JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

RAG-BASED AI TEACHING ASSISTANT FOR LECTURE VIDEO QUESTION ANSWERING WITH TIMESTAMP-GROUNDED RESPONSES

Whisper, BGE-M3 and Llama 3.2 for answers that point to the right moment in a lecture

Prathamesh Kate, Ishwari Muley, Tanmay Ozarkar, Ayush Sapkal, Dr. Rahul Kale

Final Year Student, Final Year Student, Final Year Student, Final Year Student, Assistant Professor
[1-5]Department of Artificial Intelligence and Data Science,
Dr. D. Y. Patil College of Engineering and Innovation, Varale, Talegaon, Pune, India

Abstract : Students who miss a point in a recorded lecture usually have to scrub through the video to find it again, and a general chatbot answers from its own training rather than from the course. This paper describes a retrieval-augmented generation (RAG) teaching assistant that turns recorded lectures into a searchable knowledge base in which every piece of text keeps its position in the video. Audio is extracted from each lecture and transcribed with Whisper, the transcript is cut into chunks that carry the video name and start and end times, and each chunk is stored as JSON and embedded with BGE-M3. A question is embedded the same way, the closest chunks are found by cosine similarity, and Llama 3.2 writes an answer from those chunks and points the student to the video and timestamp. The ingestion half of the pipeline is built and accounts for 40% of the planned work; retrieval, generation and the interface are being integrated. We describe the data organisation, the algorithms, a storage and search cost estimate, and an evaluation plan covering recall, reciprocal rank, timestamp accuracy and faithfulness.

IndexTerms **- Retrieval-augmented generation, educational technology, speech recognition, semantic embeddings, large language model, lecture video retrieval.**

I. INTRODUCTION

Recorded lectures have become a main study resource in engineering colleges. A single course can easily run to dozens of hours, and a student who has one specific doubt rarely has the time to watch them all. In practice the student drags the progress bar around, guesses, and often gives up. Video platforms make this worse because they index only titles and descriptions, so nothing that the teacher actually says can be searched.

Large language models (LLMs) seem like an obvious fix, since they answer questions in plain language. The trouble is that they answer from what they learned during training, not from the instructor's own explanation. The result can be correct in general but wrong for the syllabus, or simply invented, and it never tells the student where in the course the topic was taught. Retrieval-augmented generation (RAG) reduces this problem by letting the model look up passages from a trusted collection at question time and then answer from them [1], [2]. Because the passages are known, the answer can be traced back to its source.

A RAG assistant for lecture videos has to solve several small problems before it can answer anything. Speech must be turned into text without losing the time of each sentence. The text must be cut into pieces small enough that a retrieved piece is about one idea. Each piece must be represented so that a question and its answer land close together. Whisper handles the first step well for multilingual speech and returns segment times [3]. BGE-M3 gives multilingual embeddings for inputs of up to 8192 tokens [4]. Openly released Llama 3 models can be run locally for the last step [5].

This paper reports the design and current state of such an assistant, built as a final-year project. Our contributions are: (1) an ingestion pipeline that turns raw lecture videos into timestamped JSON chunks and BGE-M3 vectors; (2) a query path that returns an answer together with its source video and timestamp; (3) a description of how the project's files and records are organised, with algorithms and a storage and search cost estimate; and (4) an evaluation plan for retrieval relevance, timestamp accuracy and answer faithfulness. We also report honestly what is finished: the ingestion half, which is 40% of the planned implementation.

Stated simply, the problem is this: given a question in natural language and a set of recorded lectures, return a correct answer that is based on those lectures, together with the place in the recordings where the topic is explained. The answer should come quickly enough for a student to use it while studying, and it should say so when the lectures do not contain the answer.

Section II reviews related work. Section III describes the lecture data. Section IV explains the methods, including data organisation and algorithms. Section V gives the implementation status, a cost analysis and the evaluation plan. Section VI discusses limitations, validity and ethics, and Section VII concludes.

II. Related Work

A. Retrieval-Augmented Generation

Lewis et al. combined a dense retriever with a sequence-to-sequence generator and showed that access to retrieved passages helps on knowledge-intensive tasks [1]. A later survey organises the many RAG variants by how they retrieve, augment and generate [2]. Dense passage retrieval, which encodes questions and passages into the same vector space, is now the usual way of choosing context [12]. We follow that approach and use one encoder for both chunks and questions.

B. Speech and Embedding Models

Whisper is a Transformer encoder–decoder [13] trained on a very large amount of weakly supervised multilingual audio, and it is reasonably robust to accents and noise [3]. For text representation, BGE-M3 supports dense, sparse and multi-vector retrieval in one model and handles long inputs, which suits lecture passages that run for several sentences [4]. For the generator, the Llama 3 family offers open weights at several sizes [5].

C. RAG for Lecture Videos

Several recent works apply RAG to lecture videos. One study uses Whisper transcripts of lectures as the knowledge base for feedback on programming tasks and links students to the matching place in the video [11]. LECTRA combines speech recognition, optical character recognition and scene detection in a chatbot that answers with timestamp and scene references, and it was evaluated with proprietary LLMs [9]. A multimodal RAG framework for question answering in lecture videos was presented at AIMSA 2024 [10].

Our system has the same aim of answers that point to a time in the video. It differs in using an open stack that can run locally, in keeping every intermediate result as a readable JSON file, and in measuring progress against an explicit implementation milestone. It currently uses only the spoken content of the lecture, so text shown on slides is not captured; this is the main gap compared with [9].

D. Gap Addressed

Taken together, the earlier work shows that transcripts, dense retrieval and LLMs can be combined for lecture material, but the details that make such a system easy to reproduce are rarely written down. These include how segments are merged, what a stored record contains, how much storage and search time the index needs, and how success will be measured. This paper tries to fill that gap for a small, locally hosted system by describing those details together with the current state of the implementation.

III. Lecture Data

The knowledge base is built from recorded teaching videos in .mp4 format. Each file is one lecture session by one instructor. Only the spoken content is used for retrieval. The video title and file name are kept as metadata so that every retrieved passage can be traced back to its lecture.

The current corpus has 18 videos from a Web Development course (HTML and CSS), with a total length of 4.21 hours, spoken in Hinglish (translated to English). After transcription and chunking there are 7,252 chunks. Table 1 summarises these figures.

TABLE 1: CORPUS STATISTICS (TO BE COMPLETED WITH PROJECT VALUES)

Item	Value
Lecture videos	18
Total duration (h)	4.21
Transcribed segments	7,252
Text chunks (JSON)	7,252
Embedding dimension	1024

IV. Methods

A. System Overview

The system has two paths, shown in Figure 1. The offline path builds the knowledge base: video, audio, timestamped transcript, chunks with metadata, JSON files, embeddings, and finally a vector store. The online path takes a student question, embeds it, retrieves the Top-K most similar chunks, builds a prompt and generates a grounded answer that cites a video and timestamp. The offline path is implemented. The online path is the next stage.

Four requirements shaped the design. First, every answer must be traceable: the student should always see which lecture and which moment it came from. Second, the knowledge base must be easy to rebuild, because lectures are added and corrected during a semester. Third, the system should run on modest hardware that a college can own, so that no student question has to leave the campus. Fourth, each stage should leave a file that a person can open and check, which makes errors easier to find.

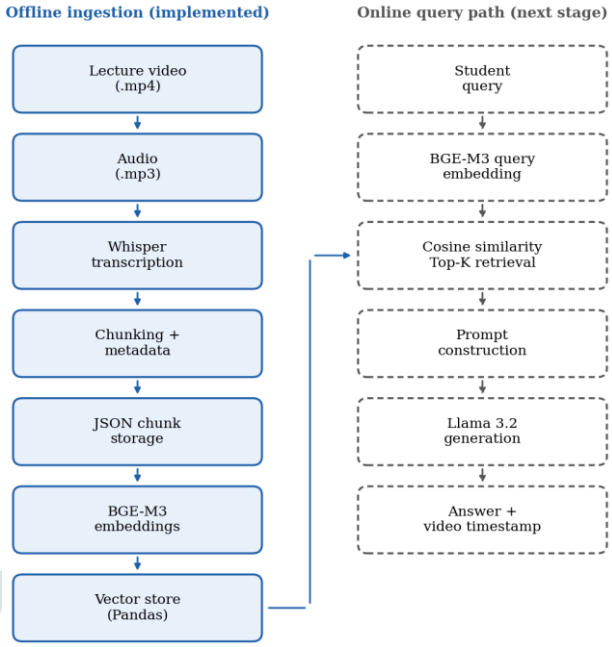

Figure 1: Proposed RAG AI Teaching Assistant. Solid boxes are implemented; dashed boxes are under integration.

B. Video-to-Audio Conversion

Each .mp4 file is converted to .mp3 so that only the speech reaches the recognizer. This makes the input smaller and removes any dependence on the video container. The source file name is kept, so the audio, the transcript and the chunks can all be tied to the same lecture.

C. Whisper Transcription

We transcribe each audio file with the large-v2 model. Whisper returns the transcript as a list of segments, each with a start time, an end time and text:

$$S_k = (t_k^{start}, t_k^{end}, x_k), \quad k = 1, \ldots, M \qquad (1)$$

Here x_k is the text of segment k and M is the number of segments in the lecture. The times are the reason this step matters for the whole system: without them a retrieved sentence cannot be linked back to a position in the video.

D. Chunking and Metadata

A retrieved piece of text is only useful if it is about one idea. Neighbouring segments are therefore merged into chunks of at most 30 tokens, with an overlap of 0 tokens so that an explanation that crosses a boundary can still be found. Chunk j is the record

$$c_j = (id_j, v_j, t_j^{start}, t_j^{end}, text_j) \qquad (2)$$

where id_j is a unique identifier, v_j is the source video, t_j^{start} is the start time of the first segment in the chunk, t_j^{end} is the end time of the last segment, and $text_j$ is the chunk text. Chunking is run over all videos, and progress is tracked so that long jobs can be monitored.

E. Data Organisation and JSON Storage

Keeping each stage's output in its own place makes the pipeline easy to inspect and to re-run. Table 2 shows the layout we use. Chunks are written to one JSON file per lecture, which separates the slow transcription step from the later embedding step: the embedding model or the chunk size can be changed without transcribing anything again. Table 3 lists the fields of a chunk record.

TABLE 2: PROJECT FILE ORGANISATION (ADJUST NAMES TO YOUR FOLDERS)

Folder	Contents	Made by
videos/	Original .mp4 lectures	Source
audio/	Extracted .mp3 files	Conversion
chunks/	One JSON file per lecture	Whisper + chunking
embeddings/	Vectors and metadata table	BGE-M3 + Pandas

TABLE 3: FIELDS OF ONE CHUNK RECORD (ADJUST TO YOUR JSON)

Field	Type	Meaning
chunk_id	string	Unique identifier
video	string	Source lecture name
start	float (s)	Start time in video

Field	Type	Meaning
end	float (s)	End time in video
text	string	Chunk text
embedding	float[1024]	BGE-M3 dense vector

F. BGE-M3 Embedding and Bulk Loading

Every chunk is encoded with BGE-M3 [4]. We use the dense vector and normalise it to unit length:

$$e_j = f(text_j) / ||f(text_j)|| \quad (3)$$

where f is the BGE-M3 encoder. The vectors and chunk records are loaded in bulk into a Pandas DataFrame [6], one row per chunk. This is the index used for retrieval. If the collection grows large, an approximate nearest-neighbour index such as FAISS [7] can replace the DataFrame without changing the rest of the pipeline.

G. Query Embedding and Retrieval

A student question q is encoded with the same model to give e_q. Relevance to chunk c_j is the cosine similarity, which for unit vectors is a dot product:

$$sim(q, c_j) = (e_q \cdot e_j) / (|e_q| \, |e_j|) \quad (4)$$

The K chunks with the highest similarity are the retrieved context:

$$R(q) = Top\text{-}K_j \; sim(q, c_j) \quad (5)$$

H. Grounded Generation with Timestamp Guidance

The retrieved chunks are placed in a prompt that asks Llama 3.2 [5] to answer only from the supplied lecture text:

$$P = [\, I \,;\, c_1, \ldots, c_K \,;\, q \,] \quad (6)$$

Here I is the instruction, c_1 to c_K are the retrieved chunks in order of similarity, and q is the question. Each chunk carries its video name and time range, so the answer can end with the lecture and the timestamp where the student can watch the full explanation. The model used is 3B (llama3.2), run locally via Ollama with temperature 0.2.

The instruction I is short and strict. A template of the kind we use is shown below; the exact wording is tuned during testing.

```
You are a teaching assistant for this course.
Answer ONLY from the lecture excerpts below.
If the excerpts do not contain the answer, say
that you could not find it in the lectures.
After the answer, list the video name and start
time of each excerpt you used.

Excerpts: {chunk 1: video, start, text} ...
Question: {q}
```

I. Algorithms

Algorithm 1 summarises ingestion and Algorithm 2 summarises answering a question. Algorithm 1 is implemented; Algorithm 2 is being integrated.

```
Algorithm 1: Build knowledge base
for each video v in videos/:
  audio  <- to_mp3(v)
  segs   <- whisper(audio)   # start,end,text
  chunks <- merge(segs, Nc, No)
  save json(chunks, chunks/v.json)
for each chunk c in all JSON files:
  c.emb <- normalise(bge_m3(c.text))
table <- DataFrame(all chunks + emb)
```

```
Algorithm 2: Answer a question
e_q    <- normalise(bge_m3(q))
scores <- table.emb . e_q
top    <- K rows with highest scores
prompt <- instruction + top.text + q
answer <- llama(prompt)
return answer, top.video, top.start
```

J. Worked Example

To make the query path concrete, consider a student who asks, “where is html concluded in this course”. The question is embedded, and the table is scored by cosine similarity. Suppose the three best chunks come from the lectures and times shown in Table 4. The prompt is built from their text, and the model writes a short answer followed by the same three sources. The student can then open the first lecture at the stated time. The values below are placeholders until the query path is finished.

TABLE 4: ILLUSTRATIVE RETRIEVAL OUTPUT FOR ONE QUESTION (TO BE REPLACED WITH A REAL RUN)

Rank	Lecture	Start	Similarity
1	13_Entities, Code tag and more on HTML	00:02	0.861
2	13_Entities, Code tag and more on HTML	08:40	0.850
3	02_Your First HTML Website	17:44	0.830

K. Implementation Details

The pipeline is written in Python and is split into small scripts, one for each stage, so that a stage can be run again on its own. The conversion script walks through the video folder and writes one .mp3 file per lecture. The transcription script reads each .mp3 file, calls Whisper, and keeps the segment list with its times. The chunking script merges segments as described in Section IV-D and writes the JSON file of the lecture. The embedding script reads all JSON files, encodes the chunk text in batches, normalises the vectors, and collects everything in one DataFrame. Each script prints its progress and skips work that is already finished, which matters because transcribing a long lecture can take a long time.

Two practical details deserve mention. First, file names are used as the key that links a video, its audio, its JSON file and its rows in the table, so names must be unique and are never changed after ingestion. Second, a lecture that fails in one stage does not stop the others: it is logged, and the rest of the folder continues. These choices are modest, but they make a long ingestion run easier to trust and to repeat.

V. RESULTS

A. Implementation Status

Table 5 lists the modules finished for Review 3 and the weight each carries in the plan. Video-to-MP3 conversion, Whisper transcription with timestamps, chunking with metadata, JSON storage, and BGE-M3 embedding with bulk loading add up to 5% + 10% + 10% + 5% + 10% = 40% of the implementation, as Figure 2 shows. The remaining 60% covers query processing, retrieval, generation, the interface and final validation.

TABLE 5: IMPLEMENTATION STATUS OF PIPELINE MODULES

Module	Tool	Weight	Status
Video → MP3	Audio extraction	5%	Done
Transcription + timestamps	Whisper	10%	Done
Chunking + metadata	Python	10%	Done
JSON chunk storage	JSON	5%	Done
Embeddings + bulk loading	BGE-M3, Pandas	10%	Done
Query embedding, retrieval	BGE-M3, cosine	—	Next
Answer generation	Llama 3.2	—	Next
Interface, validation	—	—	Next

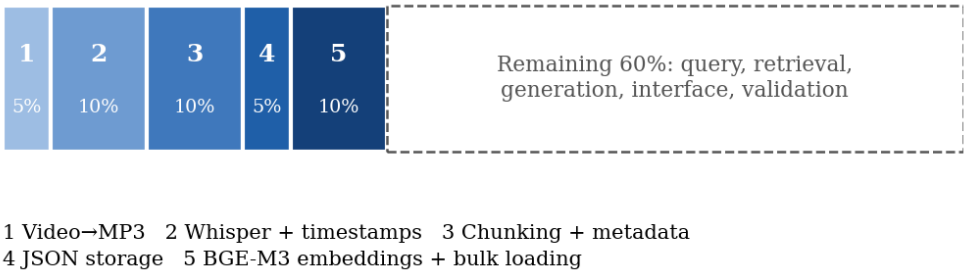

Figure 2: Implementation milestone: completed modules contribute 40% of the planned pipeline.

B. Storage and Search Cost

The cost of the index can be estimated before the system is finished. With N chunks and d-dimensional float32 vectors, the vector storage is

$$Storage \approx N \cdot d \cdot 4 \text{ bytes} \quad (7)$$

For BGE-M3 with d = 1024, each chunk needs 4096 bytes, so 10,000 chunks need about 41 MB for vectors, plus the text and metadata. Exhaustive cosine search costs N · d multiply–adds per question, which is about 10 million operations for 10,000 chunks. That is small next to the cost of running the generator, so a plain DataFrame is enough at this scale, and an approximate index only becomes useful when the collection grows by orders of magnitude.

C. Evaluation Protocol

Retrieval and generation are not yet integrated, so answer quality has not been measured. We fix the protocol now so that later results can be compared fairly. A test set of 30 questions is written from the lecture content. For each question we mark the correct chunk or chunks and the true time range $[\tau_{start}, \tau_{end}]$ in the video.

Retrieval relevance. Recall@K is the share of questions whose relevant chunk is among the Top-K. Mean reciprocal rank (MRR) rewards putting the relevant chunk near the top:

$$MRR = (1/|Q|) \Sigma_{i=1}^{|Q|} 1 / rank_i \qquad (8)$$

Timestamp accuracy. A returned time $\hat{t}$ counts as correct if it lies within a tolerance δ of the marked range:

$$Acc_{ts} = (1/|Q|) \Sigma_i 1[\, \tau_{start} - \delta \le \hat{t}_i \le \tau_{end} + \delta \,] \qquad (9)$$

Faithfulness and relevance. Faithfulness is the share of statements in an answer that the retrieved text supports; answer relevance is how well the answer addresses the question. Both can be scored by human raters or by an automatic RAG evaluation framework [8]. We will also compare Top-K = 3 and Top-K = 5 and record the time taken per question. Table 6 is the template for the results.

TABLE 6: PLANNED EVALUATION RESULTS (TO BE FILLED AFTER INTEGRATION)

Metric	Top-K = 3	Top-K = 5
Recall@K	[TBD]	[TBD]
MRR	[TBD]	[TBD]
Timestamp accuracy	[TBD]	[TBD]
Faithfulness	[TBD]	[TBD]
Answer relevance	[TBD]	[TBD]
Latency per query (s)	[TBD]	[TBD]

D. Experimental Setup

To make later results reproducible, every run will record the hardware, software and settings listed in Table 7. These values are not final yet, so they are left blank.

TABLE 7: EXPERIMENTAL SETUP TO BE REPORTED (TO BE COMPLETED)

Item	Value
Processor / GPU	Intel Core i5 / NVIDIA T4 GPU
RAM	16 GB
Python and library versions	Python 3.10, whisper, pandas, scikit-learn, joblib
Whisper model	large-v2
Chunk size / overlap	30 / 0 (Whisper segment-level)
Embedding model	BGE-M3 (dense)
Generator	Llama 3.2 3B
Top-K, temperature	K = 5, 0.2

E. Error Analysis Plan

Besides the averages in Table 6, we will read the failed cases one by one and sort them by the stage that caused the failure. We expect four main groups. The first is a transcription error, where a term was heard wrongly and so was indexed wrongly. The second is a retrieval miss, where the right chunk exists but is not in the Top-K. The third is a chunking problem, where the answer is split across two chunks or buried in a very long one. The fourth is a generation problem, where the right context was retrieved but the answer says something the context does not support. Counting the cases in each group shows which module should be improved first, and it keeps the project honest about where the errors come from.

F. Deployment Notes

The cost of answering a question has three parts: embedding the question, searching the table, and generating the answer. Transcription, chunking and embedding of the lectures all happen offline, so they do not add to the waiting time of the student. Of the three online parts, generation is expected to take by far the longest, and its time grows with the model size and the length of the retrieved context, which is one reason to keep K small. Embeddings of the lectures can be saved to disk and loaded at start-up, so the encoder is only needed at run time for the question. Keeping a version number in the folder names and in each record makes it possible to say which transcript, chunk size and embedding model produced a given answer.

G. Summary of Status

In short, the system can already take a folder of lecture videos and produce, for every lecture, a JSON file of timestamped chunks and a table of normalised BGE-M3 vectors. What it cannot yet do is answer a question end to end. The measurements that would show how well it answers, namely Recall@K, MRR, timestamp accuracy and faithfulness, depend on the next stage, and the tables prepared for them are left open on purpose. We prefer to leave them open rather than fill them with estimates.

VI. Discussion

A. Design Choices

The most important decision was to keep a timestamp on every segment and chunk. It changes the assistant from something that answers questions into something that also tells the student where to look. Writing chunks to JSON before embedding was the second: transcription is the slowest step, and this way it is done once. Using the same encoder for chunks and questions keeps both in one vector space, which is what cosine search assumes.

B. Limitations

The assistant can only be as good as the transcript. Technical terms, code and formulas that the teacher says aloud may be recognised wrongly, and they will then be indexed wrongly. Anything shown only on a slide or the board and not spoken is invisible to the system. Chunk size is a trade-off: small chunks give precise timestamps but may lose the surrounding explanation, and large chunks do the opposite. Finally, because retrieval and generation are not yet integrated, we make no claim about end-to-end accuracy.

C. Comparison with Related Systems

Compared with the lecture-feedback system of [11], which also uses Whisper transcripts and links to positions in the video, our pipeline stores chunks in a documented JSON format and is designed for open question answering rather than feedback on a programming task. Compared with LECTRA [9], we use fewer input signals, since slides and scenes are not read, but we rely only on open components that can run on a single machine. Which of these trade-offs matters more can only be settled by the evaluation in Section V, run on the same questions for each system if the other systems can be reproduced.

D. Scalability and Maintenance

A college course changes from year to year, so the knowledge base has to be updated without starting again. Because each lecture has its own JSON file, a new or corrected lecture only needs to be transcribed, chunked and embedded once, and its rows can be added to the table. If the embedding model is replaced, the stored text is reused and only the vectors are recomputed. If several courses share one installation, a course field in each record allows a question to be searched against one course only, which also avoids answers that mix material from different subjects.

E. Threats to Validity

Several factors could limit how far future results will generalise. The corpus comes from one instructor, so accent and speaking style are narrow. Questions written by the project team may be easier than real student questions, so a sample of genuine student questions should be added. The marked time ranges involve human judgement, which can differ between annotators. Finally, generated answers vary with the model size and the sampling temperature, so settings must be reported with every result.

F. Ethical and Practical Considerations

Lecture recordings belong to the instructors, and their use in the knowledge base should have their permission. Running the models locally keeps student questions on the institution's own hardware, which avoids sending them to an outside service. The assistant can still give a wrong answer, so every response should show its source video and time and invite the student to check it. It is meant to support, not replace, the teacher.

G. Rejecting Low-Confidence Questions

A teaching assistant should sometimes decline. The similarity score in (4) gives a natural signal: if the highest similarity, $\max_j \text{sim}(q, c_j)$, is below a threshold θ, the question is probably outside the lectures, and the system can say that it found no relevant segment instead of letting the generator answer from its own memory. The threshold can be tuned on validation questions to balance how many questions are answered against how many answers are wrong. This is a proposal and has not been tested.

H. Future Work

Next steps are to finish the query path and the interface, and then to run the evaluation in Table 6. After that we plan to try hybrid dense and sparse retrieval with the extra BGE-M3 outputs [4], a re-ranking step, text extraction from slides and video frames as in [9], an approximate nearest-neighbour index for larger collections [7], and a player link that opens the video directly at the returned timestamp.

VII. Conclusion

We described a RAG-based teaching assistant that turns recorded lectures into a knowledge base in which every passage keeps its place in the video. The ingestion pipeline is implemented: video-to-audio conversion, Whisper transcription, chunking with metadata, JSON storage, and BGE-M3 embedding with bulk loading. It makes up 40% of the planned work. We also described how the project's files are organised, gave the retrieval and generation algorithms, estimated storage and search cost, and fixed an evaluation protocol. The next stage is to connect retrieval to Llama 3.2, measure the results, and release the interface to students.

Acknowledgment

The authors thank their guide, Dr. Rahul Kale, Assistant Professor, and the Department of Artificial Intelligence and Data Science for their guidance and for access to the lecture recordings used in this project.

References

[1] Lewis, P., et al. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems, 33: 9459–9474.

[2] Gao, Y., et al. 2023. Retrieval-augmented generation for large language models: a survey. arXiv:2312.10997.

[3] Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., and Sutskever, I. 2023. Robust speech recognition via large-scale weak supervision. Proceedings of the International Conference on Machine Learning, 28492–28518.

[4] Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D., and Liu, Z. 2024. BGE M3-Embedding: multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. arXiv:2402.03216.
[5] Dubey, A., et al. 2024. The Llama 3 herd of models. arXiv:2407.21783.
[6] McKinney, W. 2010. Data structures for statistical computing in Python. Proceedings of the 9th Python in Science Conference, 56–61.
[7] Johnson, J., Douze, M., and Jégou, H. 2021. Billion-scale similarity search with GPUs. IEEE Transactions on Big Data, 7(3): 535–547.
[8] Es, S., James, J., Espinosa-Anke, L., and Schockaert, S. 2023. RAGAs: automated evaluation of retrieval augmented generation. arXiv:2309.15217.
[9] Thaipisutikul, T., Jiwyam, K., Ngampis, K., Saejiam, J., Tansue, J., Noraset, T., Chen, S. C., and Chen, Y. N.. 2026. LECTRA: bridging video lectures and student queries via a RAG-based chatbot for timestamped and semantic responses. Springer, Singapore. doi: 10.1007/978-981-95-9843-4_3.
[10] Tanner, T., Marfurt, A., and Oğul, H.. 2025. Enhancing question answering in lecture videos with a multimodal retrieval-augmented generation framework. AIMSA 2024, Springer, Cham. doi: 10.1007/978-3-031-81542-3_15.
[11] Jacobs, S., and Jaschke, S.. 2024. Leveraging lecture content for improved feedback: explorations with GPT-4 and retrieval augmented generation. arXiv:2405.06681.
[12] Karpukhin, V., et al. 2020. Dense passage retrieval for open-domain question answering. Proceedings of the Conference on Empirical Methods in Natural Language Processing, 6769–6781.
[13] Vaswani, A., et al. 2017. Attention is all you need. Advances in Neural Information Processing Systems, 30: 5998–6008.