Saleh Alkhalifa
Midpoint Update

CS 7290
April 20, 2025

SciRAD: Streamlined Retrieval, Aggregation, and Delivery of Scientific Insights

Abstract

The SciRAD project aims to address the critical challenge of information overload in scientific research by leveraging Generative AI agents to filter, summarize, and deliver relevant scientific summaries. This report outlines the work completed so far, highlights key achievements and challenges, and sets the direction for the next phase of development. A full demo showcasing the capability has been prepared and is functioning as expected using Streamlit. A comprehensive experimental design using MLFlow has been developed and run, demonstrating through data-driven approaches how effective the implementation can be. While two of the single-agentic frameworks have been mostly completed and show tremendous success, the third multi-agent framework has been challenging to implement given its complexity, and lack of foundation architectures to follow in the domain. This set back is minor, and will not affect the delivery of SciRAD. Currently, the project is on track (status is green, delivery is on time) with risks mitigated and deliverables being met on time as scheduled.

Introduction:

In today's information-rich world, we face an ever-growing challenge known as information overload. While we have an abundance of data at our fingertips, yet our capacity to process and consume this information remains limited. While this issue is apparent in many domains, it is particularly prevalent in the scientific and engineering domains where the volume of new research can overwhelm even the most diligent professionals. For example, PubMed alone is projected to report over 1.8 million scientific articles in 2024. Even if only 0.001% of these publications are directly relevant to an individual's field of expertise, this still represents an unmanageable volume to consume, analyze, and distill into actionable knowledge.

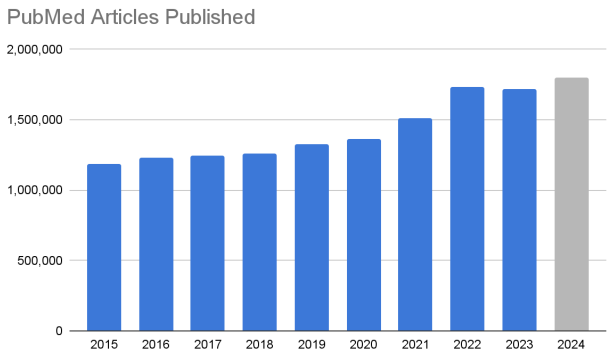


Figure 1: A bar graph showing increasing publications per year

While methods like Retrieval-Augmented Generation (RAG) have shown significant promise in recent years, one challenge is that they still rely on user-provided prompts to function effectively. In many cases however, users don't know what they don't know. Another challenge in the Information Retrieval domain is that customized results catered specifically to the user have grown in both popularity and success. To overcome these limitations, we introduce SciRAD, or Scientific Retrieval, Aggregation and Delivery: a platform designed to tackle this challenge by leveraging advanced data retrieval and artificial intelligence methods. SciRAD streamlines the processes of retrieval, aggregation, and summarization, enabling users to focus on meaningful work without being overwhelmed by unrelated data. By delivering personalized

weekly digests, the platform empowers researchers and engineers to stay updated on critical advancements through consumable bite-size portions of information while dedicating more time to their core responsibilities.

To bolster this view, advances in AI Agents have expanded the capabilities of large-language models by providing them with autonomous, goal-driven behavior. Modern agent frameworks—such as LangChain, LangGraph, and multi-agent orchestration platforms can enable LLMs to plan, execute, and iterate on complex workflows with minimal human intervention. These agents can invoke external tools, manage context across multiple steps, and even collaborate with other specialized agents to accomplish tasks that were previously infeasible with one-off prompts. As a result, AI Agents have shown remarkable promise in domains ranging from automated research assistance to intelligent data pipelines, demonstrating that LLMs can now not only generate text but also reason, retrieve information, and adaptively refine their outputs in real time. By incorporating AI Agents into SciRAD, we leverage this new technology to automate the end-to-end retrieval, aggregation, and summarization process—transforming how researchers interact with the ever-expanding scientific literature. There is an urgent need for a platform that can efficiently retrieve, organize, and distill scientific literature into actionable insights, tailored to the needs of each user.

Objectives

SciRAD aims to streamline how researchers and engineers interact with scientific literature by efficiently retrieving articles based on user-defined topics and preferences, aggregating related articles to uncover trends, and generating concise summaries using advanced Agentic Frameworks in conjunction with Large Language Models (LLMs). The idea here, depicted in the figure on the right, encompasses the agent which is equipped with tools such as the OpenAI API, metric calculators, and the ability to query the NCBI database. The agent is tasked with using these tools to generate a summary, and ensuring the quality of that summary with the judge model to ensure that the summary provides the user with a robust summary containing key insights pertaining to their field. These insights, one day beyond the scope of this project, can be delivered as optimized and personalized weekly digests, with a simple interface for managing subscriptions and selecting topics using AWS. In addition, SciRAD can be available as an open-source codebase and a customizable service for organizations seeking greater control over their infrastructure. Through this framework, the SciRAD product aims to achieve four key objectives:

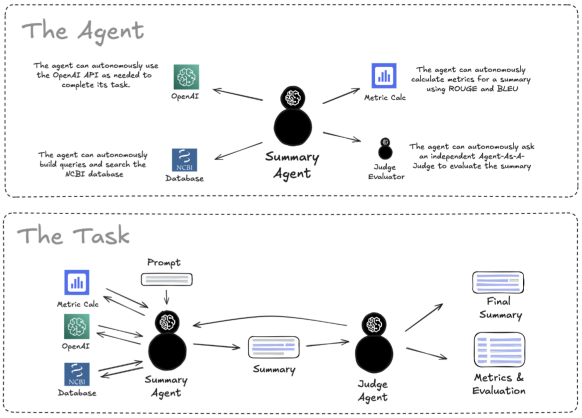


Figure 2: A depiction of the agent and task, showing the overall structure of the SciRAD framework

- **Efficiency**: Enable researchers to quickly consume key insights using summarization and ranking techniques.
- **Quality**: Maintain high quality summaries by ensuring that they accurately reflect the original content, confirmed through an agent-as-a-judge framework.
- **Personalization**: Tailor content based on individual researcher interests through Agent-driven keyword recommendations and ranking filters.

- **Scalability**: Lay the groundwork for a deployable product that can be packaged (via PyPi) or deployed on cloud infrastructures (such as AWS CDK).

Core Components

To achieve its objectives, SciRAD will implement a series of chronologically structured events. The idea here is that from the moment the user enters a query, the pipeline will retrieve content, aggregate it, summarize, and then deliver it back to the user.

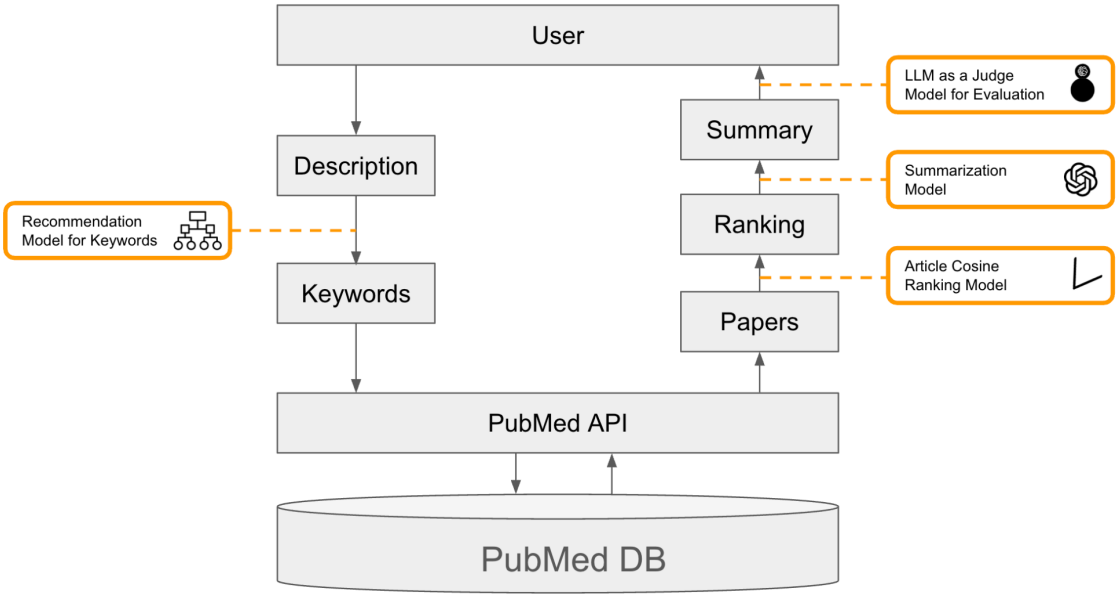


Figure 3: A graphical representation of the steps a user takes in their SciRAD journey starting from the description all the way to the final summary that is returned to the user.

1. **Retrieval**: Articles will be sourced from APIs such as PubMed's NCBI E-utilities, using NLP keyword recommendation to add/remove words with querying techniques to ensure relevance. Results will be ranked via. The recommend_keywords method is a lightweight, prompt-driven component that uses the agent's existing ChatOpenAI instance to generate research-relevant terms from an arbitrary description. Internally, it builds a single-turn conversational prompt that asks the LLM to "recommend n relevant keywords" (single words, comma-separated) for the provided user_description. After sending this prompt via self.llm, it strips and splits the raw response on commas to yield a Python list of tokens, trimming whitespace. A simple length check issues a logger warning if the model returns fewer than n items, showcasing potential prompt-response mismatches. Because it relies solely on the generative capabilities of the LLM rather than semantic embeddings or statistical extraction, its quality focuses on prompt clarity and the model's ability to interpret the instruction; it does not currently enforce strict list length or de-duplicate terms, since we treat this step as Human-in-the-loop as part of the overall pipeline. It is important to note that the Agent will automatically handle too little or too many results returned by the search terms. If too few, the Agent will extend the search back by an additional week. If the results are too many, a ranking algorithm is used to filter the most relevant results for the summary.

2. **Ranking**: When ranking the retrieved PubMed articles, we first convert each abstract to a fixed-length string (truncating to 4,000 characters if necessary) and embed it alongside the user's topic description using the same OpenAIEmbeddings model. This yields a "user" vector and one vector per abstract. For each abstract embedding, we compute cosine similarity as

```
sim = np.dot(UE, AE)  / (np.linalg.norm(UE) * np.linalg.norm(AE))
```

…where UE is User Embedding, and AE is Abstract Embedding. Any zero-vector case (where either embedding is all zeros) is assigned a similarity of 0.0 to avoid division errors. These similarity scores are then attached to each article under a `"similarity"` key, and the list is sorted in descending order of similarity so that the top entries are those most semantically aligned with the user's research focus. This process ensures that the papers most conceptually relevant to the description rise to the top of the results.

3. **Summarization**: The summarization component leverages the ChatOpenAI LLM (gpt-3.5-turbo) configured with a specific temperature and nucleus sampling to ensure deterministic, focused outputs. We construct a prompt that instructs the model to "Summarize the following articles focusing on [description]" and to produce a single coherent paragraph of approximately N words that highlights the key findings, methodologies, and implications tied to the user's keywords. Depending on the chosen prompting method (Zero-Shot, Chain-of-Thought, Few-Shot, etc.), we were able to optimize the quality of the summary. While we explored almost six different parameters, you can see a sample of the prompt method results in the figure on the right. In addition, we also prepend structured reasoning steps, example summaries, or role-based instructions to guide the model's output. After generating the text, we also log token usage and compute cost metrics, but the core of the summarization remains a concise, single-pass LLM invocation tailored to deliver an executive-style research overview.

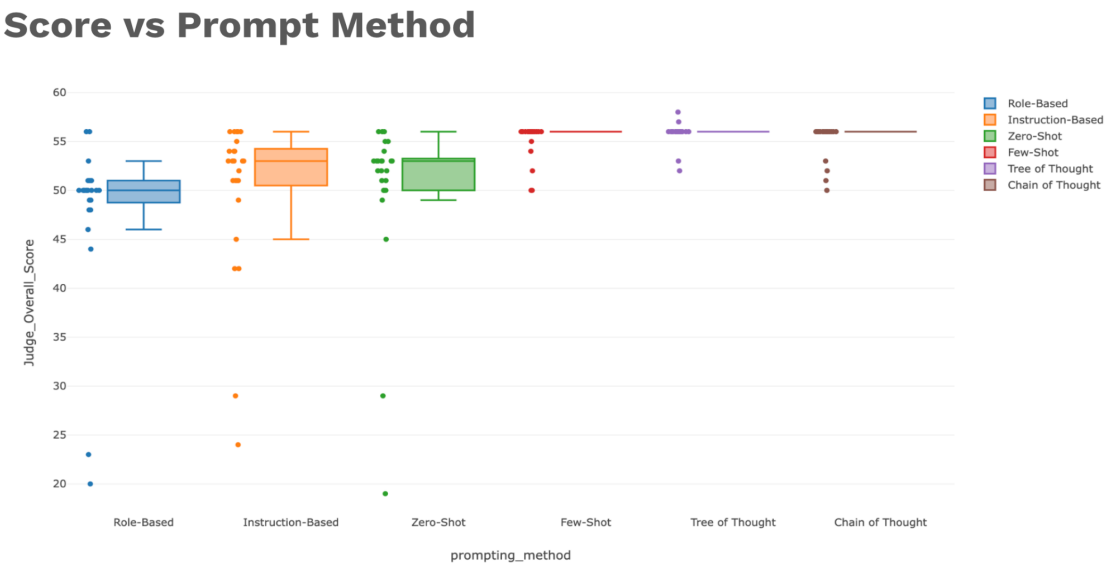


Figure 4: Sample experimental results from the analysis showing that the tree-of-thought and chain-of-thought prompts yielded the best scores relative to other prompting mechanisms.

4. **LLM-as-a-Judge:** The "LLM-as-a-Judge" module uses a separate ChatOpenAI instance (always gpt-3.5-turbo at zero temperature) to evaluate the quality of the summary against the original abstracts. We concatenate all retrieved abstracts and the generated summary, then prompt the judge to score the summary on six criteria:
    a. **Accuracy**: faithfulness to the articles' findings
    b. **Completeness**: covers of the all major points
    c. **Clarity**: readability, language, and structure
    d. **Conciseness**: brevity without loss of key details
    e. **Relevance**: focus on the topic without tangents or hallucinations
    f. **Objectivity**: absence of bias or opinion

    The judge returns a JSON object with a 0–10 score and brief comment for each criteria, plus an overall score out of 60. We purposefully frame the evaluation in this structured, criteria-driven format to get quantitative feedback and human-readable reasoning on how well the summary fulfills its research objectives. The model provides a summary reason for its selection for each criteria to give the user reasoning and clarity about its choices to avoid being a "black box model". In addition, we tested how the judge would react when we followed up on its results to assess whether the response was on the fence and would change, or if the response was strong and persistent. Randomly sampling the results, we see that the responses are strong and consistent, with reliable correlation.

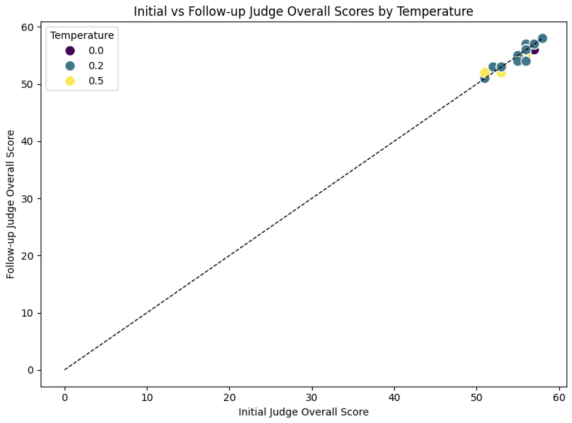


Figure 5: Results of a "Judge Follow-up" study where we asked the judge to reevaluate its conclusion. Results show the judge was quite consistent demonstrated by strong correlation in the results.

5. **Delivery**: Although deploying the application to deliver weekly digests isn't a formal deliverable for this project, we've laid the groundwork to make it possible by configuring the deployment pipeline so users can receive tailored insights in a convenient format. Subscription preferences will let individuals finetune their content even further. I wanted to capture this "what good looks like" vision, so we built a Streamlit demo environment so users can explore the app's capabilities, wrapped the core functionality into an easy-to-install Python package, and prepared a GitHub repository with clear documentation and setup instructions for easier adoption.

Experimental Results:

Experiments were conducted over a comprehensive hyperparameter optimization study using MLFlow to identify the optimal configuration of the OpenAI API settings and prompt templates for the summarization task. Each experiment was evaluated by an "LLM as a Judge" metric, in which scores were generated for summaries on six criteria: accuracy, completeness, clarity, conciseness, relevance, and objectivity, with each on a 0–10 scale for a maximum of 60 points. By systematically

logging model variants, prompt styles, sampling parameters, and resulting Judge scores, along with latency and cost, MLFlow enabled the ability to compare 192 unique configurations and identify the most effective strategies for high-quality, reproducible, and intuitive summarization.

The Judge metric was designed to capture the multifaceted nature of summary quality. Accuracy measures how faithfully the summary reflects the source material's key points, while completeness ensures that no essential findings are omitted. Clarity evaluates logical flow and readability, and conciseness upholds brevity without loss of critical information. Relevance penalizes inclusion of extraneous (hallucinated) content, and objectivity checks for an unbiased, factual tone. By aggregating these six subscores, the Judge metric provides a stronger assessment of summarization performance that aligns with user expectations for precise, comprehensive, and neutral summaries.

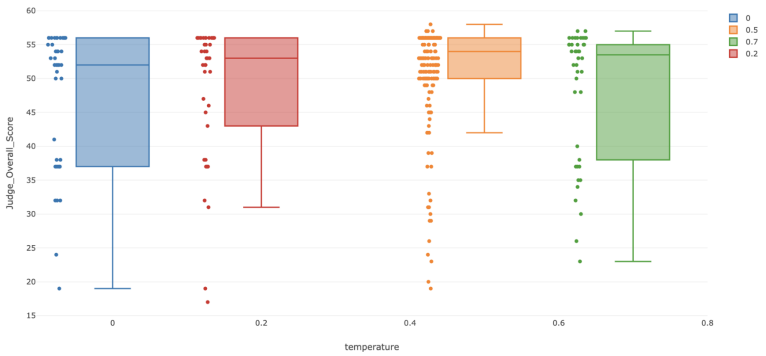


Figure 6: Sample results of the experimental study showing that the optimal temperature of the model was around 0.4-0.6

The hyperparameter search spanned three models (GPT-4, GPT-4o, and GPT-3.5-turbo), six prompt styles (role-based, instruction-based, zero-shot, few-shot, tree-of-thought, and chain-of-thought), four temperature settings ($T = 0.2, 0.5, 0.7, 1.0$), two top-p thresholds ($p = 0.9, 1.0$), and five word count limits (150, 200, 250, 300, 350). For each configuration, we generated summaries for a held-out set of n articles where n represented the number of articles retrieved, capturing Judge scores, average API latency, and per-query cost. This exhaustive grid search enabled the quantification of the effects of sampling parameters and prompting strategies on summary quality across not one, but two different AI Agent frameworks.

Analysis of the results revealed several clear trends. Judge scores peaked at a temperature of 0.5, where the balance between creative generation and factual consistency was optimal. Top-p of 1.0 provided the best trade-off between diversity (enhancing completeness) and precision (preserving accuracy). Tree-of-thought and chain-of-thought prompts consistently outperformed zero-shot and other variants by 6–8 Judge points on average, largely due to improvements in clarity and completeness. As expected, GPT-3.5-turbo and GPT-4 outscored GPT-4o by nearly 15 Judge points across configurations. Given that GPT-3.5-turbo performed as well as its counterpart, and its costs were significantly cheaper, it proved to be the more competitive model for the given task. Limiting the

maximum word count to 300 words yielded a modest completeness gain with minimal impact on latency relative to the other selections. It also produced the tightest Q1 and Q3 limits relative to others.

In summary, the analysis revealed that a temperature of 0.5 and top-p of 1.0 achieved the highest average Judge scores, striking the optimal balance between creativity and factual consistency. Tree-of-thought and chain-of-thought prompts outperformed all other styles by 6–8 points on average, primarily through gains in clarity and completeness. While GPT-4 and GPT-3.5-turbo both outscored GPT-4o by nearly 15 Judge points, GPT-3.5-turbo matched GPT-4's performance at a fraction of the cost, making it the most competitive choice for large-scale summarization. Finally, capping summaries at 300 words yielded modest improvements in completeness without significantly affecting latency, and produced the tightest interquartile range in Judge scores. These findings inform best practices for deploying cost-effective, high-quality LLM summarization systems.

Temperature	0.2	0.4	0.5	0.8	1.0
Top P	0.9	1.0			
Word Count	150	200	250	300	350
Prompting Mechanism	Role-Based	Instruction-Based	Zero-Shot	Tree of Thought	Chain of Thought
Summary Model Type	GPT-4	GPT-4o	GPT-3.5-Turbo		
Judge Model Type	GPT-4	GPT-4o	GPT-3.5-Turbo		
Framework	LC Single Agent	LG Single Agent	Multi-Agent		

Figure 7: A tabular representation of the results of the experimental study showing the optimal value for each parameter that maximized the judge overall score.

Core Deliverables:

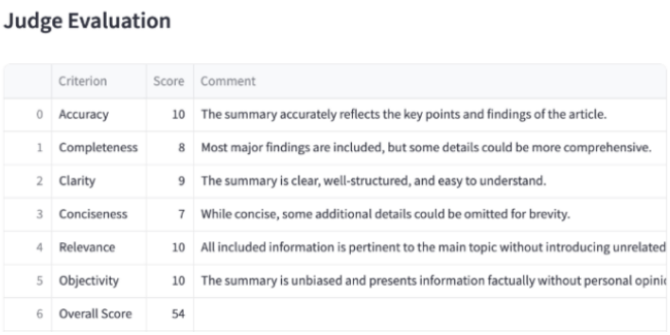
1. **Project Planning and Deliverables:**
   A full project plan was created and then approved by the professors, including milestones and detailed task assignments. The milestones are in chronological order, and will be achieved given the dependencies they have. All the milestones that were promised, were achieved. This includes the original plans such as the Agent, Ranking model, Keyword Model, Summary Model, and Judge Model. The feedback received to scope down the project to exclude AWS deployment, and include the re-prompting of the Judge was also accounted for. The only pivot made was to exclude the multi-agent application given that the performance of the single agent was sufficient. A copy of the project plan will be included in the appendix.

2. **Core Agentic Development:**
   We've built two robust single-agent pipelines with one using LangChain and the other leveraging LangGraph and both are designed to translate a user's area-of-interest description into actionable literature summaries. Each agent starts off by prompting an LLM to extract keywords from the description, then queries PubMed for papers published in the past two weeks, automatically extending the window by one-week increments if no results appear. The LangChain agent uses a ZERO_SHOT_REACT_DESCRIPTION setup with built-in tools (PubMedSearch, SummaryTool, …) and ConversationBufferMemory: it filters retrieved abstracts via cosine-similarity scoring against the user embedding, ranks the top N, truncates long texts to avoid token overload, and issues a single-pass summary prompt. On the other hand, the LangGraph agent constructs a lightweight semantic graph where each

paper is a node embedded alongside the user vector; it traverses high-similarity edges to isolate the most relevant subset before feeding those abstracts through a chain-of-thought prompt template to generate its summary. Both implementations ensure consistent, focused digests while showcasing two distinct yet complementary approaches to agentic research automation.

3. **Summarization Capabilities:**
We've built the agent to integrate a dedicated LLM-based summarizer via the OpenAI API, capable of distilling multiple abstracts into coherent, single-paragraph overviews. Early experiments demonstrate that it achieved a strong balance between brevity and informativeness, capturing essential findings based on the 80:20 rulet. While traditional metrics like BLEU and ROUGE offer quantitative gauges of overlap, they fall short of measuring "fit-for-purpose" summarization under an 80:20 lens. To bridge that gap, we built an LLM-as-a-Judge framework: after generating summaries, a second zero-temperature model scores each draft against criteria like accuracy and relevance, and a human reviewer spot-checks those scores to validate the judge's assessments. This human-in-the-loop step ensures our automated quality control aligns with real-world expectations. In addition, another re-prompt of the judge was included to determine how consistent the judge model is with its assessment. Early testing showed that the model was indeed consistent, with a high correlation of over 0.95 using random samples.

4. **Judge Agent Integration:**
A Judge Agent framework was implemented to evaluate the quality and consistency of the summaries, and ensuring that the output maintains high quality to the original content. The Judge Agent framework evaluates each summary through multiple criteria. First, it examines **accuracy** to ensure that the summary correctly reflects the key points and findings of the articles. It then assesses **completeness**, verifying that all major findings are included without omitting any essential information, which can sometimes be subjective. The judge then considers **clarity**, ensuring that the summary is well-structured, coherent, and easy to understand by the user. **Conciseness** is also critical, as the summary must be to the point, but without eliminating important details. In addition, the system reviews **relevance** by checking that all included information directly pertains to the main topic and avoids introducing unrelated content. Finally, **objectivity** is measured to confirm that the summary is presented in a factual and unbiased representation.


Judge Evaluation

	Criterion	Score	Comment
0	Accuracy	10	The summary accurately reflects the key points and findings of the article.
1	Completeness	8	Most major findings are included, but some details could be more comprehensive.
2	Clarity	9	The summary is clear, well-structured, and easy to understand.
3	Conciseness	7	While concise, some additional details could be omitted for brevity.
4	Relevance	10	All included information is pertinent to the main topic without introducing unrelated
5	Objectivity	10	The summary is unbiased and presents information factually without personal opini
6	Overall Score	54	

Figure 8: Sample judge evaluation showing the criteria, score, and the reasoning for each of the results to ensure transparency in the judge's overall score.

5. Optimization Using MLFlow:

The pipeline has been enhanced with MLFlow for parameter optimization. Various parameters such as temperature, top P, model type, and prompt strategies are being fine-tuned to improve overall performance. So far, several hundred experiments have been run in which these parameters were adjusted, and their scores calculated using the judge agent. Experiments have been conducted with different configurations and have provided insights into the optimal settings for balancing summary quality, computational efficiency, and cost.

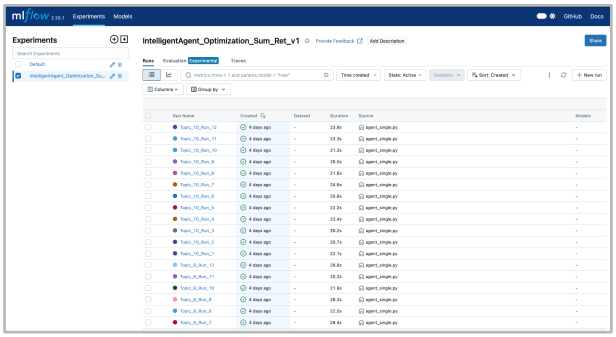


Figure 9: Sample results from MLFlow showing how experiments were documented and saved to identify the optimal values for each of the parameters.

6. Full Demo using Streamlit

As part of the mid year deliverables, a working demo built with Streamlit was prepared and offers an interactive visualization of the entire SciRAD pipeline. In the demo, users can (1) input search queries and immediately observe how the system recommends keywords (2). Next, the user is able to adjust the keywords as needed, part of the human-in-the-loop framework which upon acceptance then ignites the process to retrieve relevant scientific articles, applies cosine similarity filters, and generates (3) concise summaries using advanced LLMs. Users are not only provided the summary, but also the sources (4) to the specific articles from which the summary was generated. Next, the performance metrics (5) are also displayed to the user, in addition to the judge evaluation (6) which focuses on accuracy, completeness, clarity, conciseness, relevance, and objectivity. Finally, the total cost of the pipeline (7) is also displayed for the user. This dashboard has been helpful in validating the POC, adjusting inconsistencies, and highlighting both the positives and areas for improvement.

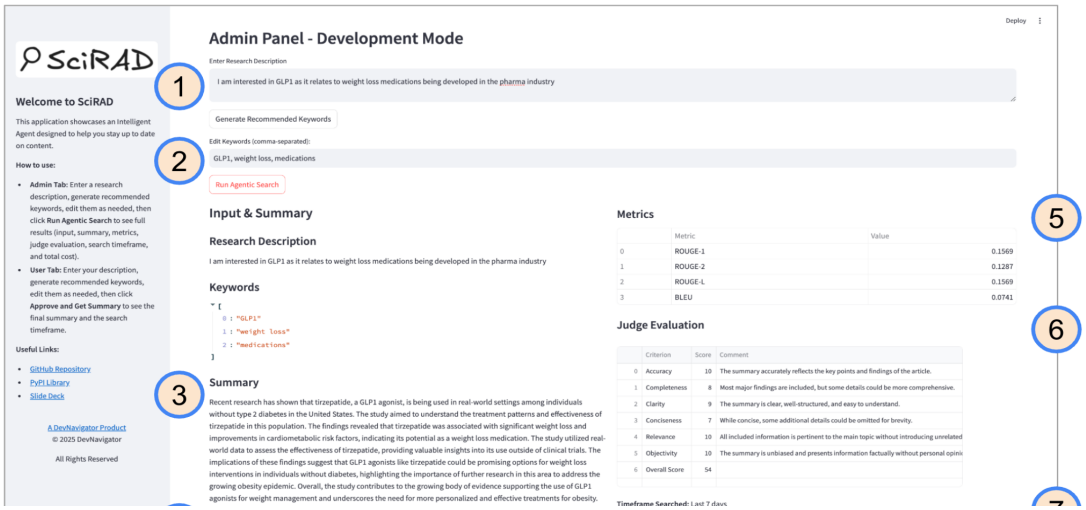


Figure 10: A demo of the application running in which each of the steps are broken out. The user starts by entering a description in (1) and ends by reviewing their results in (3,4). Metrics are made available in (5,6,7)

Scope of this project:

While this proposal showcases an end-to-end solution of "what good looks like", we will limit the scope of this project to key areas of data science to meet the requirements of this course. The scope of this project will be limited to the retrieval of articles, the implementation of a ranking algorithm if needed, the investigation, development and implementation of multiple agent-based frameworks, LLM models, including exploring various prompt-engineering techniques in an effort to maximize the accuracy and quality of the final summaries. We will use metrics such as ROUGE, BLEU, METEOR, LLM-as-a-judge, as well as human feedback to gauge the quality of the summaries. The results will be showcased to users via Streamlit for this specific course. The remaining elements relating to AWS and the full implementation will be considered stretch goals in an effort to deploy the models online.

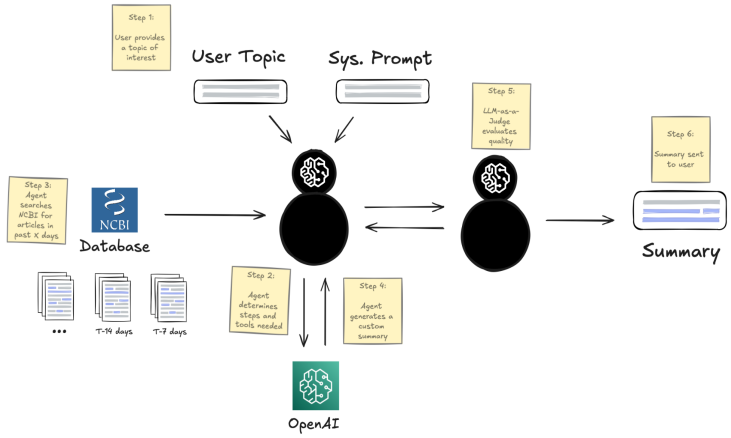


Figure 11: A graphical depiction of the end-to-end plan that is in scope for this project showing the summary agent, and judge agent, with the summary itself as a result.

Challenges and Risks

To manage challenges and risks, I implemented an Agile approach to risk management using a structured method known as the ROAM framework, categorizing challenges into Resolved, Owned, Accepted, and Mitigated areas. Concerns about database access and the availability of critical metadata such as dates were initially a risk but have since been **Resolved** by switching to a more reliable API with the required metadata. Data privacy concerns have been **Mitigated** by deciding to make the package open source, allowing organizations to implement it within their AWS accounts, ensuring compliance and data security. Challenges related to gathering sufficient user feedback remain **Owned**, as time constraints may limit the development of a fine-tuning mechanism, though efforts will be made to address this where possible. The potential for external API throttle limits has been **Accepted** as an unavoidable risk, given the dependency we have on third-party services. To address the **Mitigated** risk of reliance on a single LLM, OpenAI has been supplemented with an additional model from HuggingFace to ensure continuity and flexibility. Finally, the summarization accuracy has been **Mitigated** by incorporating expert evaluation, where subsets of summaries are ranked based on accuracy, relevance, coherence, completeness, and fluency, leveraging domain expertise to enhance quality. This systematic risk management strategy ensures the platform's robustness and reliability.

Model Deployment

To make the agent immediately usable, we packaged the core logic into the scirad Python module and pushed it to GitHub at https://github.com/alkhalifas/scirad. Although we haven't published it to PyPI yet, users can install it in one step via cloning the repo, and importing the agent. We also built an interactive Streamlit demo (app.py) that guides users through keyword extraction, PubMed searching, ranking, and summarization. Ultimately, all they need to do is clone the repo, install the requirements, and run streamlit run app.py. Finally, the GitHub repository itself contains a comprehensive README with quick-start instructions so anyone can explore, test, and extend the application with minimal friction. In addition, a Flask application was also prepared to make the capability available via API. The instructions for this are available on the Github readme file.

Lessons Learned

Throughout the project, I discovered that some models come with a significantly higher cost, making it important to balance performance with budget. This was not generally considered until after running several hundred experiments and seeing the costs begin to multiply. In our experiments, GPT-3.5-Turbo was identified as a cost-effective option that delivers excellent performance relative to its price. Exploration into multi-agent frameworks revealed that they are far more complex than originally anticipated. Despite successfully creating a first version of a multi-agent system, its overall quality did not improve further that of the single-agent setup, indicating that the framework in this space remains relatively new and immature. Although the multiagent path is not yet complete, I will continue to make progress here and see what options there are.

Establishing clear objectives and maintaining a consistent, end-to-end perspective proved to be quite essential. Rather than treating each feature in isolation, I continuously evaluated how new capabilities would mesh with the platform's overall goals thus delivering concise, accurate, and actionable digests to researchers. This systems-level thinking ensured that development efforts stayed aligned with user needs and prevented feature creep. Equally pivotal was the iterative feedback loop: regular check-ins with my professor and peers highlighted hidden assumptions, helped prioritize critical functionalities, and refined the scope to a practical, achievable scale within the academic term. Their guidance transformed vague ideas into concrete deliverables, accelerating progress and enhancing overall project quality.

Venturing into multi-agent frameworks revealed another layer of complexity. While single-agent pipelines where one LLM orchestrates retrieval, aggregation, and summarization proved reliable and straightforward to optimize, multi-agent architectures introduced challenges in coordination, context-sharing, and error propagation. Despite successfully deploying an initial multi-agent prototype, its end-to-end performance failed to outperform the simpler, single-agent configuration. This outcome reflects the changing state of multi-agent best practices: although foundational agentic AI concepts are well-understood, the rapid pace of innovation means that robust, production-ready frameworks are still emerging and changing. In the end, cost was also a factor given that the single agent consumed less tokens, yet achieved a high score related to its multiagent counterpart.

Conclusion and Future Direction

In summary, the results of this project demonstrate a robust and feasible pipeline that can be used to deliver key insights into scientific areas of research in an effective and pragmatic way to combat information overload using the 80:20 rule. We showcased our agentic framework and optimized it using experimental studies to maximize a metric using an LLM-as-a-judge. Although these experiments validated the single-agent approach across both platforms, our initial hypothesis into multi-agent architectures did not yield the performance improvements over the simpler pipelines as anticipated as we pivoted from that during the mid-year update. In future work, we hope to revisit the multi-agent framework by experimenting with advanced orchestration tools, refining inter-agent communication protocols and leveraging emerging best practices in hierarchical planning. In addition, these agentic capabilities were developed with the "full picture" in mind. We were mindful of not only the core capabilities, but also how the capabilities could one day be deployed in the broader context using cloud providers such as AWS. A future objective could be deploying these capabilities there to enable a broader use of the tool.

References:

1. Zhang, J., Zhao, Y., Saleh, M., & Liu, P. PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization. arXiv. **2019**.
2. Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv. **2020**.
3. Hernandez-Leal, P., Kartal, B., & Taylor, M. E. A Survey of Multi-Agent Reinforcement Learning. arXiv. **2019**.
4. Bawden, D., & Robinson, L. The Dark Side of Information: Overload, Anxiety and Other Paradoxes and Pathologies. Journal of Information Science. **2009**.
5. Jackson, T., Dawson, R., Wilson, D., & Riddle, D. Overcoming Information Overload in the Digital Age: A Framework for Decision-Making. International Journal of Information Management. **2017**.
6. Chen, H., Hsieh, Y., & Wang, S. Reducing Information Overload in Healthcare: A Systematic Review. Journal of Medical Internet Research. **2005**.
7. OpenAI. Learning to Summarize with Human Feedback. OpenAI Blog. **2021**.
8. OpenAI. ChatGPT: Optimizing Language Models for Dialogue. OpenAI Blog. **2022**.
9. Fu, Y., et al. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv. **2022**.
10. Schick, T., et al. Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv. **2023**.
11. LangChain Team. LangChain: A Framework for LLM-Powered Applications. LangChain Blog. **2023**.
12. LangChain Team. Introducing LangSmith: Observability for LLM Applications. LangChain Blog. **2023**.
13. Park, J., et al. Generative Agents: Interactive Simulacra of Human Behavior. Conference Publication. **2023**.
14. Kim, S., et al. Big Data and Information Overload: A Survey. arXiv. **2020**.