Issues in Information Systems
Volume 26, Issue 3, pp. 413-420, 2025

DOI: https://doi.org/10.48009/3_iis_2025_2025_133

Enhancing university education with AI: a Telegram bot leveraging RAG and external APIs for secure knowledge retrieval

Vadim Bashurov, *Comtrade, bashurov@mail.ru*
Paul Safonov, *Saint Cloud State University, safonov@stcloudstate.edu*

Abstract

This paper presents a novel AI-powered Telegram bot designed to enhance university information services by securely integrating external AI capabilities with institutional private data. The system leverages Retrieval-Augmented Generation (RAG) to transform structured university data (faculty profiles, schedules, lecture notes) into vectorized embeddings, which are dynamically retrieved and combined with responses from a general-purpose AI API (e.g., GPT-4). This hybrid approach ensures accurate, context-aware answers while preserving data privacy — raw institutional information is never exposed directly to third-party systems. Implemented at Comtrade University, the bot demonstrates significant outperforming standalone AI models for domain-specific questions. Key innovations include a scalable pipeline for embedding private data, seamless Telegram-based access, and cost-efficient prompt engineering via RAG. The solution addresses critical challenges in educational technology: balancing AI augmentation with data security and providing 24/7 conversational access to institutional knowledge. We discuss architectural decisions, privacy safeguards, and empirical results, offering a replicable framework for other universities.

Keywords: retrieval-augmented generation (RAG), educational chatbots, telegram API, LLM vector embeddings, hybrid AI systems, privacy in EdTech

Introduction

The integration of artificial intelligence (AI) into higher education has accelerated in recent years, offering transformative potential for administrative efficiency, personalized learning, and student engagement (Baker et al., 2021). However, two critical challenges persist:

1. **fragmented access** to institutional knowledge (e.g., schedules, faculty contacts, lecture materials), often siloed across disparate platforms
2. **privacy risks** associated with deploying third-party AI tools on sensitive university data.

While large language models (LLMs) (Blank, 2023) like GPT-4 excel at general-purpose tasks, their inability to natively access private, domain-specific information limits their utility in educational contexts. To bridge this gap, we present a secure, scalable solution: a Telegram-bot that dynamically combines the generative capabilities of external AI APIs with a university's private data through Retrieval-Augmented Generation (RAG) (Lewis et al., 2020). Our system transforms structured institutional data (faculty directories, course schedules, lecture notes) into vector embeddings, which are retrieved contextually during user interactions and fused with AI-generated responses. This approach ensures real-time accuracy.

For example, answering "What are Professor Kevin Floyd's office hours?" with data drawn exclusively from the university's internal records while preventing raw data exposure. In this study, we will use MySQL requests, PHP code, and Python code to demonstrate how to apply the RAG algorithm.

What is RAG?

Retrieval-Augmented Generation (RAG) is a technique that enhances large language models (LLMs) by integrating external data retrieval with text generation. Unlike traditional LLMs, which rely solely on static, pre-trained knowledge, RAG combines a retrieval mechanism that fetches relevant information from external sources (e.g., documents, databases, or the web) with a generative model to produce accurate, contextually relevant, and up-to-date responses. This approach mitigates common LLM limitations such as outdated knowledge, hallucinations (generating incorrect information), and lack of domain-specific context, making it ideal for applications like our Telegram bot, where precise and current information is crucial.

The RAG pipeline typically involves three key components:

- **Retrieval**: A retriever searches a local knowledge base (e.g., a vector database) to find relevant documents or data for a user query.
- **Augmentation**: The retrieved information is combined with the user's query to create an enriched prompt.
- **Generation**: The LLM uses the augmented prompt to generate a coherent and contextually grounded response.

The term "Retrieval-Augmented Generation" was coined in 2020 by a team at Facebook AI Research (now Meta AI), led by Patrick Lewis (Lewis et al., 2020). Their paper introduced RAG as a framework to improve LLMs for knowledge-intensive tasks. Since the 2020 paper, RAG has evolved significantly, with advancements in retrieval mechanisms, model architecture, and applications. REALM (Retrieval-Augmented Language Model Pre-Training) (Guu et al., 2020) introduced a fully dynamic RAG model where the retriever, generator, and document encoder are updated during training, improving end-to-end performance. This work explored pre-training strategies to enhance retrieval-augmented models.

In-Context Retrieval-Augmented Language Models (Ram et al., 2023) introduced a re-ranker to prioritize retrieved results before feeding them into the LLM, improving contextual relevance. The Survey categorized RAG into Naive, Advanced, and Modular paradigms, detailing advancements in retrieval, generation, and augmentation techniques. Another survey (Gao et al., 2024) is a comprehensive resource for understanding RAG's progression. Next comprehensive survey of RAG (Gupta et al., 2024) explored innovations like graph-based RAG and applications in domains such as healthcare, finance, and education. It highlighted challenges like scalability and bias mitigation. Paper (Singh et al., 2025) introduced Agentic RAG, which embeds autonomous AI agents into the RAG pipeline for dynamic retrieval strategies and multi-step reasoning.

Vectorization of Textual Documents for Retrieval-Augmented Generation

In the Retrieval-Augmented Generation (RAG) pipeline, the first step involves transforming textual data into dense numerical vectors that capture semantic meaning. This is critical to enabling the system to retrieve relevant information from private documents in response to a user query. Let the set of documents be denoted by $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$, where each d_i is a semantically meaningful text chunk (e.g., a

paragraph). Using a pretrained embedding model $f: \mathcal{T} \rightarrow \mathbb{R}^m$, such as OpenAI's text-**embedding-ada-002** each text chunk is mapped to a vector:

$$\mathbf{v}_i = f(d_i), \mathbf{v}_i \in \mathbb{R}^m, \forall i \in \{1, \ldots, N\}$$

These vectors $\mathbf{v}_i$ are stored in a MySQL table where each row contains the corresponding document ID and its vector embedding components. Though MySQL is not optimized for approximate nearest neighbor search, cosine similarity or inner product search can still be performed via SQL queries or external vector search wrappers. When the user submits a query$q \in \mathcal{T}$, it is also embedded:

$$\mathbf{v}_q = f(q)$$

To find the most relevant documents, we compute the cosine similarity between the query vector $\mathbf{v}_q$ and each stored document vector $\mathbf{v}_i$:

$$\text{sim}(\mathbf{v}_q, \mathbf{v}_i) = \frac{\mathbf{v}_q \cdot \mathbf{v}_i}{\| \mathbf{v}_q \| \cdot \| \mathbf{v}_i \|}$$

We then select the top-k documents with the highest similarity scores:

$$\mathcal{R}(q) = \{d_{i_1}, d_{i_2}, \ldots, d_{i_k}\} \text{where sim}(\mathbf{v}_q, \mathbf{v}_{i_j}) \text{is maximal}$$

The retrieved documents are concatenated with the original query to form a new prompt *P* that is passed to the ChatGPT model:

$$P = \text{concat}(q, \mathcal{R}(q))$$

This methodology allows the system to generate context-aware, factually grounded responses based not only on pretrained model knowledge but also on private or domain-specific data. The result is an interactive agent capable of referencing real-time or proprietary content without needing to retrain the LLM.

Algorithm for Semantic Vectorization

A fundamental requirement in retrieval-augmented generation (RAG) and other semantic search systems is the ability to represent text in a machine-readable format that captures its meaning. OpenAI's `text-embedding-ada-002` is a state-of-the-art embedding model designed precisely for this task. It transforms natural language input into high-dimensional numerical vectors that preserve semantic similarity, enabling downstream applications such as document retrieval, classification, clustering, and question-answering.
The algorithm operates by encoding each input text sequence into a fixed-size dense vector $\mathbf{v} \in \mathbb{R}^m$, where $m = 1536$ for this model. Formally, given a text string $t \in \mathcal{T}$, the embedding function is defined as:

$$f(t) = \mathbf{v}, \text{ where } \mathbf{v} \in \mathbb{R}^{1536}$$

The embedding space is structured such that semantically similar texts yield vectors that are close under cosine similarity. For example, the embeddings of "How to train a dog" and "Tips for dog training" will reside near each other in this high-dimensional space, even if they have few words in common.

The architecture behind `text-embedding-ada-002` is based on a transformer neural network trained using contrastive learning techniques on a massive corpus of web and document data. This training paradigm encourages the model to map semantically related texts to nearby points in the embedding space while pushing dissimilar texts further apart. A critical property of this embedding model is that it is domain-general: it can process a wide variety of texts, including scientific documents, news articles, emails, code snippets, and casual conversations. This universality is advantageous in applications where the underlying data sources are heterogeneous or unstructured.

In practical deployment, embeddings generated by `text-embedding-ada-002` can be stored in a vector database or a relational database (e.g., MySQL) and indexed using similarity search techniques. When a user issues a query q, the system computes its embedding $\mathbf{v}_q = f(q)$ and retrieves the top-k vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ from the dataset that maximize the cosine similarity:

$$\text{sim}(\mathbf{v}_q, \mathbf{v}_i) = \frac{\mathbf{v}_q \cdot \mathbf{v}_i}{\| \mathbf{v}_q \| \cdot \| \mathbf{v}_i \|}$$

These retrieved vectors correspond to the most relevant documents or passages, which can then be used as input context to a large language model (LLM) such as ChatGPT. This enables the LLM to generate responses that are both coherent and grounded in external, user-specified data.In summary, `text-embedding-ada-002` offers a powerful and general-purpose tool for transforming unstructured textual data into structured semantic vectors. Its integration within RAG pipelines facilitates intelligent information retrieval, contextual augmentation, and the construction of AI assistants that can reason over private corpora without fine-tuning the underlying language model.

Telegram Bot Deployment and Integration

Here is a step-by-step guide to creating and connecting the Comtrade 360 Telegram Bot:
We created the bot using Telegram service BotFather and called it Comtrade 360 (this is the public name users will see). Now the PHP/Python/JS scripts can control the bot using API token.

How Messages Flow

- A student asks Comtrade 360: "When is the next math lecture?"
- Telegram sends this question to `bot_processor.php` via the webhook.
- The PHP script searches the university's vector database (RAG) for lecture schedules and combines the results with an AI (using API ChatGPT) to generate a clear answer.
- The bot replies: "Math 101 is tomorrow at 10 AM in Room 205."

Replies happen instantly (no delay). Local server isn't constantly asking Telegram, "Any new messages?" and handles thousands of students without crashing. This method powers Comtrade 360 to act like a 24/7 digital assistant for students.

RAG Implementation with MySQL and LLMs

Comtrade 360 bot uses Retrieval-Augmented Generation (RAG) to answer questions by combining private university data (stored in MySQL) with external AI knowledge. Here's how it works:

1. Vectorizing Private Data

- Preprocess Comtrade's private documents (schedules, faculty info) into clean text chunks.
- Use an LLM embedding model (e.g., OpenAI's `text-embedding-3-small`) to convert each chunk into a vector (a list of numbers representing its meaning).
- Store vectors in MySQL database.

2. Retrieving Relevant Information

When a user asks the question, "Who is Professor Kevin Floyd?":

- Embed the query: Convert the query to a vector using the same LLM (e.g., OpenAI's `text-embedding-3-small`).
- Search MySQL for the most relevant chunks by comparing the query to stored vectors using cosine similarity (how "aligned" two vectors are):
- Similarity Score $= cos(\theta) = \frac{v_q \cdot v_t}{\|v_q\| \|v_t\|}$, where $\| v \| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$
- Pass results + query to the external AI (using API GPT-4) to generate a natural-language answer:

Key Advantages

- Privacy: Raw data never leaves your MySQL server—only vectors are compared.
- Accuracy: Combines Comtrade's exact data with AI's language skills.

Example:

- User Query: "When is Prof. Floyd's office hour?"
- RAG Finds: MySQL vector search retrieves "Prof. Floyd: Wed 2-4 PM, Room 305".
- AI Polishes: Bot replies: "Prof. Floyd's office hours are Wednesdays from 2-4 PM in Room 305"

Text Chunking for RAG Implementation

To ensure high-quality vector searches in MySQL, Comtrade's private data (schedules, faculty info, etc.) is split into meaningful chunks before embedding. Here's the step-by-step process:

Chunking Rules

We prioritize semantic coherence (keeping related ideas together) and practical searchability (avoiding overly long/short chunks):

Rule 1: *Sentence Splitting*

Break documents at natural language boundaries:

- Split after periods/exclamation marks only if the next sentence starts with a capital letter.
- Never split mid-acronym (e.g., "B.Sc." stays intact).

Rule 2: Fixed-Size Overlapping Windows

For long paragraphs:

- Chunk size: 128 words (optimized for LLM embedding limits).
- Overlap: 25 words between chunks to preserve context.

Rule 3: Structural Preservation

- Tables (e.g., schedules) → Keep entire rows together.
- Faculty bios → One chunk per "section" (Education, Office Hours, Research).

Why this works for Comtrade 360:

- Precision: Overlapping chunks ensure queries like "Where is Prof. Floyd on Wednesdays?" retrieve relevant office hours even if split across chunks.

- Speed: Fixed-size chunks optimize MySQL's indexing (e.g., using `FASTTEXT` indexes for vector searches).
- Context: Structural rules preserve relationships (e.g., a faculty member's name + office hours stay together).

Visualization of Chunking

Original Document:

> "Prof. Floyd (Math Dept) teaches Calculus. Office Hours: Wed 2-4 PM, Room 305. Research: Algebra."

Chunks:

- "Prof. Floyd (Math Dept) teaches Calculus. Office Hours: Wed 2-4 PM, Room 305."
- "Office Hours: Wed 2-4 PM, Room 305. Research: Algebra." ← 25-word overlap

Measuring the accuracy

Measuring the accuracy of a Telegram Retrieval-Augmented Generation (RAG) bot involves evaluating both the retrieval and generation components to ensure the bot provides relevant, accurate, and contextually appropriate responses.

A RAG system combines two key processes:

- Retrieval: The bot searches a local knowledge database to find relevant information based on the user's query.
- Generation: The bot uses a large language model (LLM) to generate a natural language response based on the retrieved information.

Accuracy depends on how well the retrieval system identifies relevant documents and how effectively the LLM generates responses using that information. We evaluate both components separately and together. To measure accuracy, we use a combination of retrieval and generation metrics tailored to our bot's use case. Below are key metrics to consider.

Retrieval Metrics

These assess how well the bot retrieves relevant documents or snippets:

- **Precision**: The proportion of retrieved documents that are relevant to the query.
- *Precision = (Number of relevant documents retrieved) / (Total number of documents retrieved)*
- **Recall**: The proportion of relevant documents retrieved out of all relevant documents in the knowledge base.
- *Recall = (Number of relevant documents retrieved) / (Total number of relevant documents)*
- **Context Relevance**: Measures how well the retrieved content matches the query's intent. Tools like RAGAs can quantify this by analyzing semantic similarity between the query and retrieved documents.

Also, we collected user feedback (e.g., thumbs-up/down ratings or surveys) to gauge perceived accuracy and usefulness. To systematically measure accuracy, set up a testing framework:

1. Preparation of a Test Dataset:

- Create a set of queries with known ground-truth answers. Include a variety of question types (e.g., factual, open-ended, multi-intent) that reflect real user interactions.
- Ensure the dataset covers edge cases, ambiguous queries, and scenarios where the knowledge base may lack information.

2. Manual Evaluation:

- For a small dataset, manually review the bot's responses to assess factual accuracy, relevance, and completeness.
- Use a scoring rubric (e.g., 0-100 scale) for metrics like correctness and context adherence.

Results

The Comtrade 360 Telegram bot has been successfully deployed and is now publicly available for anyone to use. By integrating Retrieval-Augmented Generation (RAG) with MySQL and external AI APIs, the bot provides accurate, real-time responses to user queries while maintaining data privacy.

Key Outcomes:

1. User Accessibility:

- Open to all users via Telegram: Simply search for Comtrade 360 to start interacting.
- No installation required—works on any device with Telegram installed.

2. Performance Metrics:

- high accuracy (over 90%) in retrieving correct information from Comtrade's private database.
- Less than 2-second response time for most queries.

3. Scalability:

- Handles hundreds of concurrent users without performance degradation.
- Easily updatable—new data can be added to MySQL and re-embedded without downtime.

4. Privacy Compliance:

- Sensitive data remains secure; only vectorized embeddings are processed during searches.

How to Use the Bot:

- Open Telegram and search for **@Comtrade360_bot**.
- Type your question (e.g., *"When is the next faculty meeting?"*).
- Receive an instant, AI-augmented answer based on Comtrade's official data.

Discussion

Talking about pros and cons of the proposed approach we should make some remarks:

Advantages:

- Cost-effective: Telegram is free; RAG reduces API costs (shorter prompts), cost of API ChatGPT is 20$ per month.
- Scalable: Easy to update vector DB with new lectures/faculty.
- Privacy-compliant: No PII leaks; access control via Telegram auth.

Limitations:

- Dependency on external API (latency/costs).
- Cold start for vector DB setup.

The next steps in this further development could embrace:

- Integrating multimodal data (PDFs, slides).
- Experimenting with local LLMs (e.g., LLaMA-3) to replace external APIs.

References

Baker, R. S., & Hawn, A. (2021). Algorithmic Bias in Education. *International Journal of Artificial Intelligence in Education*.

Blank, I. A., (2023). What are large language models supposed to model? Trends in Cognitive Sciences. 27 (11): 987–989.

Gao, A. K. (2023). *Vec2Vec: A Compact Neural Network Approach for Transforming Text Embeddings with High Fidelity*.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., et al. (2020). Retrieval- augmented generation for knowledge-intensive nlp tasks. Advances in Neural Information Processing Systems. 33: 9459–9474.

Bashurov, V. & Safonov, P. (2023). Anomaly detection in network traffic using entropy-based methods: application to various types of cyberattacks. Issues in Information Systems. 24 (4): 82-94.

Guu, K., Lee, K., Tung, Z. & Pasupat, P. (2020). REALM: Retrieval-Augmented Language Model Pre-Training. Proceedings of the 37th International Conference on Machine Learning. 368: 3929 – 393.

Ram, O., Levine, Y., Dalmedigos, I., Muhlgay, D., Shashua, A., Leyton-Brown, K. & Shoham, Y. (2023). In-Context Retrieval-Augmented Language Models. Transactions of the Association for Computational Linguistics, 11:1316–1331.

Gao, Y., et al. (2024). Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997

Gupta, S., Ranjan, R. &Singh, S. (2024). A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions. arXiv:2410.12837

Singh, A., Ehtesham, A., Kumar, S. & Khoei, T. (2025). Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG. arXiv:2501.09136