References..... | 27 |

Introduction

Large Language Models (LLMs) present extraordinary natural language understanding capabilities; they are capable of text classification, text retrieval, question answering, and conversing in dialogues with human users [7]. Their ability to understand and reason with written language has opened up numerous applications for their use; LLMs have the potential to become personal assistants, provide therapy and emotional support and personalised tutoring [12], and have practical applications in fields such as healthcare and business due to their potential to perform tasks such as summarising medical records, generating financial analysis reports, providing legal advice and providing customer support [3].

However, a major obstacle to the widespread adoption of LLMs for these domains is the presence of LLM *hallucinations*: LLMs generating plausible but factually incorrect or fabricated information [3, 4]. LLMs have been proven to make considerable errors in real-world tasks such as generating medical summaries [13], which could have disastrous real-world consequences if they were put to use [3]. LLM hallucinations typically occur due to limitations with the data that was used to train the model (training data), limitations with the procedure the model was trained, and with the model's inference method [1]. LLM training datasets cannot contain world events that occur after the dataset was created [4], meaning an LLMs parametric knowledge – knowledge it gained during its training process [5] – will always be frozen in time at the point its training dataset was compiled. This means that conventional methods to train LLMs will always result in models which become outdated overtime and have no knowledge of current world events [4]. The transformer architecture [14] acts as the foundation for modern LLMs: it is used in the GPT and BERT architecture which are used for LLMs such as ChatGPT, Meta Llama, Microsoft Copilot, and Google's Gemini. As discussed by Xu, Jain, and Kankanhalli, the "...attention mechanism, a key component of transformer-based LLMs, can also contribute to hallucination, possibly because attention across tokens is diluted as the sequence length grows" [1]. Furthermore, the usage of cosine similarity to compare text embedding similarity when training LLMs also presents gradient vanishing issues which make it difficult for LLMs to make fine distinctions in meaning between different texts [15].

Numerous techniques have been developed to reduce LLM hallucinations; Tonmoy et al. [3] have created a taxonomy which delineates these techniques into two major categories: *prompt engineering* techniques and *fine-tuning* techniques (model development). The aim for our project was to implement *one* hallucination mitigation technique and implement it on an open-source LLM to investigate its efficacy. This report shall discuss our research, our process implementing the technique, our final system, and the effectiveness of our technique.

Project Requirements

No requirements specification was created specifically for the project by the team, however, the following system requirements were discovered and refined during the project.

The LLM hallucination mitigation system shall:

- R1.** be written in Python,
- R2.** implement *one* LLM hallucination mitigation technique without relying on existing libraries,
- R3.** use a *small* and *open-source* LLM that has under 10 billion parameters,
- R4.** have a user interface for the user to interact with the resulting model,
- R5.** be evaluated against benchmark evaluations / datasets, and
- R6.** be runnable on the team's computing hardware

LLM Requirements

Requirement 3 dictates that our chosen LLM could only have a maximum of 10 billion parameters. There were two reasons why this requirement was imposed:

1. LLMs with a lower number of parameters hallucinate more severely [16], therefore it was hypothesized for our project that the effects of a hallucination mitigation technique would be more visible and dramatic on a smaller LLM.
2. Our computing hardware was limited; we did not all have computers with enough disk space, CPU RAM and GPU VRAM to be able to run an LLM with over 10 billion parameters.

Hardware Requirements

Requirement 6 posed a number of restrictions as to what was possible with the project due to the team's limited computing hardware. The minimum hardware requirements for the system, which were modelled from the team's lowest-end computing hardware, are as follows:

- NVIDIA CUDA-compatible GPU [17].
- 16GB RAM.
- 6GB VRAM.

System Research

This chapter shall outline how our team selected hallucination mitigation technique and the components for the system architecture for our hallucination mitigation system. To develop a comprehensive RAG pipeline, our research efforts were divided into three key areas: retrieval, semantic chunking, and answer justification. Each area was assigned to a team member, with Toan focusing on embedding retrieval, Matthew on semantic chunking, and Alex on answer justification with multi-hop question. For the overall system architecture, we collaborated and worked on it together as a team.

Hallucination Mitigation Technique

Our team evaluated which hallucination mitigation technique was most appropriate for our project from the categories of *prompt engineering* and *model development* [3].

Prompt Engineering

The most notable prompt engineering technique to mitigate LLM hallucinations is *Retrieval Augmented Generation (RAG)* – a technique which allows LLMs to gather factual information related to a user’s query to inform a potential response by accessing external, authoritative knowledge bases [18, 3]. This type of hallucination mitigation technique externalises the knowledge of LLMs by storing it outside of the model parameters – this is known as extrinsic knowledge [5], and it has a number of advantages to parametric knowledge. This approach effectively mitigates hallucinations, enhances reliability, and ensures contextually accurate outputs, making it an ideal choice for developing robust and scalable hallucination mitigation systems.

TABLE I. Comparison of parametric and extrinsic memory in LLMs [19].

| | Parametric Memory (Internal) | Extrinsic Memory (External) |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | Knowledge stored <i>inside</i> a LLM which is acquired by pre-training an LLM on a corpus. | Knowledge stored <i>outside</i> of a LLM in a <i>knowledge base</i> which is acquired from external sources. |
| Storage Method | Stored as parameters within an LLMs hidden layers. | Stored as vectors within a vector database. |
| Advantages | <ul style="list-style-type: none"> May achieve better reasoning because the knowledge is intertwined with the reasoning process due to being stored within the model weights. | <ul style="list-style-type: none"> Can easily be updated to be made current. Can be verified for factual accuracy. Can allow for smaller LLMs to have factual accuracy without needing billions of parameters. |
| Disadvantages | <ul style="list-style-type: none"> Requires huge model parameter sizes (billions) to be effective in a variety of domains. Cannot be updated to reflect new knowledge without having to fully pre-train the LLM again, which is time-consuming and expensive. Cannot be verified for factual accuracy, because it is stored within hidden layers. | <ul style="list-style-type: none"> Cannot be used without a complex system architecture. |

RAG was chosen as the hallucination mitigation technique for the project. Other hallucination mitigation techniques, such as Fine-Tuning and Self-Refinement Through Feedback and Reasoning [3], were not chosen because RAG has the following advantages [19]:

- **Efficiency:**
RAG is the most computationally efficient technique because the LLM itself does not need to be modified re-trained.
- **Modifiability:**
RAG provides the easiest method to update model knowledge to reflect current events because the model’s knowledge is *extrinsic* (see Table II).
- **Ease of Implementation:**
RAG is the most widely accepted LLM hallucination mitigation approach within the broader AI community, meaning there is a wider pool of literature and resources available to facilitate its implementation.

Furthermore, there are several subcategories of prompt engineering techniques depending on where retrieval is performed in the text generation process: these are *before generation*, *during generation*, *after generation*, and *end-to-end* [3] – each offer unique benefits and trade-offs in improving the performance and reliability of LLMs.

TABLE II. Comparison of LLM prompt engineering subcategories.

| Model | Before generation | During generation | After generation | End-to-End |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Description</i> | <ul style="list-style-type: none"> Retrieves relevant information from external sources before generating a response | <ul style="list-style-type: none"> Retrieval occurs as the LLM generates each sentence. | <ul style="list-style-type: none"> Retrieval process occurs after the entire response has been generated. | <ul style="list-style-type: none"> Pre-trained sequence-to-sequence (seq2seq) model with a dense vector index (e.g., of Wikipedia) accessed through a Dense Passage Retriever (DPR) |
| <i>Advantages</i> | <ul style="list-style-type: none"> Access to up-to-date or context-specific knowledge right from the start | <ul style="list-style-type: none"> Allowing access specific data points for each part of its response Minimizes the likelihood of generating hallucinations since the model continually verifies its statements | <ul style="list-style-type: none"> Allow the model to produce a coherent response first, then verify its accuracy Useful for correcting errors | <ul style="list-style-type: none"> Combines retrieval and generation in a more seamless, cohesive manner, with rich context. |
| <i>Disadvantages</i> | <ul style="list-style-type: none"> Lack flexibility if further context or data is needed during response generation. | <ul style="list-style-type: none"> Computationally intensive Latency as it requires frequent calls to the retrieval system | <ul style="list-style-type: none"> It doesn't prevent the model from generating initial inaccuracies Relies on subsequent editing to correct hallucinations, which might not be foolproof. | <ul style="list-style-type: none"> Requires significant setup, including indexing large datasets, which might be resource intensive. |

To reduce hallucinations in LLMs, in-generation and end-to-end rag approaches are particularly effective in addressing these issues. The pre-generation approach provides context to the LLM before it generates a response, but it does not include any mechanisms to correct hallucinations as they arise. On the other hand, post-generation allows the LLM to generate a response first and then attempts to verify its accuracy, though this method doesn't guarantee that hallucinations will be fully resolved. In contrast, in-generation techniques involve continuous verification throughout the response generation process, enabling the model to check for accuracy from start to finish. Furthermore, the end-to-end rag approach enhances context by retrieving relevant information through Dense Retrieval (DR) and Hierarchical Navigable Small Worlds (HNSW), which indexes a knowledge base with dense vectors of Parse Retrieval (PR) with word match. Therefore, this streamlined process reduces steps compared to in-generation methods, resulting in faster and more accurate responses. Its efficiency and reliability led our group to select end-to-end RAG as the preferred approach.

Large Language Model

The team carefully considered which LLM to use for the RAG system. For an LLM to be suitable for our project, it had to:

- have under 10 billion parameters,
- be runnable on 6 gigabytes of VRAM or less, and
- rank highly on the Open LLM Leaderboard [20], a benchmark evaluating general LLM performance.

Only open-source LLMs from the top of the Open LLM Leaderboard [20] with less than 10B parameters have been considered for use in our RAG system. Of these LLMs, internlm2_5-7b-chat [21] and Meta-Llama-3.1-8B-Instruct [22] were selected as potential candidates [23].

| Model | Average | IFEval | BBH | MATH Lv1 5 | GPQA | MUSR | MMLU-PRO |
|---------------------------------------------------------------|---------|--------|-------|------------|-------|-------|----------|
| intxnlm/intxnlm2.5-7b-chat | 30.46 | 61.4 | 57.67 | 8.31 | 10.63 | 14.35 | 30.42 |
| microsoft/Phi-3-small-8k-instruct | 29.64 | 64.97 | 46.21 | 2.64 | 8.28 | 16.77 | 30.96 |
| google/gemma-2-9b-it | 28.86 | 74.36 | 42.14 | 0.23 | 14.77 | 9.74 | 31.95 |
| microsoft/Phi-3-small-128k-instruct | 28.59 | 63.68 | 45.63 | 0 | 8.95 | 14.5 | 30.78 |
| meta-llama/Meta-Llama-3.1-8B-Instruct | 27.91 | 78.56 | 29.89 | 17.6 | 2.35 | 8.41 | 30.68 |
| vicgalle/Configurable-Llama-3.1-8B-Instruct | 27.77 | 83.12 | 29.66 | 15.06 | 3.24 | 5.93 | 28.8 |
| 01-ai/Yi-1.5-9B-Chat | 27.71 | 60.46 | 36.95 | 11.63 | 11.3 | 12.84 | 33.06 |
| jpacifico/Chocolatine-3B-Instruct-DPO-Revised | 27.63 | 56.23 | 37.16 | 14.5 | 9.62 | 15.1 | 33.21 |

Fig. 1. Top performing models on the Open LLM Leaderboard [15] with under 10B parameters as of 12/09/2024.

Parameter Size

In data-to-text (D2T) tasks, although larger LLMs, such as OPT, BLOOM, and Llama 2, have vast numbers of parameters, their performance in (D2T) tasks, compared to smaller LLMs like BART and T5, depends on multiple factors beyond just model size. According to the research [source], the author considered the following aspects, readability, informativeness and faithfulness:

Firstly, for **readability**, the author suggests that larger LLMs tend to produce more readable outputs (evaluated based on fluency and coherence) than smaller LLMs. This advantage is evident in higher BLEU and METEOR scores among larger models from the same architecture-based language model family, indicating superior text quality.

Alongside readability, larger LLMs also generally excel in **informativeness** (the ability to convey useful information) compared to smaller models. Metrics like BertScore and MoverScore reveal that information content improves with model size, showcasing the capability of larger LLMs to deliver richer information.

In contrast to readability and informativeness, larger LLMs may struggle with **faithfulness** (accuracy of generated content relative to source data). As shown in Table [LLM\(1\)](#), increasing the parameters of models from families like BLOOM, OPT, and Llama 2 often results in reduced faithfulness across all D2T datasets.

Computational Consideration

LLMs are often fine-tuned to perform well on specific tasks, such as generating text from structured data (D2T). While larger LLMs hold potential for better readability and informativeness, therefore, this process often requires significant computational resources, which contributes to the overall cost compared to smaller models. This added expense can be a limiting factor when choosing between LLMs of different sizes.

Knowledge Base

The corpus is an important aspect of the retrieval process, it is formed as a data structure to retrieve relevant chunks of contextual information when formulating a response to a query.

We leveraged Wikimedia APIs to provide access to the latest Wikipedia dump files and used WikiExtractor to extract the raw text. The text is split into chunks to maintain meaning of the overall text and scoped on a paragraph level, finally these paragraph chunks and their first-lined summaries are converted using the Noinstruct emedding model into a 384-dimensional vector embedding.

Retrieval

A RAG system can effectively retrieve relevant context by leveraging advanced retrieval techniques based on high-dimensional vector embeddings. The core methods our team employed include **Dense Retrieval**, **Sparse Retrieval**, and **Hierarchical Navigable Small World (HNSW)** for approximate nearest neighbor searches, which is crucial for enhancing the performance of the generative model with efficient, accurate, and scalable context retrieval.

Sparse Retrieval

Sparse Retrieval relies on traditional term-matching techniques such as TF-IDF and BM25. BM25 (Best Matching 25) is a popular sparse retrieval algorithm that ranks documents based on the presence and frequency of search terms, while also considering the document's length to provide relevant results. However, BM25 lacks the ability to understand semantic meaning or context, relying solely on exact word matches. This limitation can lead to irrelevant or misleading results, especially if synonyms or paraphrased terms are not present in the documents. Additionally, the algorithm's reliance on document content means that even factually incorrect documents might be retrieved if they contain matching keywords. While BM25 incorporates length normalization to mitigate biases against longer documents, its accuracy can decline with excessively long texts, as these may be penalized incorrectly.

Dense Retrieval

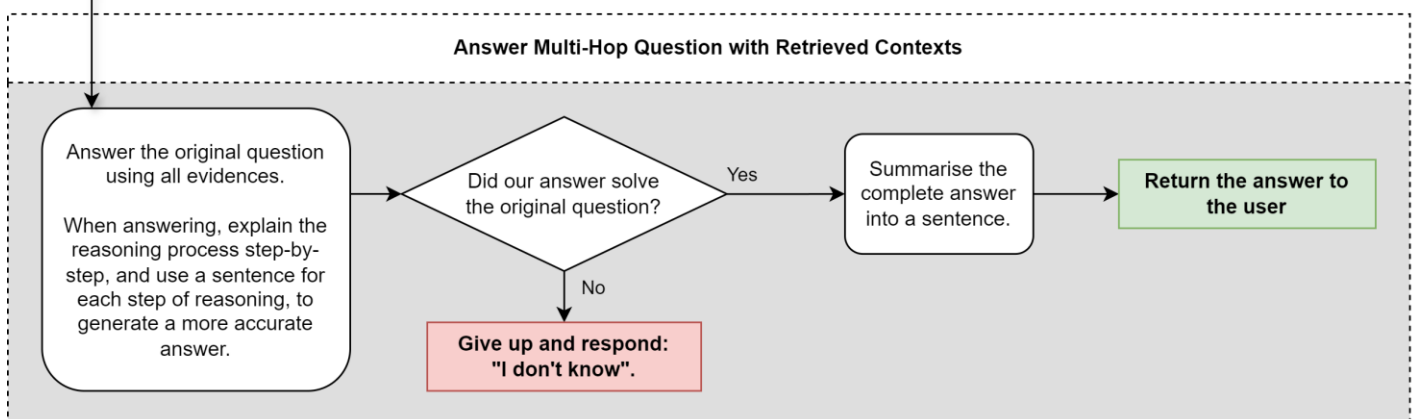
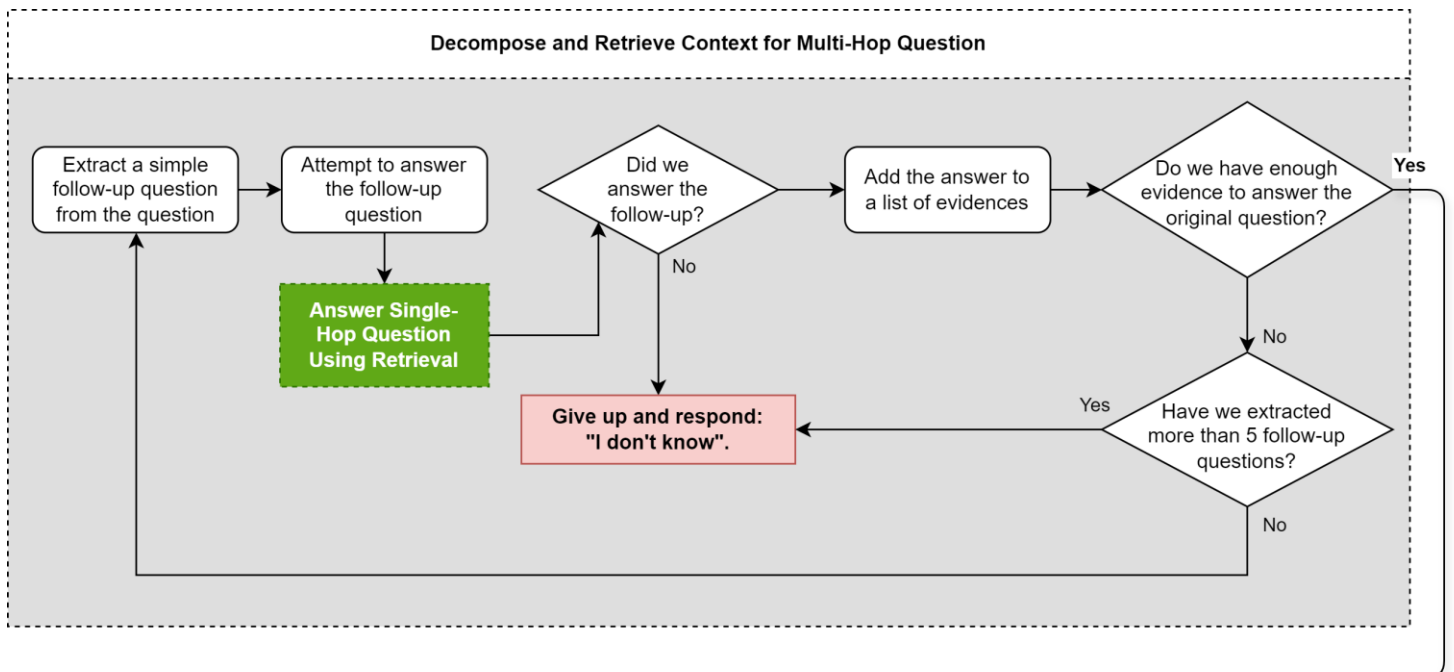
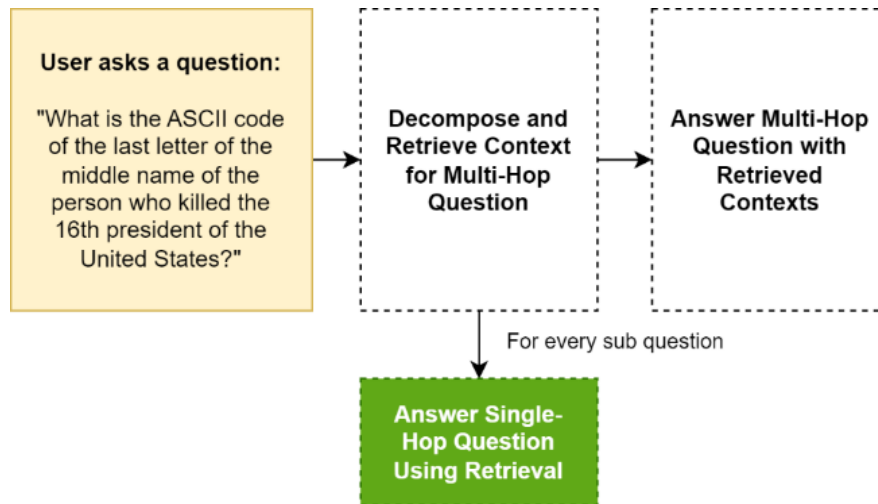
Dense Retrieval uses cosine similarity search to retrieve dense vector embeddings for both queries and documents. Cosine similarity calculates the angle between the query vector and document vectors, identifying the closest matches based on their semantic alignment. This allows dense retrieval to effectively understand context, synonyms, and nuanced relationships, however, because of comparing all the embedding vectors making it particularly raise computational cost and time consuming for natural language queries and complex search needs.

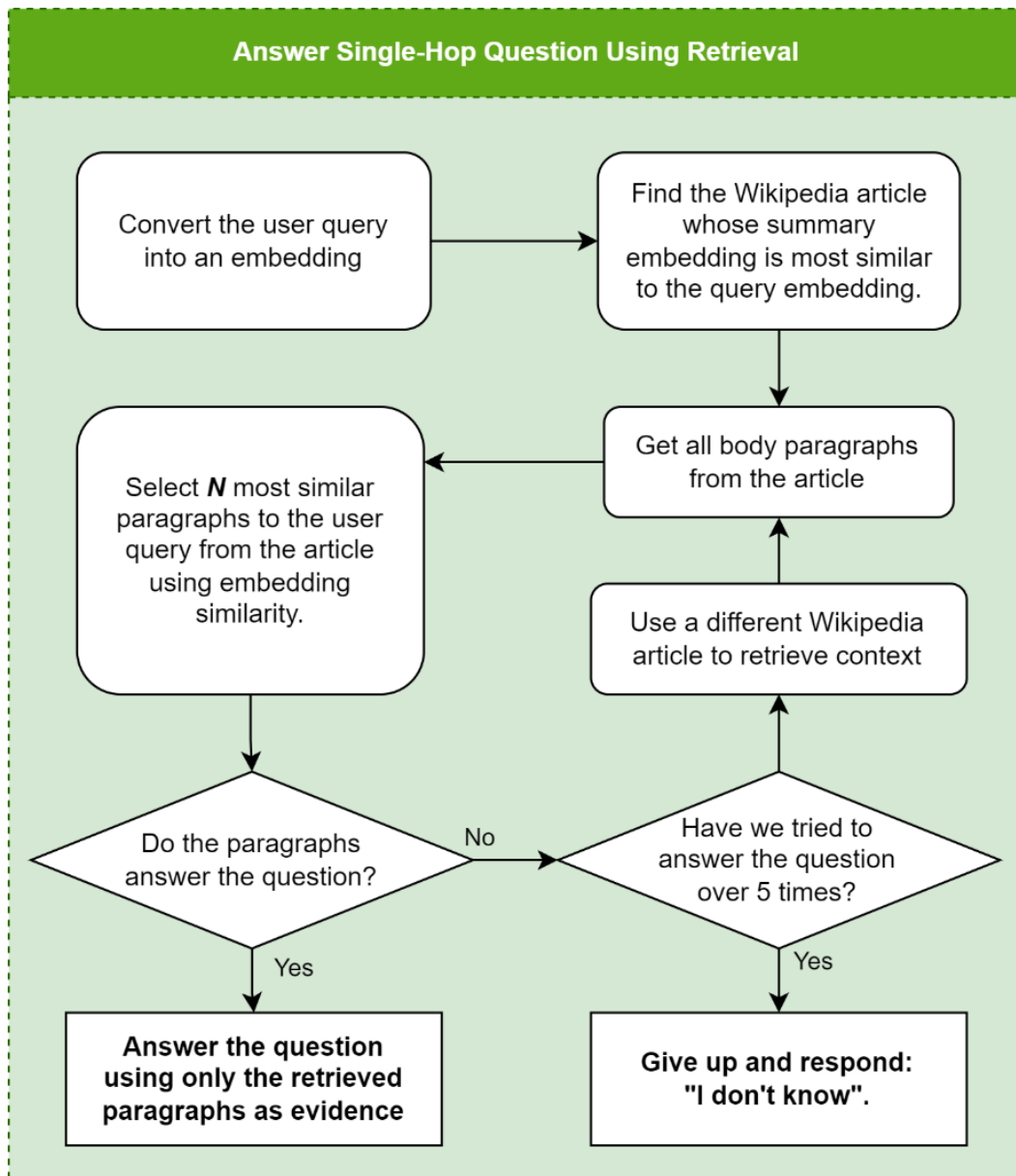
HNSW Retrieval

HNSW is an approximate nearest neighbor (ANN) search algorithm designed for scalable and efficient similarity searches in high-dimensional spaces. It is widely used in vector systems with multi layers search to improve retrieval speed and reduce computational overhead, making it one of the most efficient algorithms for vector-based systems.

System Workflow

Our RAG system workflow be explained at a high level using the following diagrams [24]:





System Architecture

This chapter shall overview the system architecture for our RAG system, including external libraries and software packages used.

Overview

This UML diagram shows the entirety of the RAG system architecture.

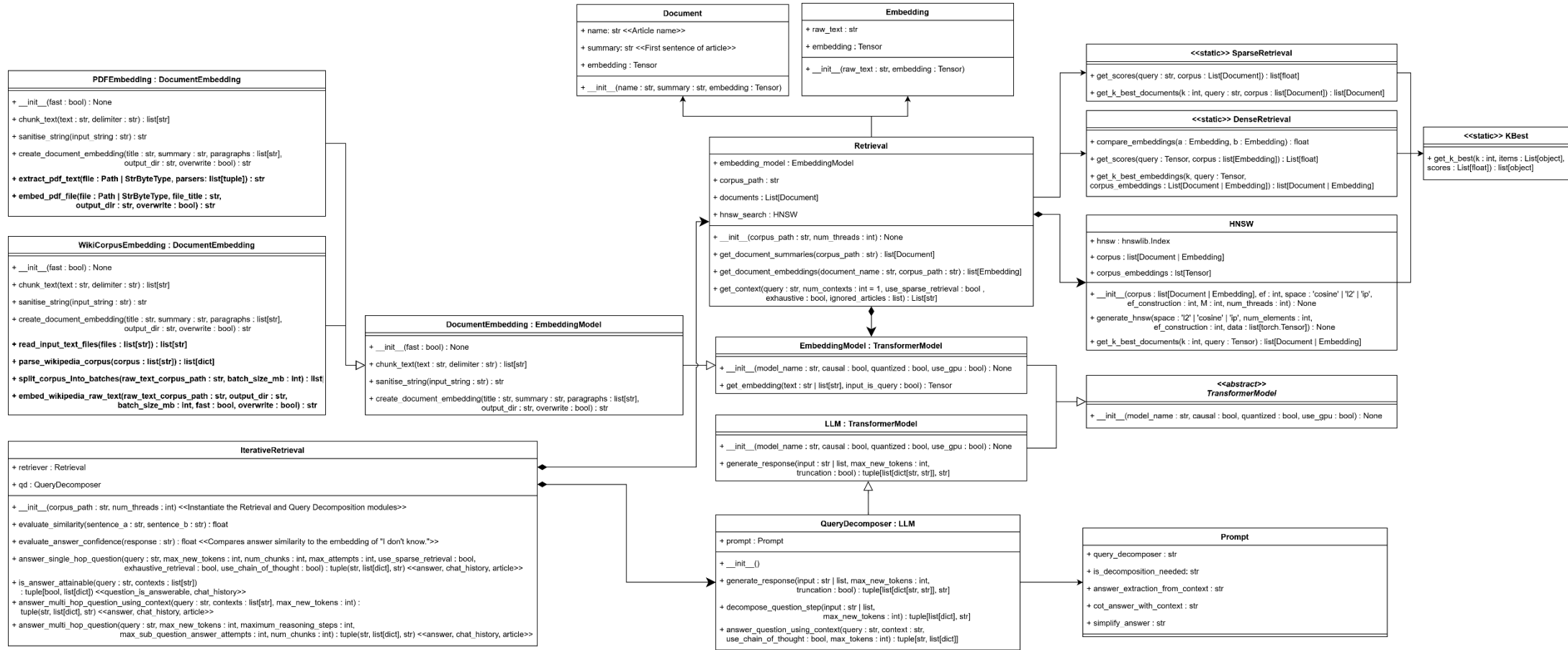


Fig. 2. UML Diagram of the RAG iterative retrieval module and its dependencies.

Libraries Used

Our solution utilised the following libraries:

- PyTorch
- transformers (Hugging Face) [25]
- hnswlib [9]
- WikiExtractor [26]
- Gradio [27]

PyTorch was our deep learning framework of choice for the project, but we did not directly use it for the construction, training, or loading of neural networks. Instead, we used the Hugging Face *transformers* library [25], which provides simpler abstractions for easy loading and usage of existing deep learning models directly from Hugging Face. The hnswLib and WikiExtractor libraries were both used for the retrieval process for our RAG system. Gradio was used as the user interface for our RAG system because it was easy to use and could be shared online.

Models Used

Our RAG system used two existing deep learning models from Hugging Face:

- | | |
|--------------------------|------------------------------------|
| • LLM | Meta Llama 3.1 8B Instruct [22] |
| • Embedding Model | NoInstruct small Embedding v0 [28] |

Meta Llama 3.1

Meta Llama 3.1 8B Instruct was selected as the chosen LLM for this project [23] because it:

- Is open source.
- Ranks highly on the Open LLM Leaderboard [20].
- Has under 10B parameters.
- Is convenient to set-up using libraries like Hugging Face's transformers library [25] or Ollama.

Unfortunately, Meta Llama 3.1 required 6GiB of VRAM to run, which narrowly exceeded our maximum VRAM of 6GB, making it impossible to deploy on our computing hardware even using the smallest model size of 8 billion parameters. To solve this issue, we employed 4-bit model quantization [29], a technique which halves the precision of a model's weights in order to halve the amount of RAM needed to deploy the model [30] This made it possible for us to run Llama 3.1 8B on our computers, however it also reduced the quality of the LLMs responses.

NoInstruct Small Embedding

NoInstruct small Embedding v0 [28] was used as our chosen embedding model for the RAG system because it has several advantages [23] over other universal text embedding models [6]. The model:

- Ranks the highest on the Massive Text Embedding Benchmark leaderboard [31] for models with under 100M parameters (as of 12/09/2024).
- Uses a data-based architecture.
- Uses triplet loss.
- Uses an optimised version of the GISTEmbed architecture [32] which leverages a novel method for unsupervised triplet mining to improve in-batch negative sample selection to reduce model biases and noise.

System Installation Guide

This chapter shall explain how you can install all the dependencies needed to run our RAG system locally on your computer.

Prerequisites

The following prerequisites are needed to install our system [33].

- **Windows or Linux machine**
 - *MacOS may work but we have not tested it.*
- **A NVIDIA GPU with at least 6GB RAM**
 - *The RAG works best with GPU acceleration, otherwise it is very slow.*
- **24 hours of time**
 - *There is an approval process to gain access to Meta Llama 3.1, our LLM.*

Installing GPU Drivers

Before you can start using the RAG system, you must first install the **NVIDIA CUDA Toolkit** so that the RAG may run on your computer's GPU (if you have one) to massively accelerate response time [33]. *If you do not have a GPU on your computer, you may skip this chapter.*

To install NVIDIA CUDA on your system, first verify your GPU meets the system requirements using the CUDA installation guide link provided:

| | |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Windows | https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html#system-requirements |
| Linux | https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#system-requirements |

If your GPU is not CUDA compatible, you may skip this chapter.

Next, download and install NVIDIA CUDA Toolkit 12.4 using this link:

<https://developer.nvidia.com/cuda-12-4-0-download-archive>

Once the NVIDIA CUDA Toolkit has installed, verify the installation has completed with the following command:

```
nvcc -V
```

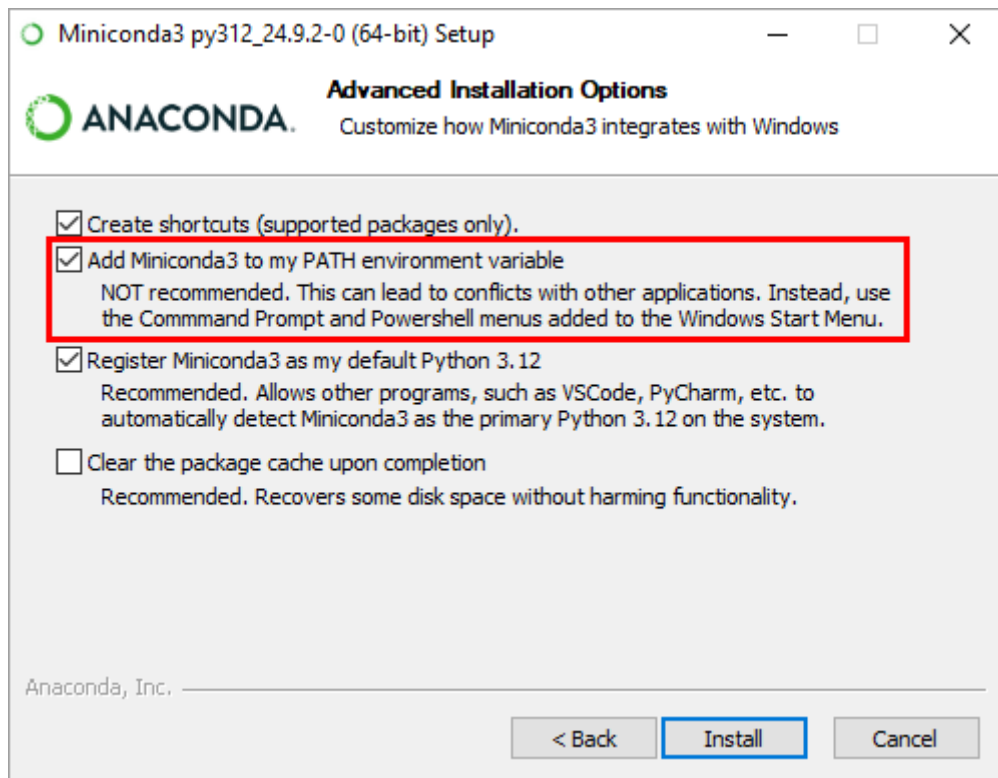
This should return:

```
> nvcc: NVIDIA (R) Cuda compiler driver
> Copyright (c) 2005-2024 NVIDIA Corporation
> Built on Thu_Mar_28_02:30:10_Pacific_Daylight_Time_2024
> Cuda compilation tools, release 12.4, V12.4.131
> Build cuda_12.4.r12.4/compiler.34097967_0
```

Installing Miniconda (Python Virtual Environment Manager)

Install Miniconda from: <https://docs.anaconda.com/miniconda/>. This will install **conda**, a Python virtual environment manager, onto your computer. We will use **conda** to create a Python virtual environment to run the RAG system locally [33].

When installing Miniconda, **ensure** that you check the option to “Add Miniconda to my PATH environment variable”. If this is not selected, you will **not** be able to use the **conda** package manager from the terminal.



Verify the installation was successful by opening the terminal (or use the application *Anaconda Prompt* if you did not add Miniconda to your PATH variable) and typing:

```
conda -V
> conda 24.5.0
```

A successful **conda** installation will show you which version of conda is installed.

Creating a Virtual Environment

Once **conda** is installed, create a new Python virtual environment for the project using the command:

```
conda create -n rag_team5 python=3.11
```

This command will create a Python virtual environment with the name **rag_team5** using Python 3.11. This will also install Python 3.11 and Python’s package manager **pip** onto your system.

Installing Dependencies

To install the RAG dependencies, you will first need to open a terminal in the project repository directory [33].

```
cd <YOUR PROJECT DIRECTORY>/ProjectCSurvival
```

To install the dependencies, execute the commands:

```
conda activate rag_team5
pip install -r requirements.txt
```

This will activate the `rag_team5` virtual environment and install the required packages onto the environment. Do not close this terminal window yet, as we will need it for the next step.

Installing PyTorch with GPU

If you installed **NVIDIA CUDA Toolkit** onto your system, you will also need to install a special version of PyTorch which supports it [33]. To do this, go to the PyTorch local install page at:

<https://pytorch.org/get-started/locally/>.

On the installation page, set the *Computer Platform* field to **CUDA 12.4** and the *Package* field to **Pip**.

| | | | | |
|-------------------|----------------------------------------------------------------------------------------------|-----------|-------------------|----------|
| PyTorch Build | Stable (2.4.1) | | Preview (Nightly) | |
| Your OS | Linux | Mac | Windows | |
| Package | Conda | Pip | LibTorch | Source |
| Language | Python | | C++ / Java | |
| Compute Platform | CUDA 11.8 | CUDA 12.1 | CUDA 12.4 | ROCm 6.1 |
| Run this Command: | pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu124 | | | |

Copy the resulting command, paste it into the terminal window, and execute it. This will install a GPU-compatible PyTorch on your system.

Requesting Access to Meta Llama 3.1

Meta Llama 3.1 is not publicly available for use – you have to request access from Meta before you can run it locally. This chapter will give detailed instructions on how to obtain access and install Meta Llama to run the RAG solution locally.

Create a Hugging Face Account

If you have not already, create an account on *Hugging Face* here: <https://huggingface.co/join>

Share Your Contact Information with Meta

Meta requires that all users of their LLM, *Meta Llama 3.1 8B instruct*, must provide them with their personal information. This chapter will overview how you can provide Meta with your personal information to access Llama 3.1 for use with the project. Access the Meta Llama 3.1 repository on Hugging Face here: <https://huggingface.co/meta-llama/Llama-3.1-8B>

By agreeing you accept to share your contact information (email and username) with the repository authors.

First Name

Last Name

Date of birth



Country



Affiliation

Job title



Your country and region (based on approximate Internet address) will be shared with the model owner.



By clicking Submit below I accept the terms of the license and acknowledge that the information I provide will be collected stored processed and shared in accordance with the Meta Privacy Policy

Submit

You will need to read the terms and conditions and fill out a personal information form to request access to the model. You will be notified by email once you have been granted access to the LLM. The authorisation process typically takes less than 24 hours.

Create a Hugging Face Token

To access Llama 3.1 on Python, you will need to connect your **conda** environment `rag_team5` to your Hugging Face account. To accomplish this, you will need to generate an **access token** to your Hugging Face account. Go to <https://huggingface.co/settings/tokens> and select “Create new token”.

Access Tokens

User Access Tokens + Create new token

Access tokens authenticate your identity to the Hugging Face Hub and allow applications to perform actions based on token permissions.
 ⚠ Do not share your Access Tokens with anyone; we regularly check for leaked Access Tokens and remove them immediately.

| Name | Value | Last Refreshed Date | Last Used Date | Permissions |
|--------|------------|---------------------|-------------------|-------------|
| test | hf_...UlnS | 12 days ago | 10 days ago | WRITE |
| Access | hf_...fdQx | Sep 17 | - | WRITE |
| Write | hf_...HuBQ | Sep 11 | about 2 hours ago | WRITE |

Set token type to “Read” and select “Create token”.

< **Create new Access Token**

Token type

Fine-grained **Read** Write

⚠ This cannot be changed after token creation.

Token name

Token name

This token has read-only access to all your and your orgs resources and can make calls to inference API on your behalf. It can also be used to open pull requests and comment on discussions.

Create token

Copy the contents of your access token somewhere you won’t lose it, but don’t share it with anybody else as it provides direct control and access to your Hugging Face account.

Save your Access Token

Save your token value somewhere safe. You will not be able to see it again after you close this modal. If you lose it, you'll have to create a new one.

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 Copy

| Name | Permissions |
|----------|-------------|
| Hello :) | READ |

Done

Sign in to Hugging Face on Python

Open a Python terminal window and enter the following commands:

```
conda activate rag_team5
huggingface-cli login
```

A dialog will appear prompting you to enter your access token. Paste your token by right clicking the terminal window and then pressing enter.

```
> To login, `huggingface_hub` requires a token generated from  
https://huggingface.co/settings/tokens.  
> Token can be pasted using 'Right-Click'.  
> Enter your token (input will not be visible):
```

If the token is successful, you should successfully have logged in to Hugging Face from your Python virtual environment. This will allow you to access Meta Llama 3.1 when running the solution locally.

```
> Token is valid (permission: read).
> Your token has been saved to C:\Users\<You>\.cache\huggingface\token
> Login successful
```

System Usage Guide

Deploying the RAG App Locally

To run the RAG system locally, open a terminal window in the project directory and execute the following:

```
cd <YOUR PROJECT DIRECTORY>/ProjectCSurvival
conda activate rag_team5
python app.py
```

The RAG app will launch on *localhost* under the URL <http://localhost:7860/>

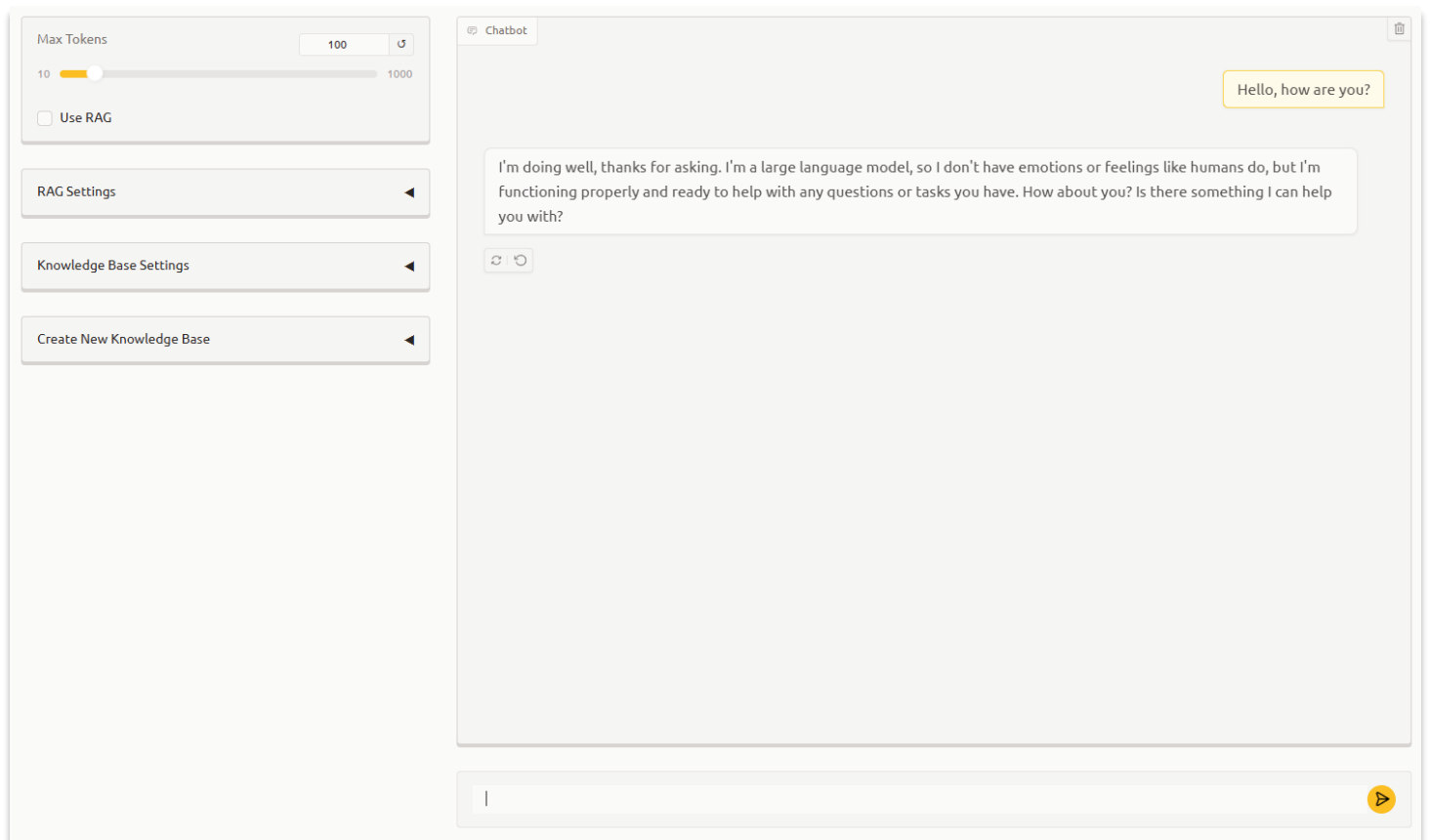


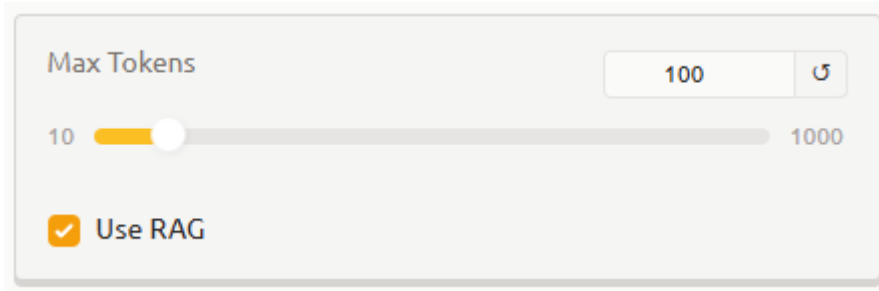
Figure 1 - The RAG interface

Using the RAG Interface

This chapter shall outline how the RAG system can be used to answer user questions from the Gradio user interface.

Output Settings

The settings panel on the top-left side of the user interface allows you to modify basic settings, such as the maximum number of words the LLM is allowed to output (“Max Tokens”) and whether RAG should be used or not to answer a user query.



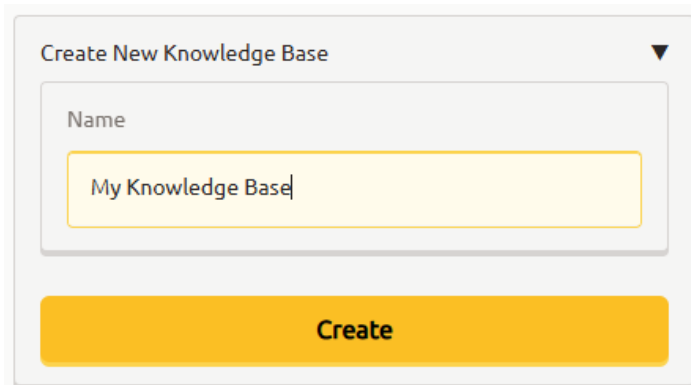
The screenshot shows a settings panel with a title bar. Inside, there is a section for "Max Tokens" with a numeric input field set to "100" and a refresh icon. Below this is a horizontal slider ranging from "10" to "1000", with a yellow bar and a white knob. At the bottom, there is a checkbox labeled "Use RAG" which is checked.

RAG Usage and Configuration

The RAG system excels at question answering (QA), but it requires a *knowledge base* (KB) to draw on to successfully answer user queries. This chapter shall explain how you can create knowledge bases to make the RAG an expert at any topic you choose.

Creating a Knowledge Base

To create a new KB, expand the “Create New Knowledge Base” accordion on the left. Enter a title for your KB, then select “Create” to create it.



The screenshot shows a form titled "Create New Knowledge Base" with a dropdown arrow. Inside the form, there is a "Name" label above a text input field containing "My Knowledge Base". Below the input field is a large yellow button labeled "Create".

Adding Contexts to the Knowledge Base

To add new context documents to your knowledge base, expand the “Knowledge Base Settings” accordion and select “Upload PDF Document”.

Knowledge Base Settings

Knowledge Base

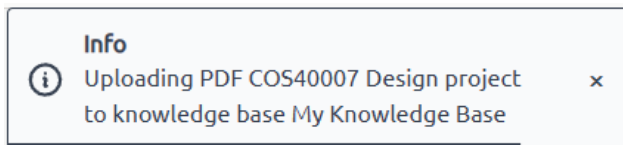
My Knowledge Base

Number of articles:

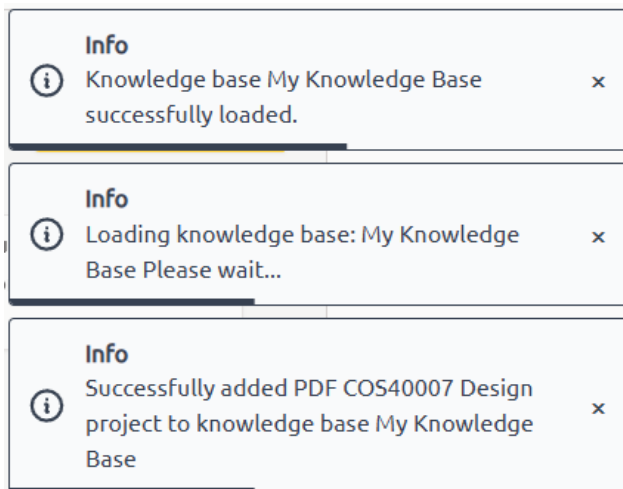
0 articles

Upload PDF Document

Then, select a PDF file to upload into the system to use as context. The PDF file will be uploaded to the knowledge base.



Once the context file is successfully loaded, the knowledge base will reload.



Configuring the RAG System

More detailed RAG parameters can be modified in the “RAG Settings” accordion. These control the number of times the RAG system is allowed to attempt to answer a single question via iterative reasoning and retrieval before giving up and responding, “I don’t know”.

RAG Settings

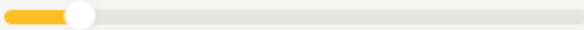
Maximum Sub-Questions

How many times shall the RAG be allowed to decompose the user question into smaller sub-questions?

1  10

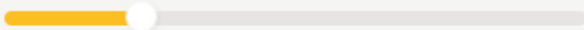
Maximum Answer Attempts Per Sub-Question

During each answer attempt, the RAG will fetch a different document to answer the sub-question.

1  10

Number of Paragraphs Per Article

How many paragraphs should be retrieved for each document during the sub-question answer attempt?

1  10

Creating a Knowledge Base from Wikipedia

It is possible to create a new knowledge base for the RAG using a Wikipedia dump to automatically generate context documents, however this functionality was not directly added into the user interface for the RAG system. This chapter shall outline how a Wikipedia knowledge base can be created using some additional steps.

1. Run the RAG FastAPI

The methods to create a Wikipedia knowledge base for the RAG system are implemented in the RAG's API, which is written in FastAPI. To run this API, first stop running the RAG Gradio UI if you are running it, then use the command:

```
cd <YOUR PROJECT DIRECTORY>/ProjectCSurvival
conda activate rag_team5
fastapi dev api.py
```

This should output:

```
INFO:      Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:      Started reloader process [19684] using WatchFiles
INFO:      Started server process [21272]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
```

Proceed to <http://localhost:8000/docs> to access the API.

FastAPI 0.1.0 OAS 3.1
/openapi.json

default ^

| | | | |
|------|----------------------------|---------------------------|---|
| GET | /download_wikipedia_dump/ | Download Wikipedia Dump | ▼ |
| GET | /generate_knowledge_base/ | Generate Knowledge Base | ▼ |
| POST | /add_pdf_to_knowledge_base | Add Pdf To Knowledge Base | ▼ |
| GET | /query/{query} | Query Lim | ▼ |
| GET | /query_rag/{query} | Query Rag | ▼ |

2. Download a Wikipedia Dump as Raw Text

To create a knowledge base from Wikipedia, you will first need to download English Wikipedia to your computer as raw text using the API method `/download_wikipedia_dump/`. This will download either the entirety or a subset of English Wikipedia to your computer as a .XML.BZ2 archive, and then extract all articles from this archive into raw text [26] and delete the archive.

FastAPI 0.1.0 OAS 3.1
/openapi.json

default

GET /download_wikipedia_dump/ Download Wikipedia Dump

Download a Wikipedia dump, convert it to raw text, and save it to `output_dir`.

Parameters

| Name | Description |
|---------------------------------------------------------------|----------------------------------------------------------------------------|
| <code>dump_url</code> string (query) | <input type="text" value="https://dumps.wikimedia.org/enwiki/latest/env"/> |
| <code>output_dir</code> string (query) | <input type="text" value="context/raw_text"/> |
| <code>subfile_max_size_megabytes</code> integer (query) | <input type="text" value="10"/> |
| <code>megabyte_limit</code> integer (query) | <input type="text" value="megabyte_limit"/> |

Execute

Parameters:

- | | |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| • dump_url | Where to download the Wikipedia dump from. This URL must link to a Wikipedia dump in .XML.BZ2 format. Defaults to the latest English Wikipedia dump. |
| • output_dir | Where to export the Wikipedia dump to. Defaults to <code>ProjectCSurvival/context/raw_text</code> . |
| • subfile_max_size_megabytes | When the Wikipedia dump is converted into raw text, it is split into a number of smaller files (chunks). This argument specifies the maximum size for each raw text Wikipedia chunk in megabytes. Defaults to 10Mb. |
| • megabyte_limit | Optional. If given, this argument specifies how many megabytes of the Wikipedia dump to download. E.g., if this argument is set to “15”, the first 15Mb of the latest English Wikipedia dump will be downloaded and converted to raw text and the rest will be discarded. If left empty, downloads the entire latest English Wikipedia dump. |

3. Convert Wikipedia Dump into Knowledge Base

To convert the Wikipedia dump into a knowledge base for use with the RAG system, use the API method `/generate_knowledge_base/`.

GET `/generate_knowledge_base/` Generate Knowledge Base

Converts a raw text knowledge corpus into a NumPy array of chunked embeddings and saves the resulting array to `output_dir` .

Articles are processed in batches of megabyte size `batch_size_mb` .

Parameters

| Name | Description |
|-----------------------------------------------------------|-----------------------------------------------------|
| <code>wikipedia_raw_text_path</code> string (query) | <input type="text" value="context/raw_text"/> |
| <code>output_dir</code> string (query) | <input type="text" value="context/knowledge_base"/> |
| <code>batch_size_mb</code> integer (query) | <input type="text" value="50"/> |

Execute

Parameters:

- **wikipedia_raw_text_path**

The file path of the Wikipedia dump to convert. Defaults to `ProjectCSurvival/context/raw_text`, the same path set by `/download_wikipedia_dump/` by default.

- **output_dir**

Which folder to save the created knowledge base to. The Gradio UI scans for knowledge bases in the `ProjectCSurvival/context` folder, so it is best to save your knowledge base to this folder.

- **batch_size_mb**

If you specify the name of an existing knowledge base, the Wikipedia articles will be added to that knowledge base, otherwise a new folder will be created with the Wikipedia articles. Defaults to `ProjectCSurvival/context/knowledge_base`.

The process to convert Wikipedia articles into embeddings for the knowledge base is handled in batches to avoid out of memory errors. This argument controls how many megabytes of Wikipedia raw text articles should be processed at once. Defaults to 50Mb.

Limitations of the RAG

Unfortunately, it is very difficult to set up the RAG with a knowledge base containing the entirety of English Wikipedia, because it takes time and compute power to download the latest English Wikipedia dump, extract it as raw text, and convert the text into embeddings.

It is the optimised components and research guided techniques that enabled the success of the team's RAG solution. But to *critically analyse* the success of this solution, we should consider the subject of LLMs and their ability to hallucinate and see if the solution resolves or alleviates its constraints. However, the solution did face challenges, hardware limitations and complexity of components lead the team to managing trade-offs from a theoretically more efficient solution to a more practical and available one. Limitations in the form of retrieving with real-time or up-to-date context was difficult to guarantee which occasionally resulted in outdated responses, though a compromise of calling an available static Wikipedia dump file that is *frequently* updated sufficed when considering an effective solution. Another instance of compromise of efficiency for practicality was apparent with both paragraph level chunking over late chunking and intuitively using the first sentence of a Wikipedia chunk as the summary over summarising every individual chunk through another LLM.

Conclusion

The implemented RAG system proved to be an effective strategy for mitigating hallucinations in LLM-generated responses. By integrating a dynamic external knowledge base and employing optimised retrieval techniques, the system enhanced response accuracy, especially for complex queries. However, the inability to use the HaluEval benchmark limits the scope of our quantitative evaluation. Future work should address this constraint and explore additional benchmarks to validate the system's performance. Overall, our RAG approach highlights the potential of extrinsic memory systems in enhancing LLM reliability atop of its own parametric learning.

References

- [1] Z. Xu, S. Jain and M. Kankanhalli, "Hallucination is Inevitable: An Innate Limitation of Large Language Models," arXiv, 22 Jan 2024. [Online]. Available: <https://arxiv.org/abs/2401.11817>. [Accessed 8 November 2024].
- [2] V. Rawte, S. Chakraborty, A. Pathak, A. Sarkar, S. T. I. Tonmoy, A. Chadha, A. P. Sheth and A. Das, "The Troubling Emergence of Hallucination in Large Language Models -- An Extensive Definition, Quantification, and Prescriptive Remediations," 23 October 2023. [Online]. Available: <https://arxiv.org/abs/2310.04988>.
- [3] T. S.M Towhidul Islam, Z. S M Mehedi, J. Vinija, R. Anku, R. Vipula, C. Aman and D. Amitava, "A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models," arXiv, 8 January 2024. [Online]. Available: <https://arxiv.org/abs/2401.01313v3>. [Accessed 25 August 2024].
- [4] T. Vu, M. Iyyer, X. Wang, N. Constant, J. Wei, J. Wei, C. Tar, Y.-H. Sung, D. Zhou, Q. Le and T. Luong, "FreshLLMs: Refreshing Large Language Models with Search Engine Augmentation," arXiv, 2023. [Online]. Available: <https://arxiv.org/abs/2310.03214>. [Accessed 12 September 2024].
- [5] B. P. R. Deepak, "Exploring the LLM Landscape: From Parametric Memory to Agent-Oriented Models," Medium, 2023. [Online]. Available: <https://medium.com/@prdeepak.babu/exploring-the-llm-landscape-from-parametric-memory-to-agent-oriented-models-ab0088d1f14>. [Accessed 25 August 2024].
- [6] H. Cao, "Recent advances in universal text embeddings: A Comprehensive Review of Top-Performing Methods on the MTEB Benchmark," 2024. [Online]. Available: <https://arxiv.org/html/2406.01607v2#S2>.
- [7] Z. Li, Z. Zhang, Y. Zhang, D. Long, P. Xie and M. Zhang, "Towards General Text Embeddings with Multi-stage Contrastive Learning," 2023. [Online]. Available: <https://arxiv.org/abs/2308.03281>.
- [8] X. Zhang, M. Wang, X. Yang, D. Wang, S. Feng and Y. Zhang, "Hierarchical Retrieval-Augmented Generation Model with Rethink for Multi-hop Question Answering," arXiv, 2024. [Online]. Available: <https://arxiv.org/abs/2408.11875>. [Accessed 29 September 2024].
- [9] Y. A. Malkov and D. A. Yashunin, "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 824-836, 2018.
- [10] O. Press, M. Zhang, S. Min, L. Schmidt, N. A. Smith and M. Lewis, "Measuring and Narrowing the Compositionality Gap in Language Models," arXiv, 2022. [Online]. Available: <https://arxiv.org/abs/2210.03350>. [Accessed 12 September 2024].
- [11] J. Wu, L. Yang, Y. Ji, W. Huang, B. F. Karlsson and M. Okumura, "GenDec: A robust generative Question-decomposition method for Multi-hop reasoning," arXiv, 2024. [Online]. Available: <https://arxiv.org/abs/2402.11166>. [Accessed 24 September 2024].
- [12] B. Gates, "AI Is About To Completely Change How You Use Computers," 9 November 2023. [Online]. Available: <https://www.gatesnotes.com/AI-agents>.
- [13] K. Adams, "How Often Do LLMs Hallucinate When Producing Medical Summaries?," August 11 2024. [Online]. Available: <https://medcitynews.com/2024/08/ai-healthcare-llm/>.
- [14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention Is All You Need," arXiv, 12 Jun 2017. [Online]. Available: <https://arxiv.org/abs/1706.03762>. [Accessed 8 November 2024].

- [15] X. Li and J. Li, "AnglE-optimized Text Embeddings," arXiv, 2023. [Online]. Available: <https://arxiv.org/abs/2309.12871v8>.
- [16] M. Elaraby, M. Lu, J. Dunn, X. Zhang, Y. Wang, S. Liu, P. Tian, Y. Wang and Y. Wang, "Halo: Estimation and Reduction of Hallucinations in Open-Source Weak Large Language Models," arXiv, 2023. [Online]. Available: <https://arxiv.org/abs/2308.11764>. [Accessed 2024].
- [17] NVIDIA Corporation, "CUDA Toolkit," NVIDIA Corporation, 2024. [Online]. Available: <https://developer.nvidia.com/cuda-toolkit>. [Accessed 8 November 2024].
- [18] I. Ilin, "Advanced RAG Techniques: An Illustrated Overview," Medium, 2024. [Online]. Available: <https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6>. [Accessed 25 August 2023].
- [19] A. Small, "COS30018 Individual Fortnightly Report 1," Swinburne University of Technology, Melbourne, 2024.
- [20] F. Clementine, H. Nathan, L. Alina, S. Konrad and W. Thomas, "Open LLM Leaderboard," Hugging Face, 26 June 2024. [Online]. Available: https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard. [Accessed 5 September 2024].
- [21] Z. Cai, M. Cao, H. Chen, K. Chen, K. Chen, X. Chen, X. Chen, Z. Chen, Z. Chen, P. Chu, X. Dong, H. Duan, Q. Fan, Z. Fei, Y. Gao, J. Ge, C. Gu, Y. Gu, T. Gui, A. Guo, Q. Guo and He, "InternLM2 Technical Report," arXiv, 26 March 2024. [Online]. Available: <https://arxiv.org/abs/2403.17297>. [Accessed 11 November 2024].
- [22] Meta, "Meta-Llama-3.1-8B-Instruct," Hugging Face, 2024. [Online]. Available: <https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct>. [Accessed 25 August 2023].
- [23] A. Small, "COS30018 Individual Fortnightly Report 2," Swinburne University of Technology, Melbourne, 2024.
- [24] A. Small, "COS30018 Individual Fortnightly Report 4," Swinburne University of Technology, Melbourne, 2024.
- [25] Hugging Face, "Transformers," Hugging Face, 2024. [Online]. Available: <https://huggingface.co/docs/transformers/en/index>. [Accessed 12 September 2024].
- [26] G. Attardi, "WikiExtractor," GitHub, 2015. [Online]. Available: <https://github.com/attardi/wikiextractor>. [Accessed 16 October 2024].
- [27] A. Abid, A. Abdalla, A. Abid, D. Khan, A. Alfozan and J. Zou, "Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild," arXiv, 6 June 2019. [Online]. Available: <https://arxiv.org/abs/1906.02569>. [Accessed 13 November 2024].
- [28] A. Solatorio, "NoInstruct small Embedding v0," Hugging Face, 2024. [Online]. Available: <https://huggingface.co/avsolatorio/NoInstruct-small-Embedding-v0/>. [Accessed 12 September 2024].
- [29] 4.-b. q. a. Q. Making LLMs even more accessible with bitsandbytes, "Belkada, Younes; Dettmers, Tim; Pagnoni, Artidoro; Gugger, Sylvain; Mangrulkar, Sourab," Hugging Face, 24 May 2023. [Online]. Available: <https://huggingface.co/blog/4bit-transformers-bitsandbytes>. [Accessed 10 September 2024].
- [30] Y. Belkada and T. Dettmers, "A Gentle Introduction to 8-bit Matrix Multiplication for transformers at scale using Hugging Face Transformers, Accelerate and bitsandbytes," Hugging Face, 17 August 2022. [Online]. Available: <https://huggingface.co/blog/hf-bitsandbytes-integration>. [Accessed 10 September 2024].
- [31] N. Muennighoff, N. Tazi, L. Magne and N. Reimers, "Massive Text Embedding Benchmark

- Leaderboard,” Hugging Face, 2022. [Online]. Available: <https://huggingface.co/spaces/mteb/leaderboard>. [Accessed 12 September 2024].
- [32] A. V. Solatorio, “GISTEmbed: Guided In-sample Selection of Training Negatives,” arXiv, 26 February 2024. [Online]. Available: <https://arxiv.org/abs/2402.16829>. [Accessed 12 September 2024].
- [33] J. Manser, K. Sherri, M. Crick, B. Dorevitch, R. Brohier and A. Small, “PlantProtect Final Report,” Swinburne University of Technology, Melbourne, 2024.
- [34] J. Schnitzler, X. Ho, J. Huang, F. Boudin, S. Sugawara and A. Aizawa, “MoreHopQA: More Than Multi-hop Reasoning,” arXiv, 2024. [Online]. Available: <https://arxiv.org/abs/2406.13397>. [Accessed 28 September 2024].