Scientific Report Title

Author Name

June 23, 2025

Abstract

This is the abstract section. It should provide a concise summary of the research, including the main objectives, methods, results, and conclusions. The abstract should be self-contained and typically not exceed 250 words.

1 Introduction

The introduction should provide the background and context for your research. It should:

- Research Problem Presented, and Claims in the Original Paper
- Retrieval Augmented Generation
-
- Explain the significance of the work

Large Language Models (LLMs) have the potential to help scientists with retrieving, synthesizing, and summarizing the scientific literature CITE. However, issues such as hallucinations CITE, lack of detail CITE, and underdeveloped retrieval and reasoning benchmarks, hamper the direct use of LLMs in scientific research.

The field is rapidly developing, with new models such as Google's Gemini 2.5 CITE and OpenAI's o3 CITE being able to 'reason', and excel at coding, maths, and langaage benchmarks. This also highlights the developments of new cutting-edge benchmarks for scientific performance in areas such as scientific discovery CITE, analysis CITE, reasnoning CITE, programming CITE, CITE, and mathematics CITE.

Alongside the development of the fundamental models themselves, techniques such as Retreival Augemented Generation (RAG) and the use of Multi-Agent Systems (MAS) allowed better use of LLMs in scientific research.

1.1 Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) is a sophisticated technique designed to enhance the capabilities of Large Language Models (LLMs) by grounding their responses in external knowledge sources. This process mitigates common LLM issues like hallucinations and outdated information by dynamically providing relevant, fact-based context. The RAG process can be broken down into several key stages:

1. **Indexing**: A corpus of documents (e.g., scientific papers, reports, web pages) is processed into a searchable format. This involves loading the documents, splitting them into smaller, manageable chunks of text, and converting these chunks into numerical representations called vector embeddings using an embedding model. These embeddings are then stored in a specialized vector database, which allows for efficient similarity searching.

2. **Retrieval**: When a user submits a query, it is also converted into a vector embedding using the same model. The vector database is then searched to find the document chunks with embeddings that are most similar to the query embedding. This similarity search efficiently identifies the most relevant information in the knowledge base.

3. **Augmentation and Generation**: The retrieved document chunks are then combined with the original user prompt and fed as context to an LLM. The model uses this augmented prompt to generate a response that is synthesized directly from the provided information, ensuring the answer is relevant, accurate, and grounded in the source material.

This architecture is powerful because it separates the knowledge base from the generative model, allowing the knowledge to be updated without needing to retrain the LLM.

INCLUDE A PLOT AND EXPLAIN HOW RAG WORKS

1.2 Multi-Agent Systems

Multi-Agent Systems (MAS) represent a shift from using a single LLM to orchestrating multiple, specialized LLM-powered agents that collaborate to solve complex problems. This approach is analogous to a team of human experts, where each member has a specific role and set of tools. Instead of being "trained" in the traditional sense, these agents are *instructed* through carefully crafted prompts that define their roles, goals, and constraints. Their power comes from a well-defined system of interaction and tool use.

The development of a MAS typically involves three key components:

1. **Role Specialization**: Each agent is given a specific persona or role (e.g., "Planner," "Code Critic," "Financial Analyst") through its system prompt. This focuses the agent's behavior on a specific part of the problem.
2. **Tool Use**: Agents are equipped with specific tools, such as web search APIs, code interpreters, or private knowledge bases, allowing them to interact with the outside world and perform actions beyond text generation.
3. **Orchestration**: A framework is used to manage how the agents collaborate. This can be a hierarchical structure where a "manager" agent delegates tasks, a sequential process where tasks are passed down a pipeline, or a conversational "group chat" where agents can interact more dynamically.

Several powerful frameworks have emerged to facilitate the creation of these systems. **Microsoft's AutoGen** CITE excels at creating flexible, conversation-based agents that can dynamically interact. **LangGraph**, CITE an extension of the popular LangChain library, provides more explicit control by allowing developers to define workflows as stateful graphs, which is ideal for complex and cyclical processes. **CrewAI** CITE offers a more intuitive, high-level abstraction, focusing on creating a "crew" of agents with predefined roles to tackle tasks in a structured, sequential manner. These tools mark a major leap forward toward automated scientific discovery and complex problem-solving.

1.3 PaperQA2

To make scientific discoveries, one must be able to synthesise scientific knowledge. Skarlinski et. al. believe this can be broken down into three vital tasks: scientific question answering, summarising, and detecting contradictions in the literature. In their paper, it is shown that their developed tool, PaperQA2, outperforms the human benchmark in all three areas CITE. The focus of this report is to understand and reproduce the question-answering result.

1.3.1 PaperQA2 Architecture

PaperQA2 operates as an agentic Retrieval-Augmented Generation (RAG) system built upon the `paperqa` Python package. Its architecture enables a flexible workflow where an agent can operate on a pre-indexed local corpus or dynamically search for and ingest new documents at query time. The typical agentic query process involves the following stages:

1. Paper Search and Dynamic Ingestion: The agent's workflow begins by invoking a search tool, such as one that interfaces with the `Semantic Scholar` API. The agent first uses an LLM to distill the user's query into a set of precise keywords. These keywords are used to query the external academic database, which returns a list of candidate papers. The agent then retrieves the relevant documents (e.g., as PDFs) and parses them to extract their content. While the framework includes a built-in `PyMuPDF` parser for text extraction, it can also be integrated with more advanced external tools like `Grobid`, which can parse the full document structure (e.g., title, sections, references). The extracted text is then segmented into configurable, overlapping chunks. These chunks are immediately converted into vector embeddings using a configurable model and indexed in both a keyword store (`tantivy`) and a vector store (`Numpy` or `Qdrant`). This on-the-fly ingestion process makes the newly found papers available for the subsequent stages of the current query.

2. Evidence Gathering and Re-ranking: With a set of newly ingested chunks available, the system proceeds to a more nuanced, multi-step evidence-gathering phase to refine the context. This involves:

- **Vector and Hybrid Search:** A vector cosine similarity search is performed on the candidate chunks. The framework supports **hybrid search**, a technique that combines two different kinds of vectors for ranking. It uses dense vector embeddings, which capture the semantic or conceptual meaning of the text, along with sparse keyword-based vectors (e.g., TF-IDF or SPLADE-style models?), which excel at exact term matching. By merging scores from both, hybrid search retrieves chunks that are both thematically related and contain precise keywords from the query.

- **Result Diversification:** To prevent the retrieval of highly redundant information from a single source, the system can apply **Maximal Marginal Relevance (MMR)**. Instead of just optimizing for similarity to the query, MMR iteratively selects chunks by balancing their relevance with their novelty compared to chunks already selected. This ensures the resulting evidence set is both on-topic and informationally diverse.

- **LLM-based Re-ranking and Contextual Summarization (RCS):** This is arguably the most critical and computationally intensive step, invoked by a `GatherEvidence` tool. The top k chunks from the previous steps (a number configurable via `answer.evidence_k`) are not used directly. Instead, each individual chunk is passed to an LLM with a specific prompt instructing it to summarize the chunk's content strictly in relation to the user's original query and to assign a relevance score. This use of an LLM as a "re-ranker" is highly effective at filtering out passages that may be semantically similar but not actually useful for answering the question, while simultaneously distilling the most important information.

3. Answer Generation: In the final stage, a `GenerateAnswer` tool compiles the highest-scoring summaries from the RCS step into a single context. The number of evidence pieces included is configurable via parameters like `answer.answer_max_sources`. This curated context, along with the original question, is formatted into a final prompt using a customizable template (`prompt.qa`). This allows for fine-grained control over how the LLM is instructed to synthesize the information. The model then generates a comprehensive answer that is directly grounded in the selected sources. The system formats this answer with inline citations that trace back to the original documents, providing a verifiable and trustworthy response.

4. Citation Traversal: As an optional step, the agent can employ a **Citation Traversal** tool to expand the scope of its search beyond the initial document corpus. After identifying a highly relevant paper, this tool can be used to analyze its bibliography or query external databases (like Semantic Scholar) to find papers it cites (backward traversal) or papers that cite it (forward traversal). This is a powerful technique for discovering foundational research or, conversely, the latest developments building upon a known paper. These newly discovered documents can then be ingested and indexed on-the-fly, allowing the agent to dynamically expand its knowledge base to answer a query more comprehensively.

1.3.2 Key Result

The paper claims that PaperQA's performance beats the human benchmark, specifically for the precision of questions answered, achieving a precision of 73.8%. The benchmark metrics are accuracy and precision.

$$Accuracy = \frac{CorrectQuestions}{AllQuestions} \tag{1}$$

$$Precision = \frac{CorrectQuestions}{AnsweredQuestions} \tag{2}$$

The process of evaluating the performance of PaperQA2 was to ask questions, where the answers were found in newly published papers and could not be inferred from either the title, abstract, or conclusion. The questions would also require some 'reasoning' and the answer could not be a direct quote from the paper. The questions were taken from recently published biology papers, post September 2021 (the GPT training cutoff). The assumed reason for this is to prevent the LLM agents from using their 'own knowledge', instead of reading and understanding the papers given to it.

The human benchmark was performed by giving Master's and PhD Biology students a financial incentive for getting answering the same multiple choice question. However, the human evaluators were not directly given the correct paper (or the correct paper within a group of papers - which is essentially how PaperQA performs evaluation). Instead, the evaluators were access to the internet and journal collections, but were told to not use generative AI tools such as ChatGPT.

2 Methods

2.1 Data Analysis

The first step was to collect and assemble the dataset for the LitQA benchmark. The questions, answers, and distractors, as well as the paper DOIs' were provided on FutureHouse's HuggingFace repository. Initially, two main issues were faced: only the training dataset was available, and that the papers themselves were not available. The test set was harder to access, but was eventually found. Then, with all of the DOIs, all of the individual paper PDFs were collected and assembled into the datasets.

The nexrt step was to analyse the papers themselves. During the collection of the papers, it was noticed that some of the dates did not meet the requirements stated by the paper authours. This was an issue because some of the results could then be affected and so the invalid papers were then identified, and it was found that 15 of the papers were from before the cutoff. FIGURE shows the years of the paper. These papers were excluded from the training dataset and a 'valid' dataset of pdfs was created.

The data from HuggingFace, which contained the question, correct answers and distractors, was turned into a `pandas` DataFrame. For every question, the correct answer and distractors were shuffled and then each assigned a letter to create a multiple choice question.

2.2 Result Reproduction

After the dataset was fixed, the next step was to reproduce the result using PaperQA. The package has a very intuitive interface, the user uses the `ask` function (which is just a syncronous wrapper for the asynchonous function `agent_query`), which then uses the prompt to find the relevant documents to answer the user's query. Each of the questions were fed to the `agent_query` and the asnwer was recorded.

The standard output of PaperQA is always a verbose summary of the answer, why the answer was chosen and citations from the paper that was used to answer the question. This is an extremely useful feature for scientists and researchers. However, this makes it difficult for us. We are looking for a single letter answer, so that it can easily be matched up with the corresponding correct result or distractors from the DataFrame. The initial thought was to use prompt engineering to try and force PaperQA to return the single correct letter.

EXAMPLE PROMPT

This method did manage to create an output with a single letter, but the agent would always return a short summary explaining the reaosning. This led to very inconsistent results (where the position of the single letter answer within the text would vary), and made it difficult create a consistent evaluation pipline.

The solution to this issue was to make use of structured outputs. Structured outputs allow users to customise the exact format of the output of an LLM. However, PaperQA uses `LiteLLM` to call LLMs for tasks such as Paper Retrieval, Evidence Gathering, and Answer Generation. Structured Outputs formats vary between different LLMs and can only be used if the LLMs are called directly. PaperQA calls LLMs through a third-party `LiteLLM`, and so there is no 'entry point' for Structured Outputs to be used.

2.3 Custom Multi-Agent Wrapper for PaperQA

The solution to the above problem came in the same format as PaperQA itself, to make use of multi-agent systems. Structured Outputs using json style formatting was possible using AG2. Using AG2's Conversable Agent (the most basic agent), a rudimentary linear system was developed to format both the input and output of PaperQA to a standardised format. This allowed a conisistent output from PaperQA. FIGURE shows the strucutre of the agent. The need for a consistent format of the input was related to the evaluation methodology.

2.4 Evaluation and Insepct AI

Inspect AI is a framework to evaluate the performance of LLMs by the UK's AI Securoty Institute. Inspect AI's `eval` tool allows for rapid performance evaluation by running LLM calls asynchonously. `eval` function consists of three parts: a dataset, a solver, and a scorer. The dataset is made up of `Samples`, which correspond to an individual task, or multiple choice question in this case. Each `Sample` contains fields such as inputs (which corresponds to our question), choices, and targets. The solver corresponds to the LLM system used to answer the question, and the scorer is the evaluation metric.

`inspect_evals` CITE is a custom library for evaluating various LLM's performances on benchamrks such as LitQA. However, it is only compatible with single LLMs, not multi-agent systems.

inspect_ai does offer support for multi-agent systems through the bridge function, but the package is new, and in its native form cannot support end-to-end MCQ evaluation with multi-agent systems. The bridge function effectly allows multi-agents to replace the solver, as long as its output follows a specific format. This format does not allow for information such as questions, choice, or target to be passed alonf in its native form, only allowing agent messages to be passed along. This prevents any information to the scorer the evaluation fails.

To this end, a custom package was built for multi-agent MCQ evaluation using inspect_ai. The package is built to intergrate easily with Inspect AI, and allows any multi agent system to be used, as long as it takes in a question and outputs a text reply. It levarages a custom Dataset builder, creating custom Samples by reading the the pandas DataFrames into the specified Structured Output JSON form. The custom Samples would be sent to a custom bridge agent containing the multi-agent MCQ function, which produced an answer that needed to be evaluted by a Scorer. Answers would be classified as CORRECT, INCORRECT, or NA. NA represents when the machine fails, or when there is insufficient information to answer the question. The answer will also be labelled as correct if NA is chosen and NA is the correct option. This was implemented as a custom inspect_ai scorer. The custom scorer function that can read the Structured Output JSON from AG2. From this, it converts the data into inspect_ai.Score object, which can be used by eval to calculate the performance of the LLM agent's performance across the whole task.

2.5 Hyperparameters

The main hyperparameter that is varied in this project is the LLM itself. In the paper, GPT-4-Turbo wsa the default LLM used for the tasks. However, with the pace of the new models coming out, and their improvements to performance, multiple models were tested for this evaluation to see the improvements in performance. Models tested were OpenAI's gpt-4o-mini, gpt-4.1, gpt-4-turbo. Due to budget restrictions, the defualt model used in the experiments was gpt-4o-mini, which offers equivalent or better performance as gpt-4-turbo CITE for a lower per-token cost.

Another hyperparameter that was changed was the text embedding model used to encode the text. The default model used was OpenAI's text-embedding-3, which at the time of writing was no longer the cutting-edge model. Instead, the best performing model available on LiteLLM (all models must be available on LiteLLM, as this is the LLM backend of PaperQA) was Google Gemini's text-embedding-004. Embeddings are crucial to all NLP related tasks, as the better the representation of the semantic meaning of text, the better chance the LLM can truly 'understand' the text, and make the best decision.

Within PaperQA there are also two key parameters that the authors varied to find out the effect of the performance. First was the Answer Cutoff (`max_sources`)hyperparameter, which limits the maximum number of sources used by PaperQA to generate its response. This happens after the RCS step, so it is looking at the top-ranked answers. The other hyperparameter is the Consider Sources or `evidence_k`, which happens during the initial document retrieval step. The best way to describe the difference between them is using an analogy: a researcher will take the 30 most promising articles from a library (this is the `evidence_k`) to complete this research. From these 30 articles, the researcher then skim reads them and picks the best 5 (this is the `max_sources`) to read thouroughly and use to inform their research. Within the paper, the default values for `max_sources` and `evidence_k` were 5 and 30, respectively. If the `evidence_k` is lower than the `max_sources`, then the `max_sources` would be automatically lowered to match. In the paper, the `max_sources` was set to 15, but its performance was worse, and the result displayed in the paper was using the an `max_sources` of 5, hence 5 was used as the defualt in the experiement.

TODO turn off RCS.

Other hyperparameters, such as temperature, were kept the same throughout the experiments.

3 Results

AG2 also uses LLMs, and changes the output of PaperQA/agent. Therefore, this adds uncertainty into the method. Checked every sample and the expected output has been outputted, so this is mitigated in this report, but should be aware for future usage.

4 Discussion

We used the test set. Smaller, therefore more uncertainty in the result. Test set papers are all valid. Did not use full dataset due to cost and time budget.

4.1 Paper Methodology and Results

4.1.1 Human Benchmark

The human benchmark defines whether LLM agents achieve superhuman performance. The criteria of the questions proposed seem suitable, with the requirements of being recently published and cutting-edge being mostly met. The majority of publications came from Nature and equivalent journals. Also, the fact that the answers to the questions cannot be inferred from the abstract or title should enforce the fact that the answers to the questions are not well known or trivial, requiring at least a basic level of reasoning to find the answers. Also, the fact that the incorrect options were distractors, often statistics or facts taken from the same paper and related statements (rather than, for exmaple, random numbers) means that the LLM agents and humans must truly understand the content, rather performing a simple search. Although this was the criteria set by the authours, the creation of the dataset was performed by external contractors, and as proved earlier, the dataset did not always meet these requirements (old, possibly well-known papers were included in the dataset). WIthout the time (and the biology expertise) to manually verify every question, this cast a doubt over the dataset, and hence both the human and the machine benchamrk.

In terms of the actual human performance, evalutors' academic backgrounds were not provided, although it might be possibly inferred as at least-univeristy level as the authours said that the evaluators had access to journals 'through their institutions'. Human evalutors were unrestricted, meaning they had access to the internet, journals and search engines, but not always the papers themselves. This implies that a large part of the task was actually finding the correct source for the question. IS THIS MIRRORED IN OUR RESULTS? LOOK AT THE DIFFERENCE IN EVIDENCE K and MAX SOURCES. Also, to maximise human performance, the evaluators were financially motivated to get as many questions correct as possible in a week. This makes the benchmark higher, which would try to emulate the maximum performance, which is approriate for testing if LLM agents are indeed superhuman. The evaluators were also told to not use generative search tools, but this was not enforaceble. Also, with financial incentive, there comes with a risk of cheating for financial gain. This risk seems fairly low, with the type of question making it very hard to cheat. So having faith in humanity, the human benchmark was accepted as valid.

4.1.2 PaperQA Results

The PaperQA result did have some major differences to the project experiemnts, some due to model differences, but also some due to general differences.

One difference was that the paper authors used custom HTTP environment to run paperqa. They used features such as MongoDB request caching, Redis object caching, global load-balancing, PostgreSQL DBs, custom cost monitoring, cloud storage and run orchestration. They said this did not affect per-run results, but increased scalability, measureability, and persistence. This highlighted a key difference in methodology, as this project was run locally on a single laptop, with average-at-best specs, local storage and local compute. Similar to the paper, the individual runs of PaperQA were always successful in testing, but running evaluations caused a lot of issues. Nearly of the errornous runs of this project was due to (wall-time) time-limit failures or LLM provider rate limits being hit. The time-limit limitation was due to the fact that the local compute would be having to run the `inspect_ai` evals alongside PaperQA calls and this was often a more compute-intensive task. The rate error limts came from running multiple PaperQA, AG2 and Inspect AI calls in parallel, often from the same API key. All of these errors would affect the result, specifically, the accruacy metric. Errornous runs would be marked as NA, which would being down the accuracy, and increase the precision (less questions answered). The authors say their custom HTTPs setup would decrease these problems, which in turn may have skewed the results they achieved, had they ran this on a local machine as ordinary users would.

4.2 Project Results

4.2.1 Methodology Breakdown

The methods employed in this project aimed to understand the source of and tackle any sources of bias and uncertainty in the original project.

The first difference in methodology, was that our project and experiemnts were only ran on the test dataset, rather than the full LitQA benchmark quiz. This was for a multitide of reasons. Firstly, was budgetary: LLM agents cost money to run, and the budget did not allow for mulitple runs of the full LitQA dataset to be run. Secondly, as proven by the authours of the paper, 147 of the questions in the LitQA had been read by Google's data aggregator and made the questions invalid. However, we do know if these questions had been updated in the LitQA benchmark and more importantly, if these had been scraped by Google, they could have been scraped by OpenAI and other LLM providers o train the latest models, which defeates the purpose of testing these agents on their ability to synthesise new scientific information.

Another point of possible error was the use of AG2. AG2 also leverages LLMs for the formatting of the input/output. Therefore there was a risk of AG2 hallucinating, creating incorrect results. Although this was ahrd to confirm for every answer in every run, the

AG2 structuring agents were tested thouroughly during prototyping and they returned the expected results every time. For all of the evaluation runs, this effect was checked for all of the questions, and all of the answers were valid.

The use of `inspect_ai` was to follow standardised methods for evaluation. Since LLMs and LLM agents are such a new and rapidly evolving field, there is no standardised workflow for new systems. This meant that integrating PaperQA into AG2 into Inspect AI was extremely difficult. Inspect AI is also a very new tool, missing a lot of the functionailty required t evaluate multi-agent systems. Using Inspect AI has many benefits, such as running LLM calls asynchronously and in parallel, which theoretically should provide a speedup. However, there are also downsides. The eval tool prevents a lot of visibility and control over what is happening under the hood. It was also very difficult to pass along information along the `eval` pipeline, which facilitated the need to AG2 and also caused a lot of problems with cost tracking.

4.2.2 Hyperparameter Effects

Evidence K and Max Sources should not make a huge effect on the answer, as all of the answers for each question come from a separate paper (no answers to a question come from multiple papers). But it does. THis is because the papers do not make up a single chunk. Instead, papers are often split up into multiple chunks. Also since all of the papers are biology, they are semantically similar, and so poor embeddings would create confusion and poor ranking.

4.2.3 Issues

One very important issue noticed was the flucatuation in performance depending on LLM usage. Noticed quite late on in the project, after performing a sizable number of runs, it appeared that the performance of LLMs would change depending on when the tests would run even with temperature 0. Looking on forums CITE, it appeared that this was a common problem, especially with OpenAI. Looking into the GPT-4 architecture could return a reason why this is happening. There is no GPT-4 techical paper that details its ful archtitecture due to competition, but leaks, expoert analysis and even confirmation by Sam Altman CITE point ot the fact that GPt-4 is a Mixture of Experts architecture. It is essentially an ensemble model of transformers and neural networks, each specialised for a 'expert' topic, rather than a general transformer, and the query is sent to a number of 'experts' to answer the question. The inference is then distributed to a number of computers and recollected for minimum computational load and maximum speed. However, it is this distributed factor that may cause the performance degradation at peak times. One reason is that even though we set temperature to 0, since the inference is distributed, it is not guaranteed that the same configuration is sent to each device (possibly due to compression etc.), and so floating point errors could accumulate and potentially lead to different outputs. Looking at MOE, it could be possible that at peak times, when resources are contested heavily, models could be degraded to ensure everyone has access to the models (e.g. instead of 3 experts you are only allocated 2). Also, to balance the load, the experts could be given less time for the current user's prompt, so that it is freed for the next user's prompt, resulting in less time for processing. These are all potential ideas why this is happening and is not confirmed by providers such as OpenAI. This points to possibly using less popular providers for better performance.

5 Conclusion

Summarize the main findings and their significance.

Acknowledgments

Acknowledge any funding, technical support, or other contributions.

Acknowledgments

Acknowledge any funding, technical support, or other contributions.

References