Metadata-Driven Hybrid Search for Enhanced TAU Online Tutor

Orpaz Ashkenazy, Guy Weintraub, Yaar Koren
Workshop On Applying Large Language Models to Education
Tel-Aviv University, February 2025

Abstract

This paper presents an enhancement to TAU Online Tutor (OT), an AI-powered educational assistant designed to support the university students, by integrating metadata-based querying into a Retrieval-Augmented Generation (RAG) framework. Our approach enhances both retrieval and answer generation through metadata-driven capabilities by introducing an additional LLM step, along with architectural functions and logic, to parse and analyze user queries, refining downstream retrieval and generation processes. The query parsing mechanism, referred to as self-querying, employs a LLM to extract metadata directly from user queries, enabling metadata-based filtering in the vector database. Further analysis of the parsed query enabled the incorporation of adaptive strategies, optimizing both retrieval parameters and the answer generation process. We evaluated the effectiveness of our work on a benchmark dataset comprising both standard queries and metadata-specific queries. Overall, our findings contribute to the advancement of OT by providing a scalable framework for intelligent, adaptive, metadata-based content retrieval and question answering.

1 Introduction

Online Tutor (OT) is an in-house project at TAU, designed as a specialized AI teaching assistant for university courses. The system, powered by a large language model (LLM), assists students by answering queries using a pre-loaded corpus of course materials of various types, such as lecture transcripts, notes, and textbooks. A pilot study across seven courses demonstrated its effectiveness. However, a key limitation has been identified: OT lacks the ability to process metadata-related queries. Specifically, it cannot recognize references to lecture numbers, distinguish between source types, or retrieve information based on temporal markers like "last week's recitation."

To understand this limitation, it is important to examine **Retrieval-Augmented Generation (RAG)**, the framework used in OT. RAG enhances LLMs by incorporating an external knowledge base to provide contextual information[1][2]. The process consists of two stages: retrieval— where the system identifies top-k relevant documents from a vector database (DB) — and generation, where an LLM formulates a response based on both the query and retrieved context. In OT's current implementation, all course materials are stored as embeddings in a vector DB, and retrieval is performed using cosine similarity search. However, no metadata is stored or retrieved, making it impossible for the system to respond adequately to metadata-based queries.

To address this issue, the OT team tasked our team with developing a solution, focusing on the use case of students requesting summaries of classes. Our work aims to give the OT system the ability to process metadata-based queries, both for the specific use cases we were asked to address and as a proof of concept for this framework, enabling more detailed and intelligent questions answering in the future.

2 Related work

Multiple ideas for improving content retrieval and summary have served as a conceptual foundation for developing the ideas of our project.

A recent approach for summary retrieval is RAPTOR[3], a method of recursive summarization in a tree structure, which retrieves relevant tree nodes, whether summaries at varying levels of abstraction or text chunks. While this approach offers an overview of the corpus as a whole, and accounts for distant inter-dependencies within it, we opted not to adopt this method for several reasons. Firstly, our primary objective was to integrate metadata-based querying into the existing retrieval system, RAG. Implementing RAPTOR would have necessitated modifications to the RAG system without directly addressing the metadata component. Additionally, in our case, the corpus is inherently "clustered" by lecture number, date, and source type, reduces the need for an external clustering algorithm. While RAPTOR could potentially enhance the overall comprehension of the entire course, our primary use cases focus on queries related to the inherent course metadata structure.

Mombaerts et al.[4] introduced a data-centric approach in which, instead of encoding the original documents, synthetic question-answer (Q&A) pairs are encoded. Moreover, the approach uses metadata, that is generated by LLM, for filtering purposes and introduces the concept of using Meta Knowledge Summaries (MK summaries), which summarize the key for a given filter. At inference time, given a user query and a set of pre-selected metadata of interests, it first retrieves the corresponding pre-computed MK Summary. The summary is then used to restrict the retrieval to a specific subset of the database. Finally, it identifies the best-matching synthetic question, retrieving the associated Q&A pairs along with the original document title.

Our methodology similarly incorporates metadata-based summaries. In our case, the summaries themselves are directly embedded in the vector database, and are based on the source type (e.g., lecture or recitation) in order to preserve the materials' structural integrity. Another key difference is that our approach maintains the traditional method of encoding the original documents rather than exclusively encoding Q&A pairs. This decision was made because we wanted to preserve the existing RAG structure and to ensure that no information from the original documents is lost. Furthermore, while their approach assumes that the metadata is explicitly provided, separately from the user query, our methodology is designed to extract metadata directly from the raw query.

Susnjak et al.[5] employed a RAG architecture to enhance an LLM's capability in the final stage of Systematic Literature Reviews (SLRs), focusing on knowledge synthesis. A key innovation they introduced was metadata filtering, structuring their vector database with two attributes: content ("text") and filterable metadata fields ("source"). During retrieval, they leveraged LangChain's self-querying retriever, which uses an additional LLM step to extract metadata fields from user queries. This self-querying retrieval approach, specifically the LangChain query parser, serves as the foundation for our enhanced system, as detailed in the following sections.

3 Methods

3.1 The Setup

3.1.1 Data Setup

The data provided to us by the OT team was materials from the course *Introduction to Hardware for Digital Science for High-Tech*[6], which was selected due to its stability and strong performance in the pilot study. The original data consists of English transcripts of the lectures, and separate SRT files with corresponding timestamps.

Since our goal was to test metadata related queries, we added additional data to facilitate our testing. The first addition was the course recitations, which we obtained by extracting and translating (using ChatGPT-4o) formal class notes of the recitation. The second addition was a summary for each lecture, as described in section 3.2.

Finally, for the purpose of our proof of concept, we created a metadata table [see attached file *metadata_table.csv*] containing the relevant fields we aimed to explore in our use cases:

- **source_type** - Type of content (e.g., lecture transcript, recitation, or summary).
- **date** - The date when the class was held.
- **number** - The class number, ranging from 1 to 13.

3.1.2 System Setup

The pipeline we worked on was a simplified version of the OT system, referred to as Simple RAG (SR). The LLM used in this system is GPT-4o-mini, the same model used in the OT System. However, The OT system differs from Simple RAG in several optimizations that our simplified system lacks. Notably, neither system is restricted to relying solely on retrieved course documents to generate responses.

3.1.3 Benchmark

To evaluate the integrity of the SR and our enhancements, we employed two benchmarks. The first, the **Original Benchmark** (Appendix A), consists of 34 non-metadata-related queries provided by the OT team. These queries are known to be stable and correctly answered in the original system.

As an initial sanity check for our SR sandbox, we used this benchmark to assess whether the Simple RAG functioned as expected with the course data. While the SR correctly answered most queries, it generated longer responses and twice as many references. It provided an incorrect answer for one query (4), encountered an error on another (33), and generated partially correct answers, that did not contain the same keywords as the OT, for eight queries (10–15, 18, 26). We subsequently applied this benchmark to our updated versions to ensure that the new pipeline did not compromise the original system's answer generation capabilities.

The second benchmark, the **Metadata-Based Benchmark** (Appendix B), was designed specifically to evaluate the system's ability to handle metadata-related use cases. This benchmark comprises 31 queries, with a significant portion focused on summaries, a key area of interest in our project.

3.1.4 Evaluation

Evaluating LLM-generated answers is inherently complex. To ensure a structured assessment, we conducted a manual evaluation of the answers. To mitigate human subjectivity, the evaluation was based on discrete measurements of correctness and the presence of key terms:

- **0** – Incorrect answer. In the case of summaries, refers to responses that contain hallucinations or introduce unrelated subjects not present in the required class. This score also applies to cases where no answer is provided, such as errors or responses indicating a lack of knowledge, when the query can be answered.
- **1** – Partially correct answer. The response is not misleading or incorrect but fails to fully address the query or lacks important keywords.
- **2** – Correct answer. Contains all keywords, and refers only to the requested materials.

To further monitor progress, we incorporated intermediate evaluation steps within the pipeline, such as verifying that the updated retrieval process returns the correct documents.

Regarding efficiency and/or complexity, this paper does not address these aspects, primarily because it serves as a proof of concept within a simplified system, and also because the time required to generate an answer for a query in all of our versions is comparable to that of SR (and OT).

In the remaining part of this section, we chronicle the gradual enhancements made to our framework, culminating in the final Version (reffered to as version 5.3, described in Figure 2). Each subsection reflects a stage in our development process, highlighting the decisions and refinements informed by intermediate outcomes. By following the sequence of upgrades—ranging from basic additions to self-querying retrieval and beyond—we clarify how we arrived at a robust, metadata-aware system and illuminate the rationale behind each step taken along the way.

3.2 The Naïve Solution - Adding Summaries To Database

Our initial approach involved a naïve method of generating summaries for each lecture. In cases where lecture transcripts were split across two files, we treated them as a single unit. Summaries were created with the prompt attached in Appendix C, using ChatGPT's user interface (UI). Each summary that was generated consists approximately 5–7 text chunks, where each chunk is 1,000 characters long, with an overlap of 200 characters between consecutive chunks.

While this method provided some improvement in answering summarization-related queries, it still retrieved irrelevant materials and lacked the precision needed to focus on the most relevant content. Additionally, it failed when handling queries that referenced a range of lectures or used synonyms for "lecture" (e.g., "class" or "lesson"), often producing completely incorrect responses. Furthermore, for queries beyond simple lecture summarization, the naïve approach did not yield any notable improvements.

3.3 Self-Querying Retriever

After observing that merely adding summaries to the materials of the existing retrieval architecture was insufficient, we adopted a new approach based on the concept of self-querying retrieval[7]. In self-querying retrieval, the user query is parsed into two components:

i. **Content Query**: The part of the query representing the subject or topic (for example, 'T-flip flop').

ii. **Filter**: The extracted metadata fields and their corresponding comparators (for example, 'before lesson 5' translates into a comparator $lt with value '5').

Combined, these form the **Translated Query**, which is subsequently used for a hybrid retrieval process. First, the system filters documents in the vector database according to the metadata-based filter. Then, it performs a similarity search on the filtered subset using the vectorized Content Query to retrieve the top-k documents.

To implement self-querying retrieval, we introduced two fundamental changes:

a. **Vector DataBase Construction:** We created a new vector DB that stores both content embeddings and metadata fields for each document. This metadata-enabled structure supports filtering during the hybrid retrieval step. We chose Pinecone[1] as our vector store due to its seamless integration with the LangChain framework and its support for operators that meet our filtering needs. In the updated setup, the same content embeddings as before are used, but each vector now includes additional metadata read from our pre-made metadata table, as mentioned in section 3.1.1.

b. **Query Parser Definition:** We employed a parser to translate the user's natural-language query into the corresponding Content Query and Filter (the Translated Query). For the query parsing, we utilized LangChain's *QueryConstructor* package along with the *PineconeTranslator*, which formats the Filter for compatibility with the Pinecone vector store. As mentioned, we focused on three relevant metadata fields for our use cases: "number", "source_type", "date". The query parser (i.e., query constructor) was tasked with identifying the relevant metadata fields and their valid comparators. To achieve this, we iteratively refined the parameters passed to it and employed few-shot learning to demonstrate the expected output format[8][9]. As a result, the parser consistently handled both our benchmark queries and specialized test scenarios.

Figure 1 illustrates some examples of how user queries are parsed and formatted.

Combining this self-querying retrieval pipeline with new content summaries resulted in what we refer to as *Version 2* of the system. The subsequent versions introduced further enhancements, described in the following subsections.

[1]https://www.pinecone.io

3.4 Functional Additions and Boolean Tagging of Summaries

Our following improvements addressed more technical and logical aspects of the hybrid retrieval workflow:

- **Boolean Tagging of Summaries:** We introduced a new boolean metadata field, *is_summary*, to tag every summary document. This allows the system to match between a source document (e.g., a lecture transcript) and its corresponding summary (Figure 1, example 1).

- **Date Field Handling in Pinecone:** Since Pinecone does not directly support Python's *datetime* field type, we stored dates as epoch time (an integer). The query constructor was taught to produce date-like filters as strings, and we added utility functions to convert between user-friendly date strings and epoch time as needed (Figure 1, example 2). Furthermore, we augmented the system with the ability to interpret date references relative to the current date (e.g., 'the recitation last week').

With these additions, the system can filter documents more accurately, differentiating raw source content from summaries and bridging the gap between user-friendly date formats and Pinecone's integer-based metadata constraints.

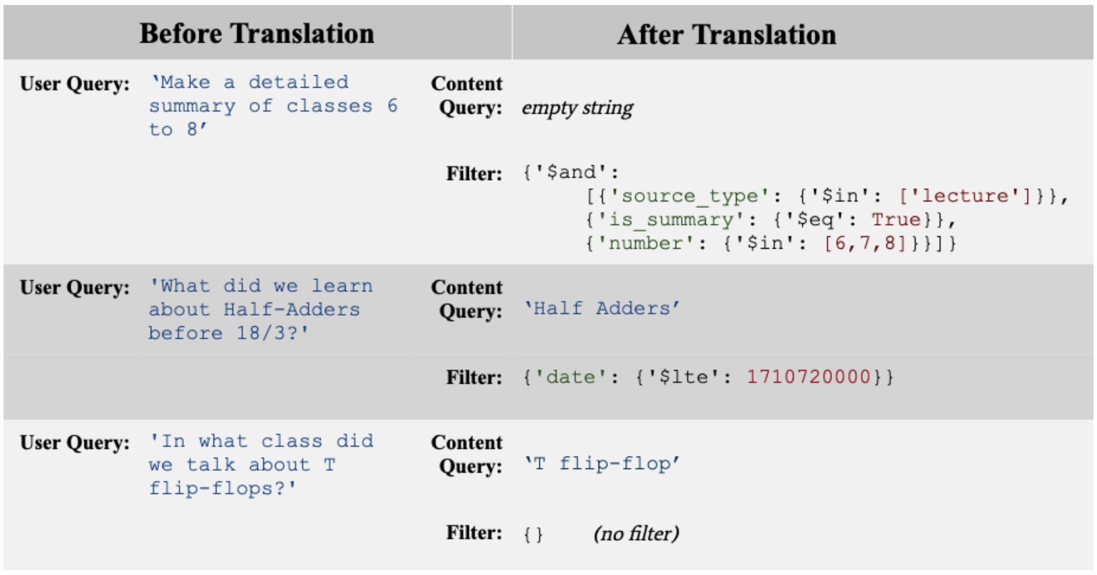
3.5 Query Analyzer — A Smarter Approach

While these additions solved some technical issues, additional problems emerged during testing, mainly:

- **Fixed context size**: A constant k (the number of documents retrieved) was chosen as a compromise between coverage and efficiency. However, a single fixed value sometimes led to incomplete answers for broad questions (e.g., "summarize lectures 6–8").

- **Unpredictable User Queries**: Relying solely on explicit keywords (like "summary" or "general topic") made it difficult to detect high-level requests such as "What did we talk about in lecture 7?" if the query did not directly say "summarize."

- **Metadata in the Generation**: The answer generation stage still used the original user query, even if it contained metadata-related



Before Translation		After Translation	
User Query:	`'Make a detailed summary of classes 6 to 8'`	**Content Query:**	*empty string*
		Filter:	`{'$and': [{'source_type': {'$in': ['lecture']}}, {'is_summary': {'$eq': True}}, {'number': {'$in': [6,7,8]}}]}`
User Query:	`'What did we learn about Half-Adders before 18/3?'`	**Content Query:**	`'Half Adders'`
		Filter:	`{'date': {'$lte': 1710720000}}`
User Query:	`'In what class did we talk about T flip-flops?'`	**Content Query:**	`'T flip-flop'`
		Filter:	`{}` *(no filter)*

Figure 1: Query Translation Examples. User queries are parsed and relevant metadata criteria appear in the created filter

prompts (e.g., date filters). This sometimes produced confusion in the final response.

To address these issues more holistically, we developed an additional module reffered to as the **Query Analyzer** (colored red in Figure 2). Building on the knowledge extracted during parsing, the Query Analyzer can read and modify the translated query, adjusting pipeline parameters such as k, the filter, and the prompt ultimately sent to the Large Language Model (LLM). Figure 3 provides a schematic of this process. Our enhancements include:

- **Adaptive k Selection**: The system changes k based on query type [see the function *choose_k_by_translated_query* in the attached code file *"Final code - Version 5.3.ipynb"*]. For broader queries, a larger k is used to ensure adequate coverage.

- **Summary Inference**: If a user's Content Query is empty (indicating no specific topic), the Query Analyzer adds is_summary = true to the filter to retrieve high-level summaries by default.

- **Adapted Query String**: Before calling the LLM, the Query Analyzer replaces the original user query with an adapted query that includes a short prefix describing the retrieved context (e.g., "These are summaries of lectures 5, 6, 7: user query"). This helps the LLM generate a more cohesive answer.

3.6 Informed Response Generation

All the adaptations described so far were based on knowledge derived from parsing and analyzing the user query. The following changes, however, were made by leveraging information gathered from the retrieval process itself. The remaining drawbacks we faced:

- **Referencing Multiple Sources Clearly**: When asked to summarize multiple classes (e.g., 'summarize lectures 6–8'), the system provided correct content but with a wrong delineation of which material corresponded to which class.

- **Identifying a Specific Class Containing a Topic ("Reversed Queries")**: Queries such as 'In which class did we talk about T flip-flops?' remained difficult to answer because the retrieved context did not explicitly indicate the class numbers or dates.

To solve these issues, we changed how context is packaged for the answer generation step. Instead of passing only text content, the system now formats the context into a single string containing both the content and the metadata for each retrieved document. This makes the answer-generating LLM explicitly aware of which content corresponds to which source.

Additionally, we introduced a safeguard to prevent unnecessary LLM calls when no relevant documents are found (e.g., when requesting a non-

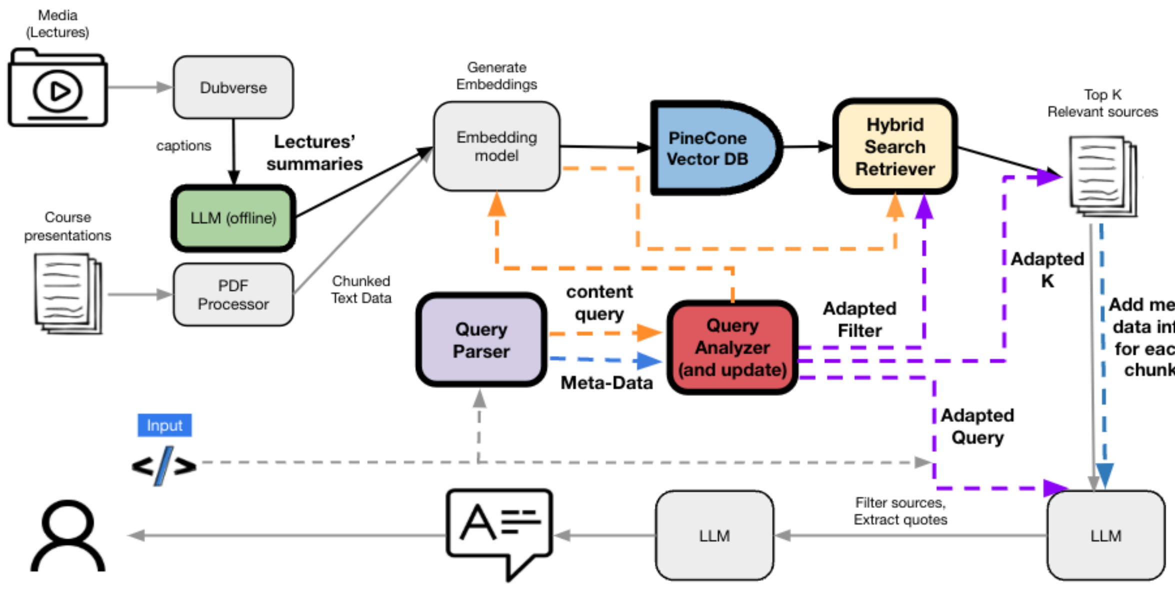


Figure 2: **Final Architecture Scheme**. New additions are in color.
• Green: Summaries creation and addition to database.
• Blue and Yellow: *Pinecone* and *Langchain* based vector DB, allowing for metadata filtering in hybrid search.
• Purple: **Query Parser** - Translation and formatting.
• Red: **Query Analyzer** - A set of functions to read and update the downstream parameters: Adapted Filter, Adapted k, Adapted Query.
The arrow on the right represents the **Context string**, formatted together with the corresponding metadata info.

existent date or an out-of-range class number). Before generation, the pipeline now verifies whether any documents were retrieved. If none are found, the system bypasses answer generation and informs the user that no material matches the request, displaying the applied filter.

Figure 2 provides an overview of the final architecture, denoted in subsequent figures Version 5.3.

4 Results

When using LLMs for question answering, each run can produce slightly different responses. To ensure confidence in the results, we executed each query at least three times for every version. All queries—except for two in the original benchmark and two in the metadata benchmark—yielded consistent results across all three runs in every version. For the queries that exhibited inconsistencies, we ran each ten times per version and assigned scores based on the majority outcome.

4.1 Original Benchmark

For the complete results for the Original Benchmark - see attached file "*Original Benchmark Answers - Version 5.3.xlsx*".

First, as mentioned earlier, our approach aims to preserve the integrity of the original benchmark, which served as a positive control. Regarding incorrect answers, both the SR and our updated version provided one incorrect answer (query 4). Additionally, SR raised an error on one query (query 33), whereas our version answered it correctly.

For partially correct answers, SR and our version demonstrated similar inaccuracies for queries 10–14. Beyond these, there were cases where SR provided a correct response that did not exactly match the expected answer, while our approach aligned with expectations, and vice versa.

A notable issue with our updated system was its tendency to return "The answer is not in the context" for queries where SR provided correct responses. This behavior occurred in five queries (8, 25, 31, 32, 34). In three of these cases, the response was consistent across 4 runs. However, in the remaining two (8, 25), inconsistencies were observed, prompting us to extend the evaluation to ten runs. In those additional runs, the response The answer is not in the context " appeared in only 10% of cases for one query and 20% for the other, while the majority of responses were correct. It is important to highlight that these queries involved subjects not explicitly present in the context, requiring inferred knowledge. This issue, along with potential explanations, is further addressed in the discussion section.

Overall, for the original benchmark, our enhanced pipeline did not introduce significant deviations, as its results closely aligned with those of the SR, as shown in the Original Benchmark Scores Table (Appendix F).

4.2 Metadata-Based Benchmark

For the complete results for our metadata-based benchmark—including Translated Queries, retrieved documents, and final answers—see attached file "*Metadata-Based Benchmark Answers - Version 5.3.xlsx*".

As shown in the Metadata-Based Benchmark Scores Table (Appendix E), our final version succeeded in answering correctly 26 out of 31 queries, demonstrating a significant enhancement in performance across the evaluated benchmarks. In the following section, we elaborate on some of the notable and interesting cases that illustrate these improvements.

Regarding the queries about lecture summaries, while only including summary documents slightly improved the answers (Figure 3), the results were still not optimal, indicating the limitations of this naïve solution. By combining the summaries with the self-query methodology (Version 2 and forward), the performance improved significantly. The primary challenge in this category laid in addressing multiple lectures summarized together (query 4). In this case, the LLM incorrectly segments the subject list between the lectures. Only in the final version, where metadata was integrated into the LLM's context, the responses were entirely accurate.

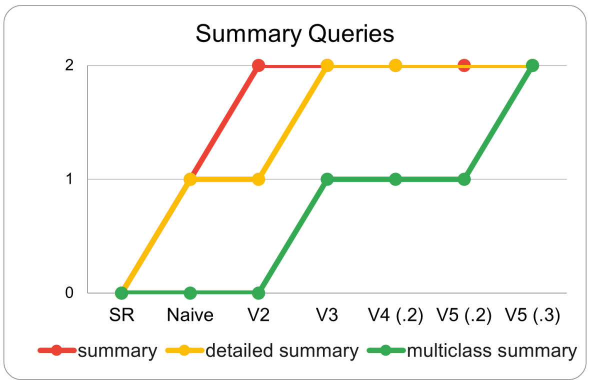


Figure 3: Improvements in responses to summary-related queries across versions.

Regarding queries about topics not covered in the specific lecture, the nature of the improvements varied depending on whether the topic was part of the course syllabus (but covered in a different lecture) or completely outside the syllabus, as shown in Figure 4. Notably, these are the two queries that exhibited inconsistencies across runs, requiring additional evaluations to determine their final scores.

For topics from the syllabus that were covered in other lectures, the SR retrieved files related to the topic but not from the specific queried lecture, leading to incorrect answers. As a result of our improvements, the LLM enhanced its contextual retrieval and correctly identified that the queried lecture did not cover the topic.

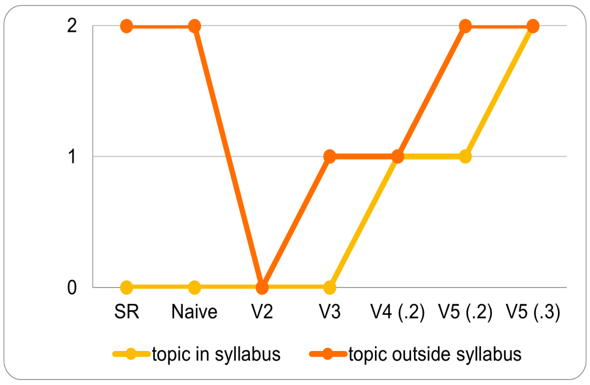


Figure 4: Responses' quality for queries regarding topics not covered in the specific lecture in which they were asked.

For topics outside the course syllabus, the SR responded correctly due to the absence of relevant files. However, our early versions led to hallucinated responses. After the further refinements mentioned above, the LLM improved and consistently provided accurate answers.

Other interesting results are the improvements regarding queries related to dates, queries regarding which class covers a specific topic ("Revered Queries"), and queries spanning multiple lectures. All of these enhancements introduce new capabilities that were previously absent in the OT system.

However, certain categories of queries remain unresolved and exhibit the same issues both before and after the applied enhancements (e.g., timestamp queries). These unresolved challenges are further discussed in section 6.

5 Discussion

In conclusion, our methodical updates and results demonstrate how the OT system can effectively leverage metadata awareness at both the retrieval and generation phases.

We introduced a self-querying retrieval mechanism, enhanced by an adaptive pipeline, to create more targeted and coherent responses. By integrating a Query Analyzer module and formatting the retrieved context with explicit metadata, we demonstrated how the OT RAG system can effectively handle both broad and specific queries. These updates form a strong proof of concept for leveraging metadata-awareness at each stage of retrieval-augmented generation to improve accuracy and clarity in answers.

Despite these improvements, certain challenges persist, including the broader issue that extend beyond the scope of metadata capabilities, which is balancing answer completeness with retrieval efficiency. To ensure the system did not respond to queries beyond the retrieved context, we applied prompt engineering modifications (Appendix D). While these adjustments improved reliability by restricting answers to the retrieved content, they occasionally led to the omission of correct responses not explicitly stated in the provided context. This underscores a broader limitation of RAG systems, where achieving an optimal trade-off between precision and recall remains a persistent challenge.

Overall, our approach demonstrated the feasibility of integrating metadata-based querying into a RAG framework, though further development is required to handle more complex queries.

6 Future work

Building upon the current work, if we had more time, we would focus on extending the scope of the project through several directions of future development and enhancement:

I **Extending Metadata Field Features**: future efforts can focus on expanding the metadata schema to include additional fields such as timestamps, professors' names, and other relevant attributes. This enrichment of metadata will provide a more detailed and contextual view of the course material, enhancing searchability and organization.

II **"Double-Headed" Filtering**: an approach to filtering that can be implemented to enable the combination of multiple filtering criteria. For instance, scenarios requiring a simultaneous focus on "lecture 3 and recitation 4" or a division of content based on dates, such as "April 20 and the rest of the course," should be supported. This feature will facilitate more nuanced and flexible data retrieval tailored to specific needs.

III **Beyond Metadata**: Beyond merely expanding metadata fields, future work can build on our advanced query-parsing and analysis capabilities to organize course materials more effectively based on their underlying content structure. The aim is to create a richer filtering process that will include data such as topic tags and material types (proofs, theorems, etc.). This approach will support a deeper understanding and more effective exploration of the course material by incorporating content related insights into the metadata.

IV **Improving Summaries**: ChatGPT's API exhibits certain limitations when compared to its user interface (UI), particularly it generates less comprehensive and detailed summaries [see our tries in the attached file *"create_summeries_code.ipynb"*]. Although our project did not focus primarily on summary creation, these limitations impact the overall response quality. This highlights the importance of future efforts focused on improving the summarization capabilities of the API. Potential enhancement strategies include refining the prompt, tuning model parameters (such as temperature), or employing advanced techniques, such as iterative refinement[10] to produce more robust and elaborate summaries, that can be created automatically offline.

V **Evaluation on Additional Courses:** As described above, our work was conducted using materials from a single course. Because the project primarily focused on metadata-related capabilities—using only minimal content examples—we believe this architecture can generalize to various types of courses. However, it remains essential to validate this assumption by testing on additional course materials.

These directions aim to enhance the usability and analytical capabilities of the system, providing both broader and deeper engagement with the content.

7 References

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Kuttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, NeurIPS 2020.

[2] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang: Retrieval-Augmented Language Model Pre-Training, ICML 2020.

[3] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, Christopher D. Manning: RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval, 2024.

[4] Laurent Mombaerts, Terry Ding, Adi Banerjee, Florian Felice, Jonathan Taws, Tarik Borogovac: Meta Knowledge for Retrieval Augmented Large Language Models, 2024.

[5] T. Susnjak, P. Hwang, N. H. Reyes, A. L. Barczak, T. R. McIntosh, and S. Ranathunga: Automating Research Synthesis with Domain-Specific Large Language Model Fine-Tuning, ACM Transactions on Knowledge Discovery from Data, 2024.

[6] Introduction to Hardware, for Digital Science for High-Tech. YouTube, 2020.

[7] LangChain: Self-Querying Retriever Documentation, 2024.

[8] van Oudenhove, Lore: Enhancing RAG Performance with Metadata: The Power of Self-Query Retrievers, 2024. Accessed: 2024-02-04.

[9] Izaguirre, Ed: How to Build a RAG System with a Self-Querying Retriever in LangChain, 2024. Accessed: 2024-02-04.

[10] LangChain: How to summarize text through iterative refinement, 2024.

8 Appendix

A Original Benchmark

1. Can a Full Adder (FA) be used as a Half Adder (HA), and why?
2. Is it always possible to replace OR with XOR as we did in the development of the FA?
3. Why is there a need to replace OR with XOR?
4. Is Look Ahead Carry also dependent on the delay of each component in sequence?
5. Why do we need Look Ahead Carry?
6. What is the meaning of $I_0, \dots, I_4$ in a multiplexer?
7. Why do we need a MUX at all? What are its uses?
8. When you implemented a logical circuit using a MUX, how did you decide on the free variable?
9. What happens if we input 1-1 into an SR-Latch?
10. With a type D-Latch, it seems like the component does nothing. Why?
11. What is the advantage of a JK-Latch over a T-Latch?
12. Why did you change the inverters to another type of gates (NORs or NANDs)?
13. Is there a difference between SR-Latch based on NORs vs NANDs?
14. Which kind of SR-Latch will we use in our class?
15. How does the gated SR-Latch work? / What is a gated SR-Latch?
16. What is the main problem with the SR-Latch design?
17. What is a D-Latch? / How is the D-Latch built?
18. How do we find the Excitation Table?
19. When to use the Characteristic Table and when to use the Excitation Table?
20. What is a JK-Latch?
21. What are the differences between $(R-1)'$s complement and R's complement?
22. Can you explain to me deeply about Gray code and what is the purpose of it?
23. Can you explain why NOR is functionally complete?
24. What should I do when I have a sum and I have a 'carry' in the leftmost digits, that is, I have no one to transfer the 'carry' to? For example: $1 + 1$ in the leftmost digits sum.
25. Hi, is NAND a complete operating system?
26. How to convert fractions to a decimal base?
27. How to convert hexadecimal base to decimal?
28. What is Boolean algebra?
29. What is the meaning of "the arithmetic of SM doesn't work"?
30. What is a minterm?
31. Can you show me the proof that $f(x, y, z) = x'y'z' + xz + yz$ is functionally complete?
32. How to subtract in base 8?
33. How to perform subtraction in the 1's complement method?
34. I need to add and subtract pairs of numbers in base 8, without converting to decimal. How can I do it?

B Metadata-based Benchmark

Queries about summaries and topics of lectures or recitation

1. What did we prove in lecture 5?
2. Please summarize the 5th class.
3. Make a detailed summary of class 5.
4. Make a detailed summary of classes 6 to 8.

5. What is the topic of lecture 8?
6. What is the topic of class 9?
7. What is the topic of lesson 9?
8. What is the topic of lecture 9?
9. What was class 10 about?
10. What was lesson 10 about?
11. What was lecture 10 about?
12. What did we learn in recitation 4?

Queries about specific topics

13. In what class did we learn about finite state machines?
14. In what class did we talk about the T flip-flop?

Queries about topic in specific lectures

15. What did we learn about ROM memory in lecture 7?
16. What did we learn about ROM memory in lecture 11?
17. What did we learn about Object-Oriented Programming in lecture 11?

Queries about timestamps

18. Summarize the last 15 minutes of lecture 1.
19. In lecture 6, part A, at time 13:33, the teacher said you can exchange the OR gate with an XOR gate. Explain why.
20. Explain what the teacher said in lecture 6, part A, at time 13:33.

Queries about specific dates

21. What did we learn in the lecture on May 2, 2020?
22. What did we learn in the lecture on February 19, 2024?
23. What did we learn in the lecture on March 18?
24. What did we learn in the lecture last week?
25. What did we learn in the recitation last week?

Queries about topic through the whole course

26. In lecture 11 we learned about MOS transistors, how does this combine with what we learned throughout the course?

Queries combining two specific lectures

27. In lesson 2 we saw the problem of two different representations of zero. In lesson 6, we talked about implementing adders and did not address the dual representation problem. Explain.

Queries about material up to a specific lecture

28. Explain what we have learned about timing, including only things we learned up until lecture 8 (included).
29. Explain what we have learned about flip-flops, including only things we learned up until lecture 7 (included).
30. Explain what we have learned about flip-flops, including only things we learned up until lecture 8 (included).

Queries combining lecture and recitation

31. In lecture 4 and in recitation 4, we learned that undefined states in the Karnaugh Map do not matter. Explain.

C Prompt for Creating Lectures' Summaries

Act as a teacher who is a professional summarizer and expert in hardware. Write a detailed, comprehensive, structured and easy-to-understand summary of the provided content.
The title should include 'Summary of Lecture [lecture number]'.
Start with a brief overview that explains the main purpose and structure of the lecture in simple terms.
Organize the summary into clear sections and with well-structured paragraphs and bullet points based on the main topics discussed.
For each section:
1. Elaborate thoroughly on explaining the key concepts, themes, or methodologies. For each subject, explain the subject in depth, including definition and explanation.
2. Include specific and concrete example for every key concept.
3. Highlight connections between different topics discussed, especially where they build upon each other or showcase cause-and-effect relationships.
Make the language professional, concise, and tailored to explain the material clearly to students.
Rely strictly on the provided text, without including external information.
The goal is to provide a summary that is accurate, easy to follow, and helps students quickly understand and review the material, without omitting any important information.
Content to summarize:
[content]

D RAG Prompt Template

Modifications are marked in **bold**.

Original Template in SR

You are an assistant for question-answering tasks. Use the following pieces of retrieved context to answer the question. If you don't know the answer, just say that you don't know. Use three sentences maximum and keep the answer concise.
Question: {user_query}
Context: {retrieved_context}
Answer:

Version 5.3 Template

You are an assistant for question-answering tasks. Use the following pieces of retrieved context to answer the question. If you don't know the answer, **or the answer does not appear in the context**, just say that you don't know or **the answer is not in the context**. Keep the answer concise.
Question: {**adapted_query**}
Context: {**metadata_formatted_context**}
Answer:

E Metadata-Based Benchmark Scores Table

Category	Q no.	Query	Answer Score per Version						
			SR	Naïve	V2	V3	V4 (.2)	V5 (.2)	V5 (.3)
Material type	1	What did we **prove** in lecture 5?	0	0	0	0	0	0	0
Summaries and main topics	2	Please summarize the 5th class	0	1	2	2	2	2	2
	3	Make a detailed summary of class 5	0	1	1	2	2	2	2
	4	Make a detailed summary of classes 6 to 8	0	0	0	1	1	1	2
	5	What is the topic of lecture 8?	0	1	1	1	2	2	2
	6	What is the topic of class 9?	0	1	2	2	2	2	2
	7	What is the topic of lesson 9?	0	1	2	2	2	2	2
	8	What is the topic of lecture 9?	0	1	2	2	2	2	2
	9	What was class 10 about?	0	0	2	1	2	2	2
	10	What was lesson 10 about?	0	0	2	1	2	2	2
	11	What was lecture 10 about?	0	0	2	2	2	2	2
	12	what did we learn in recitation 4?	0	0	2	2	2	2	2
"Reversed Queries"	13	In what class we learned about finish state machine?	1	1	0	0	0	0	2
	14	In what class did we talk about T flip-flop?	1	1	0	0	0	0	2
Topic found\not found in lecture	15	What did we learn about ROM memory in lecture 7?	0	0	0	0	1	1	2
	16	What did we learn about ROM memory in lecture 11?	2	2	2	2	2	2	2
	17	What did we learn about Object-oriented programming in lecture 11?	2	2	0	1	1	2	2
Timestamps	18	Summarize the last 15 minutes of lecture 1	0	0	0	0	0	0	0
	19	In lecture 6, part a, at time 13:33, the teacher said you can exchange the Or gate in a Xor gate. Explain why.	2	0	2	2	2	2	2
	20	Explain what the teacher said in lecture 6, part a, at time 13:33.	0	0	0	0	0	0	0
Lectures' dates	21	what we learned in lecture in May 2, 2020?	0	0	0	2	2	2	2
	22	what we learned in lecture in February 19, 2024?	0	0	0	2	2	2	2
	23	what we learned in lecture in 18.3?	0	0	0	2	2	2	2
	24	what did we learn in the lecture last week?	0	0	0	2	2	2	2
	25	what did we learn in the recitation last week?	0	0	0	2	2	2	2
"Double-headed" filter	26	In lecture 11 we learned about MOS transistor, how does it combine with what we learned throughout the course?	0	0	0	0	0	0	0
	27	In lesson 2 we saw the problem of two different representations of zero. In lesson 6, we talked about implementing adders, and did not address the dual representation problem. Explain	0	0	0	0	0	0	0
Several lectures (up until)	28	Explain what we did learn about timing, include only things we learned up until lecture 8 (included).	0	0	0	2	2	2	2
	29	Explain what we did learn about flip-flops, include only things we learned up until lecture 7 (included).	0	0	0	2	2	2	2
	30	Explain what we did learn about flip-flops, include only things we learned up until lecture 8 (included).	0	0	0	2	2	2	2
Several source types	31	In lecture 4 and in recitation 4 we learned about undefined states in the Karnaugh Map does not matter, explain	0	2	2	2	2	2	2

Table 1: Each query from the Metadata-based Benchmark was evaluated across all versions, receiving a score ranging from 0 (incorrect response) to 2 (correct and well-formulated response), as described in section 3.1.4

F Original Benchmark Scores Table

Q no.	Query	Answer Score per Version	
		SR	V5 (.3)
1	Can a Full Adder (FA) be used as a Half Adder (HA), any why?	2	2
2	Is it always possible to replace OR with XOR as we did in the development of the FA?	2	2
3	Why is there a need to replace OR with XOR?	2	2
4	Is Look Ahead Carry also dependent on the delay of each component in sequence?	0	0
5	Why do we need Look Ahead Carry?	2	2
6	What is the meaning of I0,...,I4 in a multiplexer?	2	2
7	Why do we need a MUX at all? What are its uses?	2	2
8	When you implemented a logical circuit using a MUX, how did you decide on the free variable?	2	2
9	What happens if we input 1-1 into an SR-Latch?	2	1
10	With a type D-Latch, it seems like the component does nothing. Why?	1	1
11	What is the advantage of a JK-Latch over a T-Latch?	1	1
12	Why did you change the inverters to another type of gates (NORs or NANDs)?	1	1
13	Is there a difference between SR-Latch based on NORs vs NANDs?	1	1
14	Which kind of SR-Latch will we use in our class?	1	1
15	How does the gated SR-Latch work? / What is a gated SR-Latch?	1	2
16	What is the main problem with the SR-Latch design?	2	2
17	What is a D-Latch? / How is the D-Latch built?	2	1
18	How to we find the Excitation Table?	1	2
19	When to use the Characteristic Table and when to use the Excitation Table?	2	2
20	What is a JK-Latch?	2	2
21	What are the differences between (R-1)'s complement and R's complement?	2	2
22	Can you explain to me deeply about gray code and what is the purpose of it?	2	2
23	Can you explain why NOR is functionally complete?	2	2
24	What should I do when I have a sum and I have a 'carry' in the leftmost digits, that is, I have no one to transfer the 'carry' to? For example: 1+1 in the leftmost digits sum.	2	1
25	Hi, is NAND a complete operating system?	2	2
26	How to convert fractions to a decimal base?	1	1
27	How to convert hexadecimal base to decimal?	2	2
28	What is Boolean algebra?	2	2
29	What is the meaning of "the aritmetica" of SM doesn't work?	2	2
30	What is minterm?	2	2
31	Can you show me the proof that f(x,y,z)=x'y'z' + xz + yz is functionally complete?	2	0
32	How to subtract in base 8?	2	0
33	How to perform subtraction in the 1's complement method?	0	2
34	I need to add and subtract pairs of numbers in base 8, without converting to decimal. How can I do it?	2	0

Table 2: Each query from the Original Benchmark was evaluated on both the SR and our final version (5.3), receiving a score ranging from 0 (incorrect response) to 2 (correct and well-formulated response), as described in section 3.1.4