

Multi-Provider RAG Architecture for Flexible Educational Chatbots¹

Mariia Tiahunova^{1*}, Halyna Kyrychek¹ and Kyryll Filippenkov¹

¹National University Zaporizhzhia Polytechnic, 64 Universytetskyi Str., Zaporizhzhia, 69063, Ukraine

Abstract

The increasing adoption of Large Language Models (LLMs) in education creates new opportunities for intelligent tutoring, question answering, and access to academic information. However, reliance on a single provider introduces vendor lock-in, operational costs, data-privacy concerns, service availability risks, and dependence on external connectivity, which are particularly significant for institutions operating under financial, regulatory, or infrastructure constraints. This paper proposes a multi-provider Retrieval-Augmented Generation (RAG) architecture for educational chatbots that integrates local and cloud-based LLM and embedding providers through a unified abstraction layer. The proposed architecture enables runtime interchangeability of generation and embedding back-ends with transparent capability fallback, reducing dependence on individual vendors and improving operational resilience. The RAG pipeline is formalized as a parameterized architecture in which document chunking, overlap, retrieval depth, and model selection are treated as configurable controls that can influence both system performance and pedagogical outcomes. The scientific novelty lies in the provider-independent abstraction model, parameterized RAG pipeline, and integrated cost and deployment analysis, rather than in the use of specific frameworks, vector databases, or individual LLM APIs. These components are considered interchangeable implementation blocks, ensuring that the proposed approach remains independent of particular vendors and technologies. The architecture supports fully local, cloud, and hybrid deployment scenarios and is evaluated through Computer Science and STEM education use cases. The analysis demonstrates the trade-offs among cost efficiency, data privacy, availability, computational requirements, and response quality, providing a basis for selecting an appropriate deployment configuration. The proposed architecture therefore offers an extensible and vendor-independent foundation for scalable, resilient, and adaptable RAG-based educational chatbots.

Keywords

chunking strategies, retrieval depth, model selection, STEM education, LLM frameworks

1. Introduction

Educational institutions worldwide are increasingly adopting Large Language Models (LLMs) for AI-powered tutors, question-answering assistants and documentation helpers. Yet most deployed systems rely on a single provider, which creates three critical problems. First, economic constraints: at the scale of a medium-sized university, per-token cloud pricing and peak loads make cloud-only solutions expensive. Second, data privacy and compliance: sending student queries and course materials to third-party servers may conflict with regulations such as GDPR or institutional policies requiring on-premise processing. Third, infrastructure and connectivity: institutions with unstable or costly internet may find cloud-dependent systems unreliable, especially at peak times or in crisis conditions [1]. These issues are particularly acute for universities in countries facing economic or infrastructure challenges, where budgets for commercial cloud APIs are limited and local hosting is

¹ADT-ISMDCC 2026: Advanced Digital Technologies: Intelligent Systems, Modeling, and Data-Driven Control, September 22–24, 2026, Odesa, Ukraine

^{1*} Corresponding author.

✉ mary.tyagunova@gmail.com (M. Tiahunova); kirgal08@gmail.com (H. Kyrychek); filippenkovkyryll@gmail.com (K. Filippenkov)

ORCID: 0000-0002-9166-5897 (M. Tiahunova); 0000-0002-0405-7122 (H. Kyrychek); 0009-0008-6720-5728 (K. Filippenkov)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

often preferred. At the same time, many educators are interested in leveraging high-end cloud models when possible, for example in advanced courses or research-oriented seminars.

To address this tension, the paper proposes a conceptual multi-provider RAG architecture that decouples the educational chatbot from any single LLM vendor. Instead of being hard-wired to a specific API, the system exposes an abstract interface for LLMs and embeddings, with concrete implementations for local models (e.g., via on-device or on-premise runtimes) and for major cloud providers such as OpenAI, Anthropic, and Google [2-4]. The retrieval pipeline is similarly parameterized: designers can adjust how documents are chunked, how many retrieved segments are considered, and how strongly chat history influences query reformulation, based on pedagogical goals and available compute.

The contributions of this paper are conceptual and architectural:

- provider-agnostic RAG architecture for education;
- provider abstraction for runtime switching;
- retrieval and text-splitting configuration;
- educational use cases and deployment scenarios;
- empirical cost analysis across institution sizes;
- open research questions for evaluation.

After outlining the overall blueprint, the paper explains how the abstraction layer hides vendor-specific SDK details, how retrieval parameters influence educational quality, and how institutions can combine local and cloud models depending on cost, privacy, and connectivity constraints.

The proposed approach introduces a vendor-independent architectural model for multi-provider RAG-based educational chatbots. Its core contribution is a unified provider-abstraction layer that enables runtime interchangeability of LLM and embedding back-ends while supporting transparent capability-based fallback. The RAG workflow is formalized as a parameterized pipeline in which document chunking, chunk overlap, and retrieval depth are explicitly configurable and can be adjusted to the requirements of educational content and learning tasks. In addition, the architecture incorporates a quantitative cost and deployment analysis that supports the selection of local, cloud, or hybrid configurations according to computational resources, operational costs, privacy requirements, and availability constraints. This combination of architectural abstraction, configurable retrieval parameters, and deployment-oriented analysis provides a flexible foundation that is independent of specific LLM vendors, APIs, or software frameworks.

2. Theory and related works

2.1. Retrieval-Augmented Generation and Knowledge-Intensive Tasks

Retrieval-Augmented Generation improves LLM performance on knowledge-intensive tasks by complementing parametric memory with external documents. Lewis et al. showed that combining retrieval with generation maintains factual accuracy and reduces hallucinations in open-domain question answering [5], and later work extended this to scientific literature and technical documentation [6]. In education, RAG is attractive because it grounds answers in course materials and supports references to specific passages that students can revisit [7], aligning with good pedagogical practice. However, the effectiveness of RAG in knowledge-intensive tasks depends not

only on the availability of external knowledge but also on retrieval quality, document segmentation, retrieval depth, and the capability of the underlying generation model.

2.2. LLM Providers, Model Diversity, and Comparative Studies

A growing body of work examines differences among LLM providers in terms of reasoning performance, robustness, and cost. Technical reports and empirical evaluations of OpenAI’s GPT-4 document strong performance on coding, reasoning, and language understanding tasks, albeit at relatively high per-token prices [2]. Anthropic’s Claude family focuses on constitutional AI and safety, offering competitive reasoning capabilities and long context windows [3]. Google’s Gemini models emphasize multimodal capabilities and cost-efficient scaling across different sizes [4]. Independent benchmarking efforts have begun to compare these families across tasks relevant to education, such as mathematical reasoning, code generation, and reading comprehension [8]. This diversity suggests that model selection should be treated as a configurable architectural decision rather than a fixed implementation choice, particularly when performance, cost, availability, and deployment constraints vary across educational scenarios. Most existing systems, however, choose a single “best” model or provider for their use case. While this simplifies implementation, it amplifies the risk of vendor lock-in and does not leverage the diversity of strengths across models. Our work instead advocates a multi-provider perspective, in which the architecture is explicitly designed to accommodate heterogeneous providers and to support dynamic switching between them.

2.3. Educational Chatbots and Technology-Enhanced Learning

Educational chatbots and intelligent tutoring systems have been studied for decades, predating large-scale neural LLMs. Earlier systems focused on tightly scripted dialogue, rule-based reasoning, or domain-specific models [1, 7]. With the advent of deep learning, research shifted toward data-driven dialogue systems and tutoring agents that can generate more natural, flexible responses [9]. Recent work has explored ChatGPT-like systems for tasks such as feedback on student writing, formative assessment, and concept explanations in STEM fields [10]. Systematic reviews emphasize that successful educational chatbots share several characteristics: they are integrated with course materials, support follow-up questioning, foster metacognitive reflection, and provide transparent references to underlying content [10]. At the same time, concerns have been raised about over-reliance on opaque AI systems, potential biases in training data, and the risk of shallow learning if students simply copy AI-generated answers. Our architecture responds to these concerns in two ways. First, by adopting a RAG approach, it ensures that answers can be traced back to specific educational materials. Second, by promoting a configurable multi-provider design, it enables institutions to select models and deployment strategies that align with their ethical, legal, and pedagogical requirements.

2.4. Design Principles and High-Level Structure

The proposed architecture follows a layered design that separates user interaction, retrieval and reasoning, and provider integration, as illustrated in Figure 1, where each numbered block denotes a functional element. A student works through the User Interface Layer (first block – chat area, configuration panel, history management and help panel). Every turn is sent, together with the running conversation history, to the RAG Pipeline Orchestrator (second block), which performs history-aware query reformulation. The reformulated query is matched against the Vector Store (third block, FAISS), where course materials – previously converted into embeddings by the Document Ingestion branch (sixth block: load → split → embed) – are searched to return the top-k relevant segments. The retrieved context and the query are passed to the Provider Abstraction Layer (fourth block), which forwards them to the selected local or cloud model; the grounded Response with its source citations (fifth block) is returned to the interface for display. Through the interface, instructors and students can ingest heterogeneous course documents – lecture slides and

presentations (PDF, PPTX), text and word-processing files (TXT, DOCX) and web pages (URL) – which may themselves contain diagrams, figures, graphs and tables. The ingestion pipeline extracts the textual and tabular content of these files; interpretation of purely graphical elements (figures, hand-drawings, plots) is currently text-based and is revisited among the limitations.

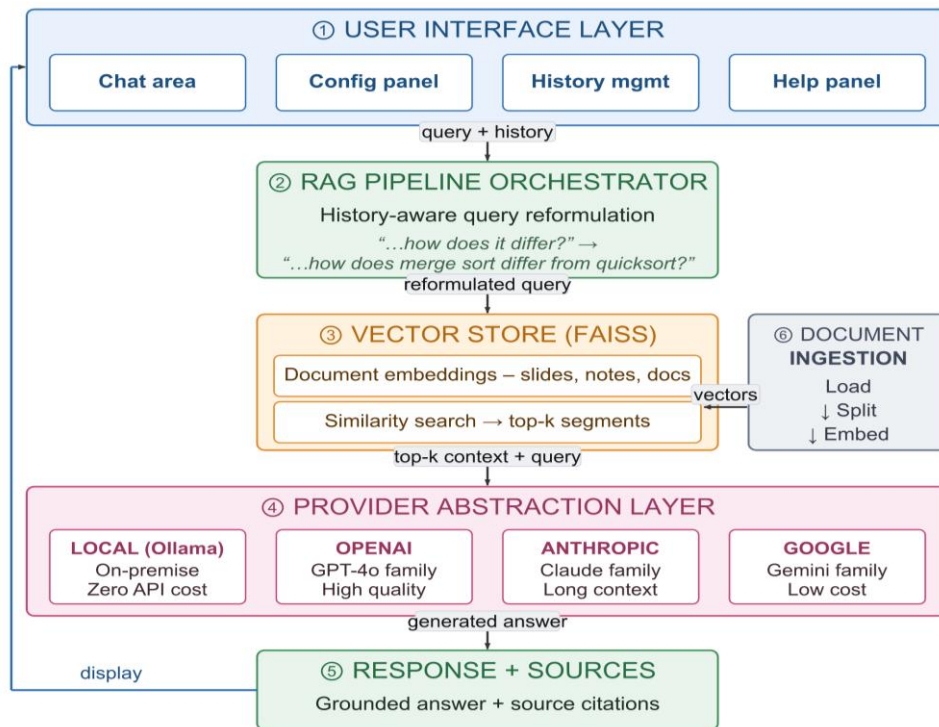


Figure 1: High-level architecture of the multi-provider RAG educational chatbot

The provider abstraction decouples LLM and embedding components from specific vendors, while configuration-driven control enables provider, model, and retrieval parameters to be adjusted dynamically at runtime. Caching targets expensive embedding and indexing steps, and history-aware interaction improves retrieval for follow-up questions that depend on previous turns. Modularity allows UI, retrieval, and provider integrations to evolve independently. Although our discussion is grounded in common Python ecosystems such as LangChain, FAISS, and Streamlit [11–13], the architectural pattern is general and can be realized using alternative toolkits.

2.5. Provider Abstraction Layer

At the heart of the architecture lies a provider abstraction layer that defines a uniform interface for LLMs and embeddings; its structure is shown as a UML-style class diagram in Figure 2. The Provider enumeration (top-left) lists the supported back-ends (LOCAL, OPENAI, ANTHROPIC, GOOGLE) and acts as the system-wide selection key, while APIConfig (top-right) stores credentials and exposes `is_provider_configured()` and `get_available_providers()`. The central ProviderFactory uses both classes and creates objects that conform to two interfaces, BaseLLM (with `invoke()`) and Embeddings (with `embed_query()/embed_documents()`). The concrete adapters at the bottom – OllamaLLM, ChatOpenAI, ChatAnthropic and ChatGoogleGenerativeAI for generation, and OllamaEmbeddings, OpenAIEmbeddings and GoogleGenerativeAIEmbeddings for embeddings – implement these interfaces. The dashed link at the bottom of Figure 2 marks the capability fallback: because Anthropic exposes no embedding model, the factory transparently substitutes OpenAI embeddings. This design lets institutions mix and match generation and embedding components based on availability and performance without restructuring the application: factories encapsulate credentials and defaults, and the capability fallback covers functional gaps such as a missing embedding model for a given vendor.

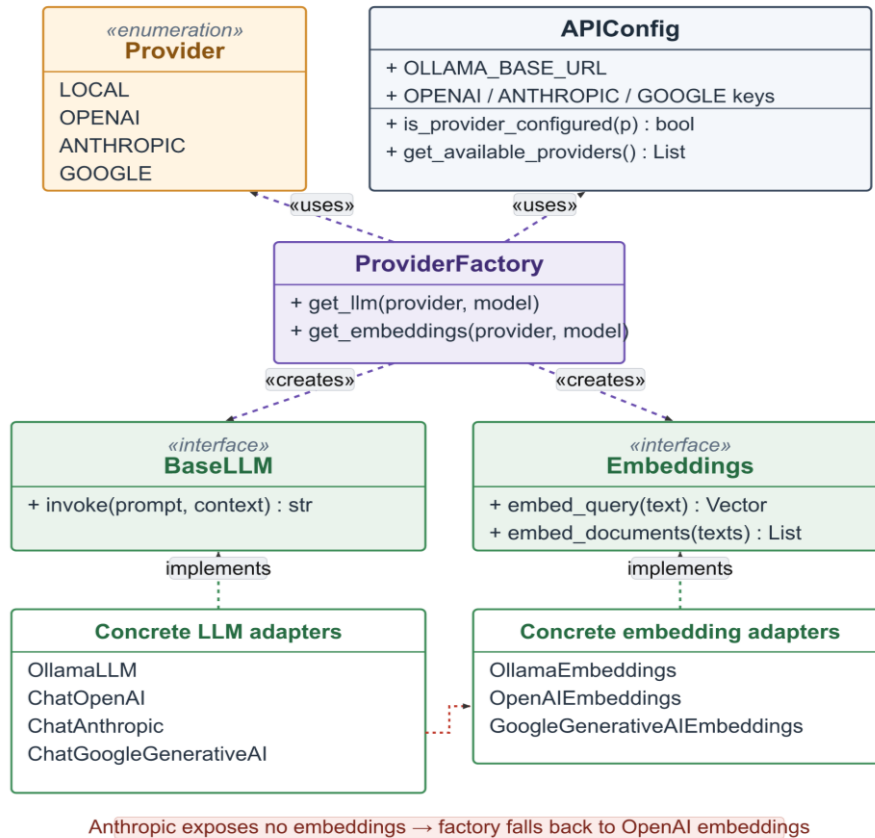


Figure 2: Provider abstraction and factory pattern used in the system

At runtime, provider selection can therefore be adapted to changing service availability, computational constraints, or application requirements without modifying the retrieval and generation logic. This separation of concerns also facilitates the integration of additional providers and models, allowing the architecture to evolve without introducing provider-specific dependencies into the core application layer.

2.6. Retrieval Pipeline and Text-Splitting Strategy

The retrieval pipeline typically loads documents, splits them into smaller segments, encodes those segments as vectors and performs similarity search [5, 11, 14]. Rather than fixing numeric values, the conceptual architecture proposed by the authors emphasizes the influence and trade-offs of key parameters. Text-splitting granularity trades precision for context: smaller chunks match targeted questions, while larger ones preserve surrounding explanation. Overlap mitigates boundary artifacts at the cost of storage; retrieval depth balances completeness against noise; and similarity metrics can be combined with metadata filters (source, difficulty, recency) for curriculum alignment. Exposing these as configuration rather than hard-coded values lets teachers or administrators tune retrieval to course level, assessment type and available compute. The RAG pipeline consists of sequential indexing, retrieval, and response generation stages (Figure 3). During indexing, the document collection is loaded and divided into overlapping text chunks of a predefined size. Each chunk is converted into a vector representation using an embedding model and stored in a FAISS index, enabling efficient semantic retrieval. At query time, the current question is processed together with the conversation history and, when necessary, reformulated into a standalone query. The reformulated query is then used to retrieve the most relevant document segments from the vector index. The retrieved segments, together with the query and dialogue context, are provided to the language model, which generates the final response. The main configurable parameters are chunk size, chunk overlap, and the number of retrieved segments. These parameters determine the index

structure, retrieval coverage, and the amount of contextual information provided to the generative model. Thus, pipeline configuration defines a trade-off between accuracy, completeness, efficiency, and computational cost of the RAG system.

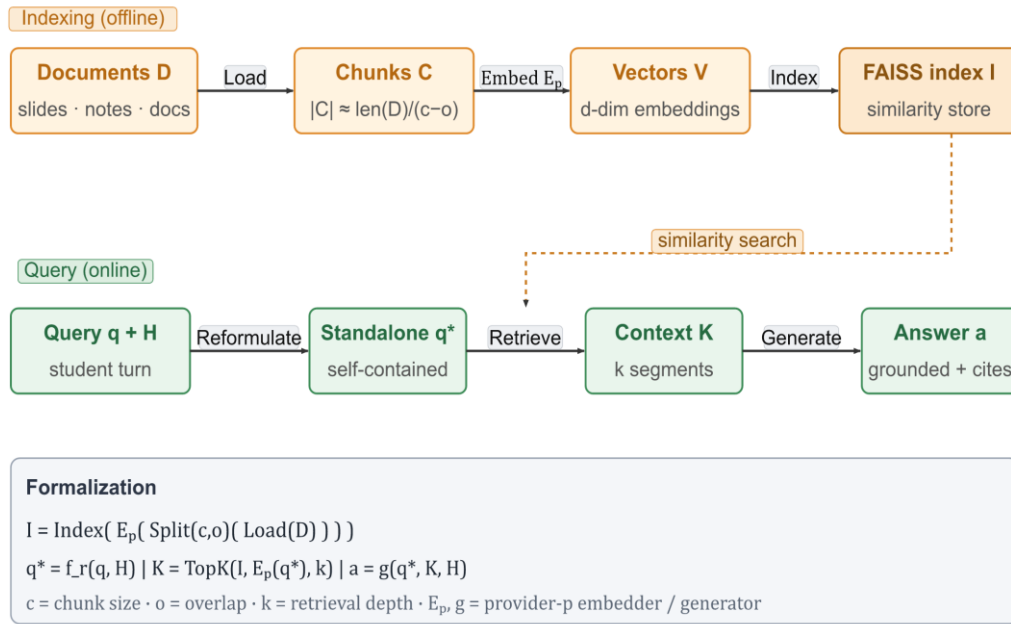


Figure 3: Formal model of the indexing (offline) and query (online) pipeline processes

2.7. History-Aware Question Reformulation

Students rarely ask questions in isolation; instead, they build on previous dialogue turns. A retrieval system that processes each question independently may fail to interpret such contextual references correctly, leading to irrelevant retrieval and confusing answers. To address this, the architecture incorporates a history-aware reformulation step. Before querying the document store, the system reformulates the latest user question into a standalone query that explicitly encodes the relevant context from prior turns [5, 11]. For example, if the conversation previously discussed sorting algorithms and the student now asks “How is it different in terms of time complexity?”, the reformulated query might become “How is merge sort different from quicksort in terms of time complexity?”. The effectiveness of this step depends on how much history is considered, how the prompt guiding reformulation is designed, and which LLM is used for reformulation.

2.8. User Interface Layer

From the student’s perspective, interaction occurs through a web-based chat interface (the implemented user interface is illustrated in Figure 4) that typically includes: main chat area; configuration panel; history management; help panel.

The chat area presents turns clearly, the configuration panel exposes provider/model and data-source selection, history management supports export/import for learning activities, and the help panel offers prompt templates to guide students. The interface can also present source citations for each answer, showing snippets of retrieved segments and their metadata. This supports transparency, encourages students to verify answers by consulting the original material, and aligns with the broader goals of academic literacy.

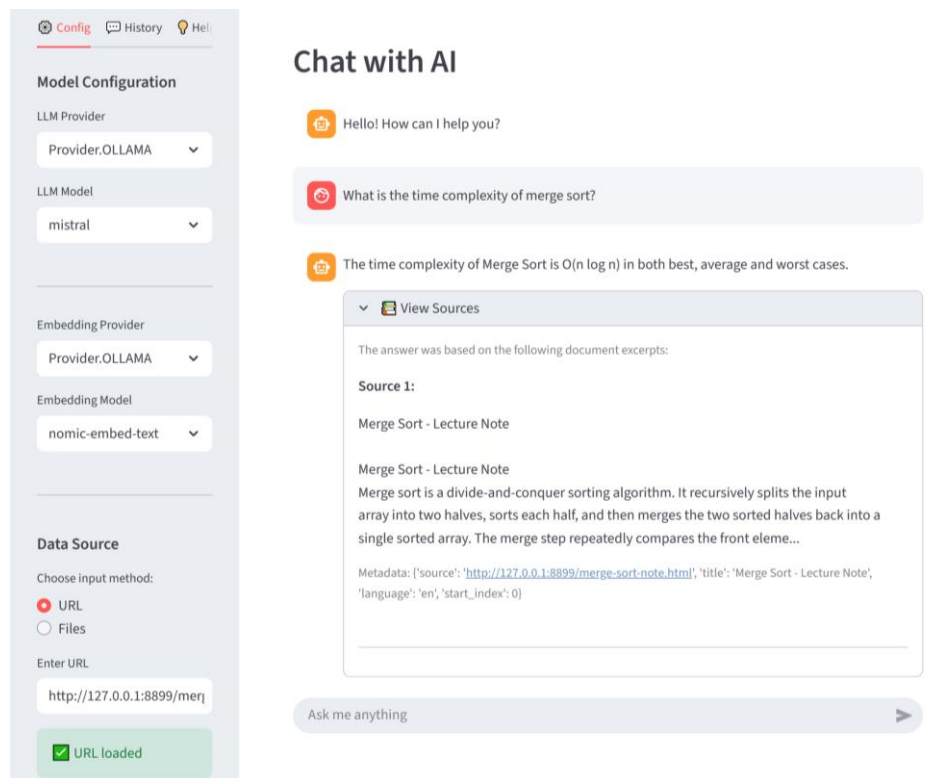


Figure 4: User interface, configuration options, chat area, and source citations

2.9. Provider Configuration and Error Handling

The system implements flexible provider configuration through environment variables and enumerated provider types. The configuration layer distinguishes between four provider categories: local (Ollama), and three cloud providers (OpenAI, Anthropic, Google) [11, 15]. This separation allows provider-specific credentials and settings to be managed independently from the core application logic, reducing configuration dependencies and simplifying deployment across different environments. It also enables the system to adapt its available functionality according to the providers that are currently configured and accessible.

Rather than requiring every provider to be configured, the system detects which providers have valid credentials (`is_provider_configured()`, `get_available_providers()`) and, for each, populates the interface with its supported LLM and embedding models. Institutions can thus start with a single provider and add others incrementally without reconfiguration.

3. Implementation Concepts and Extensibility

3.1. Provider and Model Configuration

Different deployments have access to different providers and models. To accommodate this diversity, the architecture uses a configuration mechanism – environment variables and/or configuration files – that determines which providers are available and how they are used. A typical configuration specifies, for each provider, API endpoints, credentials, a default decoding temperature and a maximum context length; for each model, a short description of its strengths and rough cost; and, for embeddings, the default provider and model. For factual, source-grounded question answering the prototype sets the decoding temperature to 0 for all cloud generators (OpenAI, Anthropic, Google), so that answers are deterministic and stay close to the retrieved passages instead of being creatively rephrased; local models use the runtime's default sampling. The configuration informs

both the factory layer and the UI controls, and changing provider or model invalidates the relevant caches.

3.2. Caching and Performance Trade-Offs

Vector index construction and embedding generation over large collections of educational materials constitute computationally intensive operations that can significantly increase processing time and resource consumption. To mitigate these costs, the proposed architecture employs a caching mechanism for document loading, embedding computation, and vector-index construction, thereby avoiding redundant processing during repeated queries and application sessions. This approach reduces response latency and improves computational efficiency, particularly when the underlying document collection remains unchanged. At the same time, caching introduces a consistency challenge because previously generated embeddings and indexes may become invalid when source materials, embedding models, or relevant configuration parameters are modified. To address this issue, the cache management mechanism monitors configuration changes and explicit user-triggered updates and invalidates the affected cached components when necessary. The corresponding documents are then reprocessed and the vector index is rebuilt using the current configuration. This ensures consistency between the indexed knowledge base and the active retrieval configuration while preserving the performance benefits of caching during stable operating conditions. Consequently, the architecture provides a controlled trade-off between computational efficiency and data consistency, which is particularly important for continuously evolving educational content.

3.3. Integration with Existing LLM Tooling

The architecture aligns with modern LLM toolkits: LangChain provides LLM, retriever and chain abstractions [11], FAISS efficient similarity search [12], and Streamlit rapid UI prototyping [15]. The contribution of this work lies in the integration of these components into a unified, education-oriented multi-provider RAG architecture. The proposed architecture introduces a provider-abstraction layer that decouples vendor-specific integrations from the core application, a retrieval pipeline based on generic document loaders and text-splitting mechanisms, and an application interface that coordinates provider configuration, document retrieval, response generation, and result presentation.

4. Educational Use Cases and Deployment Scenarios

The four representative use cases presented below are described using a consistent and structured framework that specifies the relevant actors, input data, system operations, and representative queries or outputs. This unified formulation facilitates systematic comparison of the use cases while maintaining a clear connection between the proposed architectural components and their practical application in educational scenarios.

UC1 – Lecture Material Exploration. The primary actor is the student, while the input consists of lecture slides, notes, and other instructional materials previously uploaded and indexed by the instructor. The system retrieves relevant content segments from the course-specific knowledge base and generates responses grounded in the available instructional materials, with retrieved segments aligned with the corresponding topics or instructional units. This approach enables students to obtain context-specific explanations while maintaining consistency with the materials used in the course. Representative queries include: “What is the main idea of the recursion lecture?” and “Explain depth-first versus breadth-first search using the examples provided in the lecture materials.”

UC2 - Programming documentation assistant. The primary actor is the student, and the input consists of programming language and library documentation. The system retrieves relevant documentation segments and combines concise conceptual explanations with corresponding code

examples, thereby preserving the relationship between theoretical concepts and their practical implementation. Representative queries include: “How do I use decorators in Python?” and “What is the difference between a list and a tuple?”

UC3 - Research paper and project support. The primary actor is the graduate student, while the input consists of selected research papers, surveys, or other scholarly sources. The system retrieves relevant passages according to the structure of scholarly documents, including sections such as introduction, methodology, and results, and provides context-grounded responses. Representative queries include: “What problem does this paper address?” and “How does the proposed method differ from the baseline?”

UC4 - Conversation history and reflection. The actors are students and instructors, while the input consists of previously generated dialogue stored as structured records containing roles, content, timestamps, provider and model information, and retrieved sources. The system supports conversation export for assignment submission, re-import for continuation across sessions or devices, and subsequent review by students or instructors. The resulting conversation record can serve as a documented learning artefact that supports metacognitive reflection, learning-process analysis, and reproducibility of AI-assisted educational activities.

Beyond these educational use cases, the proposed architecture supports several deployment scenarios that address different institutional requirements for privacy, cost, and computational resources. A fully local deployment keeps models, embeddings, and educational data within the institution’s infrastructure, providing a high level of data control and operational independence while requiring sufficient computational resources and potentially limiting access to the most capable models. A cloud-only deployment simplifies infrastructure management and provides access to a broad range of high-performance models, but introduces recurring service costs, external connectivity requirements, and additional data-governance considerations. A hybrid deployment combines both approaches by processing routine or privacy-sensitive requests locally while directing computationally demanding tasks to selected cloud providers through the provider-abstraction layer. This flexibility allows institutions to select a deployment strategy according to their privacy requirements, available infrastructure, budget, and expected workload, while also enabling gradual transition between deployment configurations as these conditions change.

5. Implementation Results and Cost Analysis

5.1. Cost-Per-Query Comparison Across Providers

To provide institutions with actionable information for deployment decisions, a comparative cost analysis of leading LLM providers for educational use cases was conducted. Based on published pricing from major providers as of August 2026, the paper calculated the cost per typical educational query using standardized assumptions.

The analysis assumes 500 input tokens (retrieval context plus student question) and 300 output tokens per query, with 1,000 queries per month as a baseline volume. Pricing uses published API rates as of August 2026 and does not include volume discounts. Table 1 presents cost per query, monthly costs, and annual costs for a medium-scale educational deployment (100 concurrent users, 1,000 queries/month each).

The cost analysis is based on the assumed workload of 500 input tokens and 300 output tokens per educational query, with 1,000 queries per month per user. Using the published API prices listed in Table 1, the estimated cost varies substantially across providers and models. For example, Gemini 2.5 Flash costs approximately \$0.0009 per query, corresponding to about \$0.90 per 1,000 queries (\$1,080 per year for 100 users), whereas Claude Sonnet 4 costs approximately \$0.0060 per query, or

about \$6 per 1,000 queries, \$7,200 per year. For a deployment involving 100 users, these differences result in substantially different annual operating costs.

The proposed hybrid strategy combines a lower-cost model for routine requests with a higher-capability model for more demanding tasks. Assuming that 70% of queries are processed by Gemini 2.5 Flash and the remaining 30% by Claude Sonnet 4, the resulting blended cost is approximately \$0.0024 per query. Under the stated workload assumptions, this corresponds to approximately \$2,900 per year for 100 users, representing a reduction of about 60% compared with using the premium model exclusively. These estimates illustrate the economic advantage of dynamic model selection while preserving access to higher-capability models when required.

Table 1
Cost comparison across LLM providers for educational deployment

Provider/Cost	Input (per 1M)	Output (per 1M)	Cost per Query	Monthly (1K queries)	Annual (100 users)
Ollama (Local)	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00*
Google Gemini 2.5 Flash	\$0.30	\$2.50	\$0.0009	\$0.90	\$1,080
OpenAI GPT-4o-mini	\$0.15	\$0.60	\$0.000255	\$0.26	\$306
Google Gemini 2.5 Pro	\$1.25	\$10.00	\$0.0036	\$3.63	\$4,350
Claude Haiku 3.5	\$0.80	\$4.00	\$0.0016	\$1.60	\$1,920
OpenAI GPT-4o	\$2.50	\$10.00	\$0.00425	\$4.25	\$5,100
Claude Sonnet 4	\$3.00	\$15.00	\$0.0060	\$6.00	\$7,200
Claude Opus 4.1	\$15.00	\$75.00	\$0.030	\$30.00	\$36,000
Proposed (hybrid)	–	–	\$0.0024	\$2.43	\$2,916

*Hardware: ~ \$1,500-3,000 one-time

From a broader deployment perspective, local inference eliminates recurring per-query API costs after the initial hardware investment, while low-cost cloud models provide an economical option for pilot deployments. Mid-range and premium models offer greater capabilities but incur higher operational costs, particularly at scale. Adaptive hybrid routing can mitigate these costs by assigning routine queries to cost-efficient models and escalating more demanding tasks to higher-capability models.

5.2. Break-Even Analysis

The authors performed a break-even analysis comparing local and cloud deployment strategies across institutions of different sizes, as presented in Table 2. The results show that for small institutions (fewer than 100 concurrent users) and deployment periods of up to two years, budget-oriented cloud solutions are generally more cost-effective.

Table 2
Break-even analysis for local vs. cloud deployment

Institution Size	Monthly Queries	Optimal Strategy	Annual Cost	Break-Even
Small (10 users)	10K	Local	\$0	~ 60 years
Medium (100)	100K	Hybrid	~ \$2,900	~ 60 years
Large (1,000)	1M	Hybrid	~ \$29,000	~ 1 year
Very Large (10K)	10M	Local	\$0	~ 2 months

For larger user populations or longer deployment periods, local infrastructure becomes increasingly competitive due to the absence of recurring per-query service costs.

5.3. Prototype Implementation

The working prototype of the proposed architecture was implemented using the following technology stack (Figure 5):

1. backend (Python, LangChain);
2. frontend (Streamlit);
3. vector database (FAISS);
4. providers (Ollama, OpenAI, Anthropic, Google);
5. deployment (Docker).

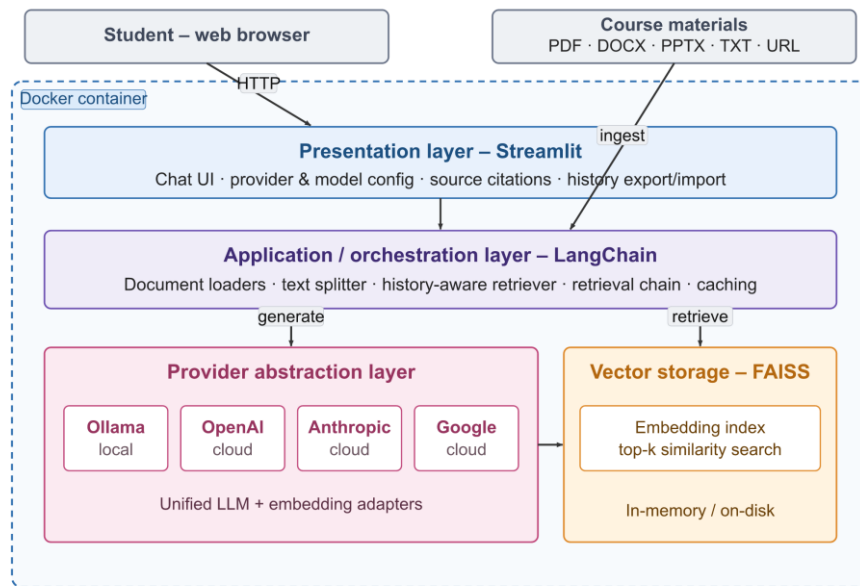


Figure 5: Functional diagram of the prototype technology stack (Docker-packaged)

The prototype implements the principal capabilities of the proposed architecture, including runtime switching between LLM providers without application redeployment, history-aware query reformulation, retrieval-grounded response generation with source citations, configurable retrieval parameters, and multi-document ingestion and indexing. The experimental evaluation was conducted on a commodity desktop workstation equipped with an AMD Ryzen 5 3600 processor (6 cores, 12 threads), 32 GB RAM, and an NVIDIA GeForce GTX 1660 SUPER GPU with 6 GB VRAM, running Windows 10. Local inference was performed using Ollama 0.13, with Python 3.11, LangChain, FAISS, and Streamlit comprising the application software stack. The embedding model was nomic-embed-text, while local generation was evaluated using a small approximately 1B-parameter Gemma 3 model and Mistral 7B. The evaluation corpus consisted of a digital design (VHDL) course reader and associated lecture notes. Documents were processed using a chunk size of 1000 characters, an overlap of 200 characters, and a retrieval depth of five segments. Indexing throughput was measured under cold-cache conditions, whereas end-to-end response latency, defined as the interval from query submission to answer generation, was averaged over repeated queries using a pre-built index. The complete source code and configuration files are openly available, supporting reproducibility of the reported experimental results.

Under the evaluated configuration, nomic-embed-text processed approximately 11 one-thousand-character chunks per second, corresponding to roughly 250 pages per minute assuming 2,000

characters per page. Local query latency was primarily determined by the selected generation model, ranging from approximately 1.5–4 s (mean ≈ 2 s) for the small approximately 1B-parameter model to 12–20 s (mean ≈ 15 s) for Mistral 7B. The higher latency of Mistral 7B reflects the computational limitations of the evaluated GPU configuration. Cloud-based providers typically achieved response times of approximately 2–8 s, depending on the selected model and service load. Memory consumption was primarily determined by the loaded model weights, requiring approximately 0.8 GB for the 1B model and 4.4 GB for Mistral 7B, with an additional approximately 0.3 GB consumed by the Python and FAISS processes. These results indicate that local model size is a primary determinant of inference latency and memory requirements and should therefore be considered alongside the provider cost analysis when selecting a deployment configuration.

6. Discussion and Limitations

6.1. Advantages of a Multi-Provider, RAG-Based Design

The conceptual architecture described above offers several advantages over single-provider, non-retrieval-based systems (Table 3).

Table 3

Comparison of the proposed architecture with typical alternatives

Capability	Single-provider RAG	Closed cloud chatbot	Proposed multi-provider RAG
Runtime provider/model switching	No	No	Yes
On-premise / local inference	Rare	No	Yes (Ollama)
Automatic capability fallback	n/a	n/a	Yes
Grounding + source citations	Yes	Limited	Yes
Vendor lock-in	High	High	Low
Cost control / hybrid routing	Limited	No	Yes
Data sovereignty / privacy	Partial	Low	High (local option)

First, it reduces vendor lock-in by modelling providers as interchangeable components, so an institution can start with whichever provider is most accessible and later switch or augment its setup without rewriting the core logic; and it enhances transparency and robustness by grounding answers in specific materials and showing citations, which aligns with academic norms. Second, it supports cost and privacy optimization: local deployments avoid recurring API charges and reduce data exposure, while hybrid routing applies more expensive cloud models only when they add clear pedagogical value (our analysis shows 40-60% savings for medium-to-large institutions); and the architecture is extensible to future providers, models and modalities without disrupting its structure.

6.2. Limitations and Open Challenges

Controlled classroom studies are still required to measure learning outcomes and instructor workload. Several technical limitations also remain. Retrieval quality is bounded by the embedding model and the chosen (c, o, k): poorly curated or purely graphical materials lower recall, and the current text-based ingestion does not interpret diagrams or figures. Hallucination is reduced but not eliminated by grounding – the generator may still misread retrieved passages – so answer-faithfulness and citation checks are needed. Provider switching is not cost-free: different models change latency and answer style, and changing the embedding provider requires full re-indexing because embedding spaces are not interchangeable. Finally, behavior under realistic peak workloads (many concurrent users, large corpora) has not yet been stress-tested, and local hosting may remain difficult for under-resourced institutions.

6.3. Future Work

Future research can explore several directions: controlled classroom studies comparing single-provider and multi-provider RAG chatbots; adaptive provider selection policies; integration with learning analytics; multimodal extensions incorporating images and code execution; teacher-facing analytics and authoring tools; cost-aware model selection algorithms; and regional model optimization for specific languages and curricula [1].

7. Conclusion

This paper presents a conceptual architecture for a multi-provider, RAG-based educational chatbot tailored to Computer Science and STEM education. By explicitly separating provider concerns from retrieval and interaction layers, the design enables institutions to combine local and cloud-based LLMs, adjust retrieval parameters to pedagogical needs, and ground responses in their own course materials. The architecture addresses pressing challenges of cost, privacy, and robustness, especially for institutions under strict data protection regimes or limited connectivity. The conducted empirical cost analysis demonstrates institutions of varying sizes can find deployment strategies balancing innovation with fiscal responsibility. The architecture remains flexible to leverage ongoing advances in LLM capabilities and open-source tooling. Multi-provider RAG architectures represent a promising direction for institutionally sustainable educational AI, balancing innovation with practical constraints. Realizing this potential requires collaboration between educators, system designers, researchers, and LLM developers to evaluate and refine such systems in authentic learning environments. This transition is not merely technical improvement but necessary for equitable, private, and economically viable educational technology for institutions worldwide. The prototype integrates four interchangeable providers and more than eight models behind a single abstraction, indexes course materials at about 250 pages per minute, answers in about 2 s with a small local model (about 15 s with a 7B model) and about 2-8 s with cloud providers, and needs roughly 1-5 GB of memory depending on the local model, on commodity desktop hardware. The cost analysis spans from \$0 recurring (local) to about \$36,000 per year (premium cloud at 100-user scale); the hybrid routing of the proposed architecture cuts recurring cost by roughly 60% relative to premium-only deployment while preserving grounding and data sovereignty.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT in order to: grammar and spelling check. The use of AI tools was limited to supporting the writing process and did not influence the conclusions drawn from the evidence. After using this tool/service, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

References

- [1] I. Solohubov, A. Moroz, M. Y. Tiahunova, H. H. Kyrychek, S. Skrupsky. Accelerating software development with ai: exploring the impact of chatgpt and github copilot. In CEUR Workshop Proceedings, 2023, 3679, pp. 76–86, doi: 10.55056/ceur-ws.org/Vol-3679/paper17.pdf.
- [2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023, doi: 10.48550/arXiv.2303.08774.
- [3] W. C. Choi, I. C. Choi, C. I. Chang, L. C. Lam. Comparison of Claude (Sonnet and Opus) and ChatGPT (GPT-4, GPT-4o, GPT-o1) in analyzing educational image-based questions from block-based programming assessments. In 2025 14th International Conference on Information and Education Technology (ICIET), 2025, pp. 55-60, doi: 10.1109/ICIET66371.2025.11046263.

- [4] G. Team, R. Anil, S. Borgeaud, Y. Wu, J. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. Dai, A. Hauth et al. Gemini: A family of highly capable multimodal models. arXiv preprint arXiv:2312.11805, 2024, doi: 10.48550/arXiv.2312.11805.
- [5] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, D. Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in neural information processing systems*, 2020. (33), pp. 9459–9474. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.
- [6] G. Izacard, M. Caron, L. Hosseini, S. Riedel, P. Bojanowski, A. Joulin, E. Grave. Unsupervised dense information retrieval with contrastive learning. arXiv preprint arXiv:2112.09118, 2021, doi: 10.48550/arXiv.2112.09118.
- [7] S. Z. Salas-Pilco, Y. Yang. Artificial intelligence applications in latin american higher education: a systematic review. *International Journal of Educational Technology in Higher Education*, 2022. 19(21), doi: 10.1186/s41239-022-00326-w.
- [8] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, D. Zhou. Self-consistency improves chain of thought reasoning in language models. arXiv preprint arXiv:2203.11171, 2022, doi: 10.48550/arXiv.2203.11171.
- [9] M. Emara, N. M. Hutchins, S. Grover, C. Snyder, G. Biswas. Examining student regulation of collaborative, computational, problem-solving processes in open-ended learning environments. *Journal of Learning Analytics*, 2021. 8(1), pp. 49–74, doi: 10.18608/jla.2021.7230.
- [10] S. Polak, G. Schiavo, M. Zancanaro. Teachers’ perspective on artificial intelligence education: An initial investigation CHI conference on human factors in computing systems extended abstracts, 2022. 431, pp. 1–7, doi: 10.1145/3491101.3519866.
- [11] D. Grigorov. Langchain and python: Advanced components. *Intermediate Python and Large Language Models*, 2025, pp. 59–137, doi: 10.1007/979-8-8688-1475-4_2.
- [12] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P. E. Mazaré, M. Lomeli, L. Hosseini, H. Jégou. The faiss library *IEEE Transactions on Big Data*, 2025. 12(2), pp. 346–361, doi: 10.1109/TBDATA.2025.3618474.
- [13] R. Saraf, G. Kolhe, S. Badjate, R. Gadekar, P. Joshi. Document query processing with mistral 7b and streamlit. 2024 8th International Conference on Computing, Communication, Control and Automation (ICCUBEA). IEEE, 2024, pp. 1–6, doi: 10.1109/ICCUBEA61740.2024.10774917.
- [14] R. Gupta. Comparative analysis of deepseek r1, chatgpt, gemini, alibaba, and llama: Performance, reasoning capabilities, and political bias. *Authorea Preprints*, 2025, doi: 10.22541/au.173921625.50315230/v1.
- [15] X. Hou, J. Han, Y. Zhao, H. Wang. Unveiling the landscape of llm deployment in the wild: An empirical study. arXiv preprint arXiv:2505.02502, 2025, doi: 10.48550/arXiv.2505.02502.
- [16] F. S. Marcondes, A. Gala, R. Magalhães, F. Perez de Britto, D. Duraes, P. Novais. Using ollama Natural Language Analytics with Generative Large-Language Models: A Practical Approach with Ollama and Open-Source LLMs, 2025, pp. 23–35, doi: 10.1007/978-3-031-76631-2_3.

- [17] R. Purushothaman. A comprehensive analysis of consumer behavior of OpenAi. Available at SSRN 5146257, 2025, doi: 10.2139/ssrn.5146257.
- [18] E., Mogaji, V. Jain, (2024). How generative AI is (will) change consumer behaviour: Postulating the potential impact and implications for research, practice, and policy. *Journal of Consumer Behaviour*, 23(5), pp. 2379-2389, doi: 10.1002/cb.2345.
- [19] A. Valiulla. Comparative analysis of next-generation large language models (llms): Architectural advances. Reasoning Capabilities, and Multimodal Integration (2024-2025), 2025, doi: 10.2139/ssrn.5329049.
- [20] M. Li, B. Krishnamachari. Evaluating LLMs Accuracy in Solving Coding Problems of Different Complexity Levels: An Empirical Analysis, SoutheastCon 2026, Huntsville, AL, USA, 2026, pp. 1-6, doi: 10.1109/SoutheastCon63549.2026.11476431.