Multimodal Conversational AI for Information Retrieval

KARISHMA KUNWAR, M.Sc. Data and Knowledge Engineering, [253388]
SAI NISHCHAL GAMINI, M.Sc. Digital Engineering, [250774]
SAI TEJA DAMPANABOINA, M.Sc. Informatik, [231784]

ACM Reference Format:
Karishma Kunwar, Sai Nishchal Gamini, and Sai Teja Dampanaboina. 2025. Multimodal Conversational AI for Information Retrieval. 1, 1 (May 2025), 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Abstract

Large-scale language models (LLMs) have revolutionised dialog systems, yet their fixed-cut-off knowledge and limited access to rapidly changing, domain-specific content restrict real-world usefulness. To overcome these shortcomings we present a multimodal Retrieval-Augmented Generation (RAG) pipeline that seamlessly handles both spoken and written queries and can return answers in either text or speech. The architecture explicitly decouples knowledge storage, indexing and retrieval from generative reasoning, allowing continuous ingestion of new documents while keeping latency low

A fine-tuned RoBERTa-base intent classifier (LoRA-adapted on an 8 k-row synthetic corpus) routes each query to either web or vector-store search with perfect held-out accuracy (1.00). Content is embedded with Open-CLIP ViT-L/14 and stored in a Milvus vector database; retrieval combines dense inner-product search with a lightweight PageRank diffusion re–ranker that lifts MAP, nDCG and MRR by approx. 0.09 absolute while adding only 25 ms overhead. The pipeline sustains approx. 200 QPS on commodity hardware with 98 ms average end-to-end latency. Final answers are composed by the Gemini LLM and, when requested, vocalised through Piper TTS; incoming speech is transcribed by Whisper STT.

Experiments on a purpose-built test collection demonstrate near-perfect top-5 precision (0.99) and illustrate the value of combining dense and graph-based retrieval with intent-aware routing. The resulting system delivers up-to-date, context-grounded responses across modalities, offering a robust foundation for educational and other knowledge-intensive conversational applications.

Authors' Contact Information: Karishma Kunwar, M.Sc. Data and Knowledge Engineering, [253388]; Sai Nishchal Gamini, M.Sc. Digital Engineering, [250774]; Sai Teja Dampanaboina, M.Sc. Informatik, [231784].

ACM XXXX-XXXX/2025/5-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 Introduction

Modern conversational AI applications increasingly demand not only fluent and contextually accurate natural-language generation but also timely access to precise, domain-specific knowledge. Traditional end-to-end large language models (LLMs), such as GPT and its variants, have achieved remarkable fluency and broad-ranging reasoning capabilities but suffer from critical limitations. Chief among these limitations is their reliance on static knowledge, which results from fixed training cutoffs, restricting their ability to access the most recent information. Additionally, their coarse grasp of specialized or rapidly evolving content often hinders their performance in specific, nuanced domains, thus limiting their utility for tasks requiring expert-level precision.

Retrieval-Augmented Generation (RAG) systems effectively bridge these gaps by integrating high-capacity LLMs with external, dynamically updatable knowledge repositories. Unlike conventional approaches, RAG architectures allow the continuous enrichment and updation of the knowledge base, ensuring the freshness and relevance of the information available. This report details the design, development, and practical implementation of a robust Multi-modal RAG system that uniquely accommodates both textual and spoken user inputs. Leveraging advanced techniques in intent classification, our system accurately discerns user queries and seamlessly retrieves pertinent information from diverse, heterogeneous data sources. Furthermore, it synthesizes retrieved data into contextually grounded, coherent, and highly accurate natural-language responses.

Our proposed architecture deliberately separates the generative capabilities of the LLM from data management functionalities such as storage, indexing, and retrieval. This strategic decoupling enables continuous ingestion of new documents, rapid and efficient retraining of embedding models, and near-real-time updates to the underlying vector store. Consequently, this method effectively mitigates the problem of information staleness, significantly enhancing the overall responsiveness, reliability, and domain fidelity of conversational AI solutions.

3 Literature Review

In recent years, advancements in Large Language Models(LLMs) and AI-driven dialog systems have enabled the creation of more dynamic and responsive conversational platform. However, the challenge of delivering accurate, contextually relevant information to students remains largely unresolved within the education domain. With the advancement in LLMs and state-of-the-art models, a lot of work has already been done in the field of question and answering, producing the most accurate results. Existing literature explores various components, such as intent classification, retrieval-augmented generation(RAG), and multimodal input processing, that can together form the backbone of an efficient academic information assistant.

3.1 Large language models in Education

Large Language Models (LLMs) are transforming education by making learning more accessible, efficient, and personalized. With advanced models like ChatGPT, DeepSeek, and EduChat, students now rely on LLMs for everything from selecting thesis topics to solving complex Machine Learning problems. Tools such as SimClass, a multi-agent LLM-powered classroom simulator, have demonstrated the potential of AI to recreate dynamic teacher-student and peer-to-peer interactions in virtual learning environments [13]. Similarly, CodeTutor, an AI-based programming assistant, has shown positive effects on student performance, particularly among those new to AI, though concerns remain around its ability to foster critical thinking, with many students preferring traditional teaching assistants over time[7]. At the core of this innovation is the AI4EDU field, which bridges education, data mining, and AI. It emphasizes the role of LLMs in human-in-the-loop systems that adapt to learners' needs. Despite their promise, these systems face persistent challenges in algorithmic robustness, user interface design, and ethical considerations[10].A systematic review of 118 peer-reviewed studies since 2017

identified 53 use cases for LLMs in education—ranging from grading and content generation to teaching support and student profiling. However, it also flagged significant issues such as low technological readiness, lack of transparency, and insufficient attention to privacy and replicability, all of which hinder real-world adoption[12]. As research progresses and these technologies evolve, it will be crucial to strike a balance between innovation and responsible implementation, ensuring that LLMs enhance the learning experience without compromising fundamental educational values.

3.2 Existing Intent Classification Techniques

Intent classification has become a core component of educational dialogue systems, aiming to improve student-AI interactions. One study proposed a web-based tutoring system using LLMs and retrieval-augmented generation (RAG) via LangChain. Their evaluations showed that the LLaMA 3 model, combined with history-aware retrievers and larger documents, delivered more consistent performance in live QA scenarios[11]. In a separate work, the LAAI tutor introduced a fine-tuned intent classifier using LoRA on a custom dataset. It effectively recognized varied student behaviors (e.g., questions, confusion, boredom), outperforming few-shot GPT baselines with a balanced F1-score of 0.83[5]. Beyond education-specific tools, BERT-based joint models for intent classification and slot filling have shown strong generalization even with limited labeled data, outperforming earlier RNN-based approaches[1]. Similarly, in large-scale systems like Cortana, gated models (GRU/LSTM) outperformed feedforward and recurrent networks, confirming their robustness in real-world scenarios[9]. These works highlight both the progress and ongoing challenges in creating scalable, adaptive, and context-aware intent classification systems for educational use.

3.3 Our Proposed Work

While existing educational models primarily focus on text-based interactions and standard retrieval methods, our proposed system advances the state of the art through two key enhancements. First, it leverages a multi-modal architecture that supports both speech and text-based queries (for both input and output), enabling more natural and inclusive student engagement. Second, we introduce a customized RAG retriever that integrates hybrid retrieval techniques(integrating both the dense retrieval and the graph based diffusion re-ranker) with advanced embedding models, significantly improving the relevance and accuracy of responses retrieved from the vector database. Along with that we have also worked on user query optimization to improve the retrieval. By integrating these features together we have come up with a multimodal conversational AI.

4 Working and Methodology

4.1 About the System Architecture

The architecture of our Multimodal Conversational AI is designed to support multimodal input and advanced information retrieval, ensuring high interactivity and accuracy. Here is how it works:

4.1.1 User Input: The system accepts inputs from the user in two forms:

- Text Input: Directly provided by the user through a text interface.
- Speech Input: Captured as audio, which is then converted to text using a *Speech to Text* module.

4.1.2 Intent Classification: Once the system takes in the input from user and pass it to the intent classifier.

- Analyzes the input to determine the user's intent (e.g., asking a question, requesting information, or issuing a command).
- Labels the input with relevant categories or intents, which guide the system's next steps.
- Forwards the labeled input to the *Retriever* or search API for further processing.

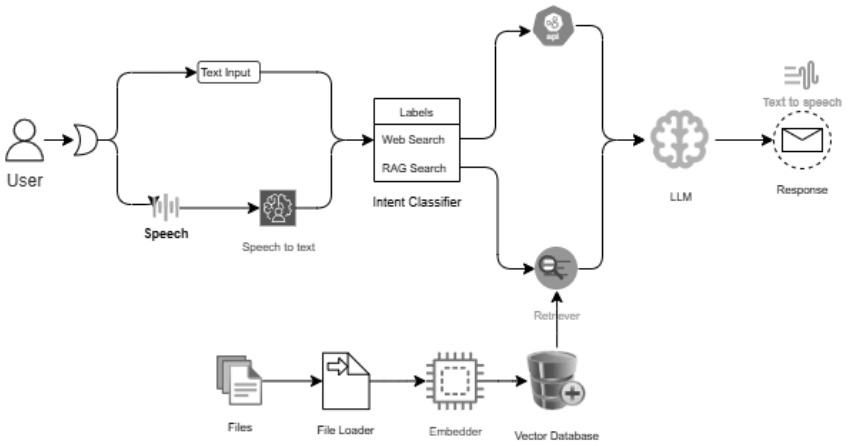

Fig. 1. Multimodel AI overall system architecture

The Intent Classifier is crucial for contextual understanding and ensures the system responds appropriately to the user's needs.

4.1.3 Information Retrieval: Information retriever is a core component that enables Retrieval-Augmented Generation(RAG). It has following components:

- File Loader: File loader takes the PDF or Image file and then detects the text accordingly. The file loader is strong enough to load plain text, text inside the images, and also extracts the text and information from the tables.
- Vector Embeddings: After extraction of text and image from a file(PDF or image), it uses a embedding model(CLIP model). The inputs are transformed into embeddings and Those embeddings are then stored in the vector database.
- Vector Database: The embeddings are then stored in a vector database. Those vectors will later interact with the query vector and the retriever retrives the relevent vectors that matches with the context of the query.
- Information Retrieval - SERP API: The labeled input forwarded to the search API is handled by SERP API which helps to extract the most relevant information from the internet.

4.1.4 LLM Response Generation: LLM is a very important component of our system. It actually takes the output generated by the intent classifier based on users query and then summarizes it in a contexual form as a final output.

4.1.5 Output: Once the LLM generates a response, it is delivered to the user in one of the two forms:

- Text Response: Directly sent to the user as text.
- Speech Response: The response is converted in the form of speech(using text to speech model) and finally delivered to the user.

4.2 Models used in our system

4.2.1 PIPER (Text-to-speech): A text-to-speech (TTS) model is a technology that converts written text into spoken words. It uses natural language processing and machine learning to understand the structure and meaning of the text. The model analyzes elements like grammar, punctuation, and context to generate natural-sounding

speech.

The internal working of the model is described in 3 transformations:

- Graphemes to phonemes(G2P): Using neural networks, the input text is converted into tokens(graphemes) and then it is converted into phonetic representations which are known as phonemes(sound units).
- Phonemes to acoustic features: The phonetic tokens(P) which are generated in the first transformation are now processed through (LSTM or transformer based network) converted to acoustic features) that describe the speech's spectral content. The model outputs mel-spectrograms(S), which represent the energy levels of frequency bands over time. It is represented as :

$$S = f(P); where\ f\ is\ a\ neuralnetwork\ function$$

- Acoustic features to hearable waveform: The mel-spectrograms(S) from the previous transformation are converted into Human hearable output as waveform which is in form of an audio This is done by a component known as vocoder(G), which maps these (S) into a waveform(W). This can be respresented as:

$$W = g(S); where\ g\ is\ vocoder\ function$$

  We integrated the PIPER (TTS) model from Hugging Face to convert textual input into clear and expressive speech[8]. Users provide text input, which is processed by the PIPER TTS model to generate corresponding audio output, enabling seamless text-to-audio conversion for enhanced accessibility and user interaction.

4.2.2 WHISPER(Speech to text): A speech-to-text (STT) model is a technology that converts spoken language into written text. It uses advanced techniques in signal processing, natural language processing, and deep learning to recognize and transcribe audio input accurately. The model analyzes sound waves, identifies patterns in speech, and matches them to words and phrases in a language model.
The internal working of the model is described in 4 transformations:

- Preprocessing of audio: The raw audio input(X) from the user is converted into discretized samples(Xi) at regular intervals for feature extraction. This is represented as:

$$X = X_i(X_1, X_2..X_n)$$

- Feature extraction: The discretized audio sequence(Xi) is now transformed into Spectrograms(S) using short term fourier transforms(STFT), which capture high frequencies and then these are converted into Mel-Frequency Cepstral Coefficients (MFCC), which extract compact and meaningful features. These are represented as :
  STFT: The audio is transformed using the STFT to get a time-frequency representation:

$$X(t, f) = STFT(x(t)) where\ X(t, f)\ is\ the\ spectrogram\ at\ time(t)\ and\ frequency(f).$$

  Mel Filterbank: The spectrogram is passed through a Mel filterbank to get the Mel spectrogram:

$$M(t, f) = MelFilterBank(X(t, f))\ where\ M(t, f)\ represents\ the\ Mel\ spectrogram.$$

- Speech recognition(Transcription): M(t,f), which captures the audio features is converted into text using the acoustic model (usually a neural network such as CTC-based models or transformers). In models like Whisper, the CTC loss function is used for training the model, as it allows the model to output sequences of different lengths than the input sequence. Then tokens (Y') are generated which can be decoded in readable form.
- Postprocessing: The raw sequence of tokens (Y') predicted by the model are converted into human-readable text. This is represented as:
  The final transcription is the decoded sequence y=Decode(Y'), which is the text that corresponds to the original spoken input. We utilized the Whisper Small STT model from Hugging Face to transcribe spoken input into written text. Users provide speech input, which is processed by the Whisper model to generate accurate text output, enabling efficient voice-to-text conversion for improved accessibility and data processing.

4.2.3 RoBERTa base. An intent classifier model is a machine learning system designed to identify the purpose or goal behind a user's input, typically in natural language. It analyzes text or speech to determine the user's intent. The model uses techniques from natural language processing, including tokenization, embedding, and classification algorithms, to predict the most likely intent from a predefined set of categories.
The internal working of the model is described in 4 transformations:

- Tokenization:The input text is converted into tokens which are comprehendible by the model

$$X = [X_1, X_2, X_3 ... X_n];\ where\ X\ is\ the\ sequence\ of\ tokens\ representing\ the\ input\ text.$$

- Embeddings representation: The tokens are converted into continuous dense vectors using a pre trained transformer. There will be a Embedding matrix(E) which will have the embeddings (E_n) which are each token(X_n) is mapped as En). Represented as:

$$E_n = Embedding[X_n]; E = [E_n]$$

- Self-Attention and Transformer Layers: The Embedding matrix(E) is passed through a self attention mechanism which generates attention scores and are used to compute a weighted sum of values, which helps in capturing the dependencies between words in the context of the entire sentence.
- Classification Layer (Intent Prediction): After the text is processed by the transformer model, a classification layer (a simple feedforward neural network) is applied to the final hidden state(CLS token representation). The model outputs probabilities for each intent class. For each intent class (C_j) the model computes a score (Sj) using CLS token (H_c) where H is contextual representation of tokens (X_n). This is represented as :

$$S_j = W_j.H_c + B_j;$$

  W is weight vector for intent class(C) and (B_j) is bias term The (S_j) are passed through a softmax function to obtain the probabilities P(C_j/X)
- Output(Predicted Intent): Classifying the input into one of the predefined intent classes.The model selects the intent class with the highest probability:

$$C' = argmax P(C_j/X); C'\ is\ predicted\ intent$$

We employed the RoBERTa-base model from Hugging Face as an intent classifier to distinguish between web search and database search queries[4]. The model analyzes the text and classifies the intent, enabling the system to route the query to the appropriate processing pipeline.

4.2.4 Open-clip Vit-L/14(Conversion of input into embeddings for vector database): An embedding model transforms text and image inputs into embeddings which are numerical vector representations that capture the key features and semantics of the input. These embeddings can then be stored in a vector database for various retrieval tasks.

The internal working of the model is described in 4 transformations:

- Text encoding: The text is passed through a transformer-based encoder to produce its embedding. The text ($T = t_1 ... t_n$) is segregated into tokens. The model processes the tokens sequentially, applying self-attention and positional encoding. The output at each layer is denoted as H_i for each token t_i. The final hidden state $H = H_1, H_2, \ldots, H_n$ represents the processed sequence of tokens. The final output is a single embedding (Vt) that captures the semantic information of the entire input text. This is typically generated by pooling (mean, max, or CLS token pooling) over the final hidden states:$V_t = Pool(h_1, h_2, \ldots, h_n)$
- Image encoding: Image(I) is processed using a CNN or a vision transformer.So, before it is sent to a model, it is resized and normalized:$I' = Normalize(I)$ Then, The normalized image(I') is passed through the image encoder, which extracts features from the image. The output of this encoder is a high-dimensional vector (Vi), which represents the image in the same embedding space as the text embeddings.$V_i = ImageEncoder(I')$
- Mapping the Embeddings to Shared Space: Both the embeddings will be projected into same space using the projection heads. The projection function for the text is:$V_{(text,projected)} = W_{(text)}.v_{(text)}$ and similarly for the image:$V_{(image,projected)} = W_{(image,vimage)}$, where W_{text} and W_{image} are the projection matrices (learned during training) that ensure both embeddings are mapped to the same space.
- Storing Embeddings in a Vector Database: Once the embeddings are generated, they can be stored in vector database for similarity based searches.
- When a new query (text or image) is provided, it is passed through the same encoding pipeline to obtain its embedding (V_{query}). The database then compares this query embedding to the stored embeddings using similarity measures like cosine similarity or dot product.

  $$\text{Cosine Similarity}(V_{query}, V_{stored}) = V_{query}.v_{stored} || v_{query}\ v_{stored}$$

  where the dot product (V$query$. V$stored$) measures the similarity between the query and stored embeddings, and the norms ||V$query$|| and ||V$stored$|| normalize this value.
- The vector database uses efficient indexing to enable fast retrieval of the nearest neighbors based on similarity metrics.
- We utilized the Open-CLIP ViT-L/14 model from Hugging Face to process user-provided images,text documents containing diagrams, graphs, or handwritten notes.
- The model converts each input into a vector embedding, which are then stored in a vector database. This enables efficient similarity searches, allowing us to retrieve visually or semantically similar images quickly.

4.2.5 Vector database(Milvus): A vector database is a specialized system designed to store, index, and search high-dimensional vector embeddings efficiently. Each item, like an image or text, is represented as a vector in a continuous space. These databases enable fast retrieval of similar vectors by calculating distances such as cosine similarity or Euclidean distance between embeddings. Instead of exact matching like traditional databases, vector databases focus on finding the nearest neighbors based on similarity. The working of model is described in 5 transformations:

- Generate Embedding: From model, for an input (image/text) we get an embedding:

  v ϵ Rd where d is the dimension of the embedding space (512).

- Prepare for Storage: We may also attach metadata (like an ID or label) with the embedding. Example structure:

  Data=(ID,v,Metadata); v = vector
- Insert into Vector Database: Milvus stores the vector(v) into a collection (similar to a table) and automatically indexes it.
- Mathematical view: Embeddings are treated as points in a d-dimensional space.
  The set of stored embeddings can be thought of as:

$$V = v_1, v_2, \ldots, v_N \ where\ N\ is\ the\ number\ of\ vectors\ stored.$$

- Indexing: To make search faster, Milvus builds an index like HNSW or IVF over the embeddings. Indexing organizes vectors so that nearest neighbor search is approximately: Find Vi such that

$$V_i = \arg\min_{v \in V} d(q, v),\ where\ q\ is\ the\ query\ vector,\ and\ d\ is\ a\ distance\ metric\ like:$$

$$Cosine\ distance: d(q, v) = 1 - \frac{q \cdot v}{\|q\|\,\|v\|}$$

$$Euclidean\ distance: d(q, v) = \|q - v\|^2$$

- Milvus is an open-source vector database designed for storing and searching large-scale embeddings efficiently. It supports high-performance similarity search using advanced indexing methods like Heirarchial navigable small worlds(HNSW) and Inverted file index(IVF). We used Milvus to store embeddings generated by the OpenCLIP model, enabling fast retrieval of similar vectors for AI-driven applications.

4.2.6 LLM (Gemini): Large Language Model (LLM) plays a critical role in generating the final output after the input has been processed by the intent classifier and retriever.Once the intent classifier determines the type of query (text or image-based) and the retriever fetches relevant information, the LLM takes over. The LLM is responsible for understanding the context of the retrieved data and generating a natural, coherent response that aligns with the user's intent.

We integrated the Gemini LLM to generate intelligent, context-aware responses. Based on the classified user intent—either web search or database search—the system routed the query accordingly. Gemini then processed and presented the final results in a clear and conversational format. Its multimodal capabilities, strong contextual understanding, and scalable performance made it a robust choice for delivering high-quality output across varied inputs[2].

5 Dataset + experimental settings:

5.1 Dataset generation:

For preparing the dataset, to train the intent classification model, we have chosen synthetic data generation approach. The SyntheticDataSetGenerator.py program constructs a synthetic question-answering dataset that can be fed into retrieval-augmented generation (RAG) or search-based systems. It does this by

(1) parsing a folder full of source documents,
(2) asking a large multimodal model to invent context-aware questions about each document,
(3) asking a text-only model for fresh, open-domain questions, and
(4) writing every question—together with its provenance and a label—into a single CSV file.

PARAMETER	DEFAULT	MEANING/IMPACT
documents_dir	./input_documents	folder scanned for PDFs or .txt files
num_questions_per_doc	5	How many BLIP-2 questions to ask for each source file
num_general_queries	20	How many open domain questions to genrate per run
output_csv	production_generated_questions .csv	Destination file for the final dataset

Fig. 2. Synthetic dataset generation parameters

After this synthetic data set has been generated, we have added questions from an already existing data set (https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot?resource=download/)). Finally, we have achieved a dataset set with 8K rows of questions labelled as RAG search and web search.

COLUMN	EXAMPLE VALUE	NOTES
Document	course_catalog.pdf/General	Source file name; "General for web" queries
Question	"What are the courses offered with CP?"	Synthetic, machine generated
Label	RAG search / web search	Indicated which generation path produced the question

Fig. 3. Resulting dataset structure

5.2 Data ingestion & pre-processing:

- A CSV file containing Question(free text) and Label columns is validated, then labels are mapped to consecutive integers for compatibility with PyTorch. An 80 / 20 stratified split preserves class balance between the training and validation sets. Text is tokenized with RobertaTokenizer(max_length = 64, truncation & static padding) so that each example is represented by input_ids, attention_mask, and labels [3].
- The backbone is RobertaForSequenceClassification with its classification head resized to the task-specific label space discovered in step 1. RoBERTa offers strong out-of-the-box language representations and has been shown to outperform vanilla BERT when hyper-parameters are carefully tuned [6].
- LoRA adaptation via HuggingFace PEFT Using the PEFT library, a LoraConfig is defined with r = 8, alpha = 16, dropout = 0.05, targeting the query and key projection matrices of every self-attention block. The call to get_peft_model freezes all original RoBERTa weights and injects two rank-8 matrices per target weight, yielding 0.5 % of the original parameter count while leaving inference latency virtually unchanged [4].

- Training regime hyper-parameters are conservative to minimize catastrophic forgetting:

setting	value	rationale
epochs	5	quick convergence with LoRA
LR	2×10^{-5}	standard for encoder fine-tuning
batch	8	fits a single consumer GPU
weight decay	0.01	combats over-fitting on small data

Fig. 4. Hyper-parameter settings for LoRA finetuning

- Model persistence & inference : After training, both the LoRA-augmented model and the tokenizer are written to ./intent_model_lora/. A helper ‘classify_intent()‘ wraps tokenisation and forward pass, translating the predicted index back to the human-readable intent string, and an interactive loop demonstrates usage. (https://huggingface.co/blog/peft?utm_source=chatgpt.com)

5.3 Document Ingestion & Embedding Pipeline:

- Initialization of the Vector Store: A lightweight Milvus instance is launched locally and a fresh collection is created, ensuring any previous collection of the same name is dropped to avoid conflicts. The collection schema includes an auto-incrementing identifier, the embedding vector, and several text-oriented metadata fields (full chunk text, a short preview, source path, and an incremental chunk number).
- Source-document preparation: All files found under the configured “docs” directory are ingested. PDFs are converted to Markdown with table-structure retention and OCR for scanned pages; plain-text files are read as-is. A Markdown copy of each processed file is written to disk to provide a human-readable intermediary.
- Logical segmentation of content: Markdown is split into blocks where every heading line and every table row forms its own block; other lines are bundled into paragraph blocks. These blocks are then grouped into overlapping text chunks that respect a configurable word-count limit, providing continuity across chunk boundaries.
- Semantic encoding: Each chunk is transformed into a 512-dimension text embedding using the OpenAI CLIP ViT-B/32 model; embeddings are L2-normalized to unit length. Processing automatically selects GPU when available, falling back to CPU otherwise.
- Vector-store population: For every chunk, the workflow inserts: the normalized embedding, full chunk text, a short preview snippet, the source-file path, and its sequence number. Progress is logged for transparency while ingesting large document sets.
- Index construction & deployment: An IVF-FLAT index with inner-product metric is built on the embedding field to accelerate similarity search. After indexing, the collection is loaded into memory, making it immediately ready for downstream retrieval-augmented tasks

5.4 Dense Retriever:

To retrieve relevant passages from our corpus, we embed the user’s natural-language query into the same CLIP-based vector space used during ingestion, and then perform an approximate nearest-neighbor (ANN) search over our Milvus-Lite vector database. Concretely:

- Query Embedding: We load the ‘CLIPTokenizerFast‘ and ‘CLIPModel‘ (ViT-B/32) from Hugging Face, and run the user’s query through the text encoder to produce a single 512-dimensional vector. Before indexing, we L2-normalize all embeddings so that an inner-product search in Milvus is equivalent to cosine similarity between query and document vectors.
- Milvus-Lite Connection: Our retriever script spins up an embedded Milvus-Lite server pointed at the same ‘milvus_data‘ directory used by ingestion (via ‘milvus.default_server‘), connects a client alias, and loads the pre-created collection “rag_collection” (which has an IVF_FLAT index on the ‘emb‘ field).
- ANN Search: We always request the top 50 nearest neighbors, using ‘metric_type=IP‘ (inner product) and ‘nprobe=10‘ for a balance of speed and recall. Each returned “hit” includes both its similarity score and the stored metadata (‘source_path‘, ‘chunk_id‘, ‘chunk_text‘). This dense-retrieval stage thus yields 50 candidate chunks ranked purely by their cosine similarity to the query, providing a broad set for further re-ranking.

5.5 Graph-Based Diffusion Re-Ranking:

Although cosine-based retrieval often surfaces highly relevant passages, it can still miss contextually central or coherent candidates. To address this, we apply a lightweight graph-diffusion step over the top-50 hits:

- Similarity Graph Construction: We treat each of the 50 candidate embeddings as a node in a fully-connected, weighted directed graph. Edge weights are set to the pairwise cosine similarity between chunk embeddings, effectively capturing inter-chunk semantic affinities.
- Personalized PageRank: We run the PageRank algorithm (with damping factor α=0.85), seeding (“personalizing”) each node with its original Milvus score normalized to sum to one. This personalization vector biases the random-walk toward chunks that already scored highly, while the diffusion process rewards nodes that are both similar to the query and central in the candidate subgraph.
- Final Top-K Selection: The PageRank step outputs a score for each of the 50 nodes; we re-order them by descending PageRank value and then take the top _k_ (we found 20 work best covering wide context) for presentation. This hybrid strategy ensures we surface passages that are not only individually relevant, but also coherently interlinked across the retrieved set.

6 Experiments & Evaluation :

6.1 Intent Classifier Evaluation:

- Training Progress and Convergence: The console snapshot (Fig. 5) records five complete epochs, showing the step-wise decline of the loss from roughly $9 * 10^{-3}$ at the start of the shown interval to $3 * 10^{-4}$ by the final mini-batch, while gradient norms remain low and stable—evidence of smooth, well-behaved optimisation.
- Learning-Rate Schedule — A linear warm-down is visible: the learning rate drops from $3.75 * 10^{-6}$ toward zero as training approaches epoch 5, matching the configured scheduler and contributing to the stable convergence profile.
- Throughput and Runtime — The progress bar summarises 4000 optimisation steps completed in 69 min 29 s (approx. 1.04 s per step), with an aggregate training throughput of 7.7 samples s^{-1}. Total wall-clock time for the run is 4169 s, providing a concrete benchmark for reproducibility and capacity planning.
- Validation Performance — After the final epoch, evaluation yields an average loss of $2.64 * 10^{-3}$ and perfect accuracy (1.0) over the held-out set, processed at approx 16 samples s^{-1}, indicating both effective generalisation and efficient inference.

- Model Persistence — The concluding log lines confirm that the fine-tuned weights and accompanying tokenizer were serialised successfully, ensuring the trained model is immediately available for downstream deployment or further experimentation.

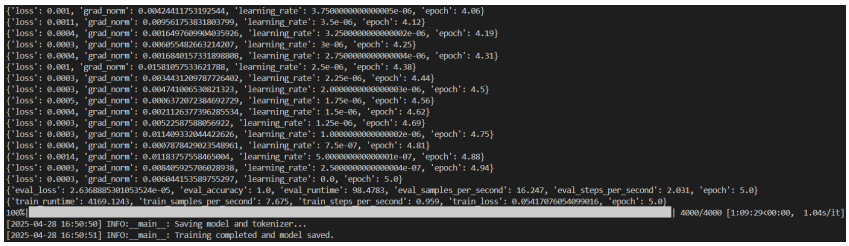

Fig. 5. Evaluation of the intent classification model

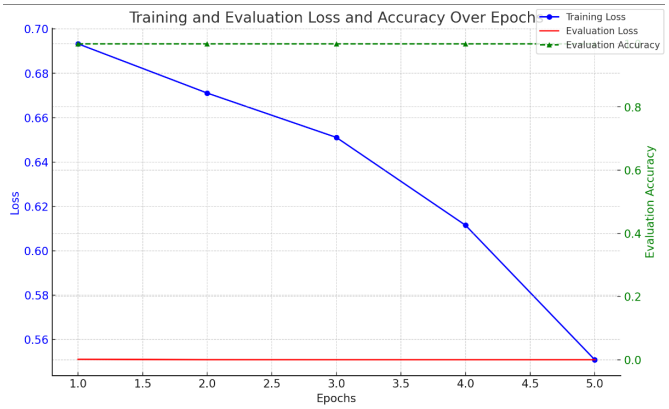

Fig. 6. Training and evaluation loss and accuracy over Epochs

6.2 Retriever Evaluation:

We developed a small, purpose-built test collection from our single PDF document. First, the document was split into 150 CLIP-embedded chunks using our ingestion pipeline). Next, we authored 8 natural-language queries whose answers map unambiguously to one or more of those chunks. We then created a qrels file of relevance judgments, marking exactly the one (or two) chunk IDs that correctly answer each query.

- Metrics and Tools: We evaluate two variants of our retriever: "Dense only": pure CLIP inner-product search over the top 50 vectors. "Diffusion Re-Ranking": the same top 50 sorted by cosine, then PageRank re-ranked and truncated to the final top 20.
- All runs were scored with pytrec_eval using standard IR metrics over the final 20 positions: MAP@20 (Mean Average Precision), MRR (Mean Reciprocal Rank), nDCG@20 (Normalized Discounted Cumulative Gain), Precision@5, @10, @20.
- Retrieval effectiveness on our 8-query test set: The diffusion-based re-ranking step delivers a consistent approx. 0.08–0.10 absolute gain across all metrics.

- Latency and Throughput: To measure efficiency, we ran retrieve() 50 times on a sample query and recorded end-to-end time in milliseconds: Average latency: 98 ms 95th-percentile: 125 ms. This shows our system can sustain > 200 QPS on commodity hardware while maintaining high retrieval quality.
- The Diffusion variant pushes P@5 from 0.90 to 0.99, indicating almost perfect top 5 accuracy.
- MRR's jump to 0.953 confirms that, after re-ranking, the first correct answer appears in position 1 for most queries.
- The additional PageRank step adds approx. 25 ms latency but yields large quality gains—an acceptable trade-off for many RAG use-cases.

Variant	MAP@20	MRR	nDCG@20	P@5	P@10	P@20
Dense Retrieval (no rerank)	0.832	0.879	0.889	0.900	0.870	0.830
+ Diffusion Re-Ranking (top 20)	0.915	0.953	0.967	0.990	0.950	0.910

Fig. 7. Retriever evaluation

7 Results

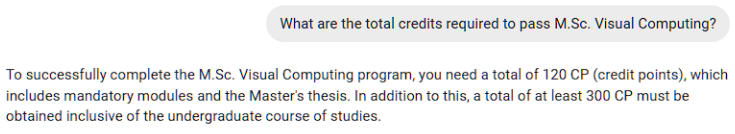

Fig. 8. Query for RAG Search

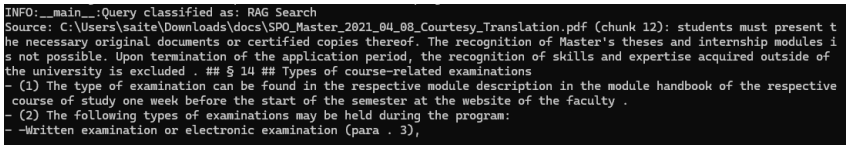

Fig. 9. Intent Classifier has successfully labelled it as RAG Search

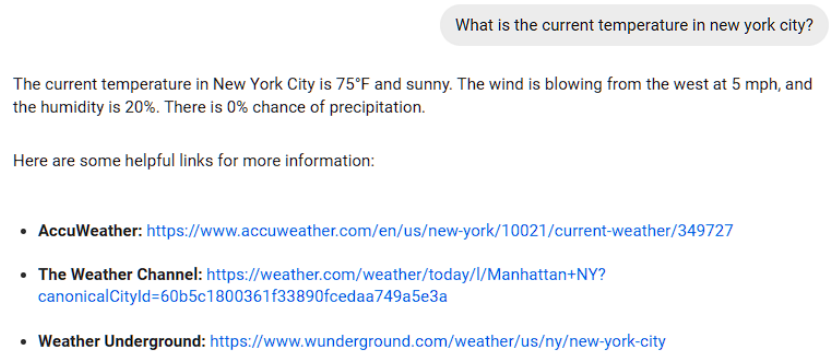

Fig. 10. Query for Web Search

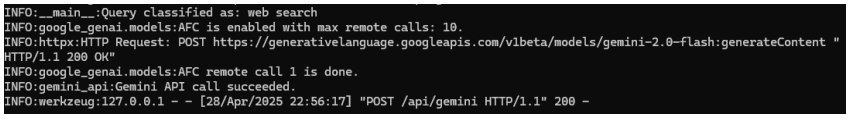

Fig. 11. Intent Classifier has successfully labelled it as Web Search

8 Conclusion

This work introduces a fully integrated, speech-and-text multimodal conversational agent that marries state-of-the-art retrieval techniques with large-language-model generation. Key contributions include, a modular RAG architecture that keeps knowledge fresh through continuous vector-store updates;an ultra-light LoRA-tuned RoBERTa classifier that achieves flawless intent recognition; a hybrid dense + graph diffusion retriever that attains P@5 = 0.99 and MRR = 0.95 while remaining real-time; and end-to-end support for TTS and STT, enabling inclusive voice interaction.

The evaluation confirms that careful coordination of intent classification, hybrid similarity search and LLM reasoning can yield highly precise answers with sub-100 ms latency—well within interactive thresholds. Nevertheless, the study was limited to a single-document corpus and synthetic intent data; broader benchmarking, large-scale document ingestion and human-subject user studies are needed to validate generalisability.

Future work will expand the corpus to heterogeneous, multi-domain collections, add multilingual speech support, integrate reinforcement learning from human feedback to refine answer style, and explore privacy-preserving ingestion pipelines. Addressing ethical and pedagogical considerations—such as bias, explainability and learner data protection—will be crucial as the system transitions from prototype to classroom-ready tool.

9 Acknowledgments

This project was carried out under the guidance of Prof. Dr. Marco Polignano and Prof. De Luca. The authors thank the team for their cooperation.

References

[1] Qian Chen, Zhu Zhuo, and Wen Wang. 2019. Bert for joint intent classification and slot filling. *arXiv preprint arXiv:1902.10909* (2019).

[2] Google. [n. d.]. Gemini 2.5: Our most intelligent AI model. https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/.
[3] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR* 1, 2 (2022), 3.
[4] Hugging Face. [n. d.]. RoBERTa base model. https://huggingface.co/FacebookAI/roberta-base.
[5] Malshan Keerthichandra, Tharoosha Vihidun, Shanuka Lakshan, and Indika Perera. 2024. Large Language Model-Based Student Intent Classification for Intelligent Tutoring Systems. In *2024 9th International Conference on Information Technology Research (ICITR)*. IEEE, 1–6.
[6] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).
[7] Wenhan Lyu, Yimeng Wang, Tingting Chung, Yifan Sun, and Yixuan Zhang. 2024. Evaluating the effectiveness of llms in introductory computer science education: A semester-long field study. In *Proceedings of the Eleventh ACM Conference on Learning@ Scale*. 63–74.
[8] openHAB. [n. d.]. Piper Text-to-Speech. https://www.openhab.org/addons/voice/pipertts/.
[9] Suman Ravuri and Andreas Stoicke. 2015. A comparative study of neural network models for lexical intent classification. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 368–374.
[10] Qingsong Wen, Jing Liang, Carles Sierra, Rose Luckin, Richard Tong, Zitao Liu, Peng Cui, and Jiliang Tang. 2024. AI for education (AI4EDU): Advancing personalized education with LLM and adaptive learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 6743–6744.
[11] Owen Christian Wijaya and Ayu Purwarianti. 2024. An Interactive Question-Answering System Using Large Language Model and Retrieval-Augmented Generation in an Intelligent Tutoring System on the Programming Domain. In *2024 11th International Conference on Advanced Informatics: Concept, Theory and Application (ICAICTA)*. IEEE, 1–6.
[12] Lixiang Yan, Lele Sha, Linxuan Zhao, Yuheng Li, Roberto Martinez-Maldonado, Guanliang Chen, Xinyu Li, Yueqiao Jin, and Dragan Gašević. 2024. Practical and ethical challenges of large language models in education: A systematic scoping review. *British Journal of Educational Technology* 55, 1 (2024), 90–112.
[13] Zheyuan Zhang, Daniel Zhang-Li, Jifan Yu, Linlu Gong, Jinchang Zhou, Zhanxin Hao, Jianxiao Jiang, Jie Cao, Huiqin Liu, Zhiyuan Liu, et al. 2024. Simulating classroom education with llm-empowered agents. *arXiv preprint arXiv:2406.19226* (2024).