ECHELON: YOUR AI ACADEMIC ADVISOR

by

Harrison Leath

Submitted in partial fulfillment of the

requirements for Departmental Honors in

the Department of Computer Science

Texas Christian University

Fort Worth, Texas

May 5, 2025

ECHELON: YOUR AI ACADEMIC ADVISOR

Project Approved:

Supervising Professor: Bingyang Wei, Ph.D.

Department of Computer Science

Ze-Li Dou, Ph.D.

Department of Mathematics

Bo Mei, Ph.D.

Department of Computer Science

ABSTRACT

Academic advising is a critical yet time-intensive process that often leaves students with limited access to timely, personalized guidance. This thesis introduces Echelon, an AI-powered academic advisor designed to augment the advising experience by leveraging semantic embeddings, large language models (LLMs), and retrieval-augmented generation (RAG). Echelon ingests institutional documents and student transcripts to deliver accurate, context-aware responses through a conversational web interface. Echelon is publicly accessible via echelongpt.com and can also be internally deployed on TCU's servers, offering enhanced privacy and security controls tailored for institutional use. This paper details Echelon's underlying technologies, system architecture, and deployment and outlines future enhancements such as proactive advising and domain expansion beyond academia. The result is a scalable, extensible framework for transforming how students interact with institutional knowledge.

Echelon: Your AI Academic Advisor

Harrison Leath

May 5, 2025

Contents

1 Introduction **5**

2 Background **6**

2.1 Semantic Embeddings 6

2.2 Large Language Models 9

2.3 Retrieval-Augmented Generation 11

3 System Architecture **11**

3.1 Frontend Component 12

3.2 Backend Component 14

3.2.1 The Rust Language 15

3.2.2 The Four Modules 16

3.3 Compute Component 19

3.4 Database Component 19

4 Personalization **20**

5 Echelon vs. General-Purpose LLMs **23**

6 Deployment **24**

7 Limitations and Future Work **25**

8 Conclusion **26**

9 Acknowledgements **27**

10 References **29**

A Prompt Used for Q&A **31**

B Prompt Used for Generation of Chat Titles **31**

C Prompt Used for Transcript Parsing **32**

D Access to Source Code **32**

List of Figures

1 Word embeddings visualized in a 2D vector space. Words with similar meaning (e.g., *awesome* and *fantastic*) are close together. 7

2 System architecture of Echelon. The platform is organized into four major components: frontend (pink), backend (purple), compute (blue), and database (yellow). Arrows indicate the flow of data and dependencies between modules. 12

3 An example of Echelon's frontend interface during a graduation audit query. 14

4 An example of Echelon's frontend interface describing a student's academic profile. 21

5 The output of Echelon's transcript parsing pipeline in Markdown format. This is an example academic profile used to provide context on a student in the RAG pipeline. 22

1 Introduction

Every semester, thousands of undergraduate students at TCU navigate the complex maze of academic requirements and limited access to expert guidance. Echelon reimagines this system through the lens of artificial intelligence.

Academic advising is a three-party system of students, professors, and advisors. Of the three, students begin their career with the least amount of useful information about their degree plan and must rely heavily on the guidance of professors and advisors to map out their next four years. Professors possess deep domain expertise since they build the coursework, but their time is scarce. Meeting with dozens or even hundreds of students each semester pulls professors away from conducting research, writing grants, and lecturing. In contrast, advisors are experts on institutional requirements but often lack firsthand experience with the academic challenge of individual courses. Like professors, their time is limited, and they cannot be expected to recall every detail of a student's academic history. While tools like Stellic attempt to modernize the process of advising, they often introduce a steep learning curve for students and may even contain incomplete or inaccurate information since TCU must migrate their documentation into a form more acceptable to these platforms (Stellic Inc., 2025).

The complexity found in our academic advising process inspired the development of Echelon, a personalized AI academic advisor for every student. Common advising queries – such as "Am I on track to graduate next semester?" or "Which courses fulfill my remaining major requirements?" – are efficiently handled by Echelon, making interactions with professors and advisors significantly more productive. The platform uses large language models alongside crucial documents, like TCU's course catalog and a student's transcript, to generate accurate and personalized responses. Echelon enhances student decision-making by providing timely, personalized guidance in their advising journey, so meetings with professors and advisors can be significantly more efficient, saving time for all involved parties across all departments

on campus.

Before discussing the implementation details of Echelon, it is crucial to understand the underlying technologies that enable Echelon to function. Just as a native English speaker would find it challenging to read Japanese, computers have a difficult time understanding human language, as they have only been exposed to the language of math. Researchers developed semantic embeddings to address this gap; these semantic embeddings act as a translator for computers to derive numbers from words, sentences, or phrases in a language like English (Mikolov, Chen, Corrado, & Dean, 2013). We can understand these embeddings as giving computers the ability to 'read' our languages, but they do not enable computers to 'write' in our languages, so large language models (LLMs) were developed as a method of allowing computers to generate human language. With these advancements, computers gained the ability to read and write in human language, and, if we give these programs access to useful information, computers can even deliver useful insights to their users.

Echelon takes these core principles and applies them to the domain of academic advising. The following background section will explore landmark literature in each of these relevant fields to better illuminate the technologies the Echelon system is built on top of.

2 Background

2.1 Semantic Embeddings

The development of semantic embeddings can be attributed to the field of natural language processing (NLP). These embeddings provide an interface for computers to extract the meaning of words into a high-dimensional vector representation. Semantic embeddings function similarly to coordinates on a map, placing words and sentences close together if they share meaning. Just as we measure physical distances on a map, semantic similarity can be measured through vector distances, such as cosine similarity or Euclidean distance.

As shown in Figure 1, four words are plotted in a two-dimensional vector space to

illustrate how semantic similarity can be captured through embeddings. Words like awesome and fantastic are positioned close together due to their similar meanings, while horrible and bad form a separate cluster reflecting negative sentiment. If a new word, such as terrible, were introduced, it would be embedded and placed near horrible and bad, rather than near awesome and fantastic. This spatial arrangement highlights the underlying structure captured by semantic embeddings and serves as a simplified example of how meaning is preserved in high-dimensional space.

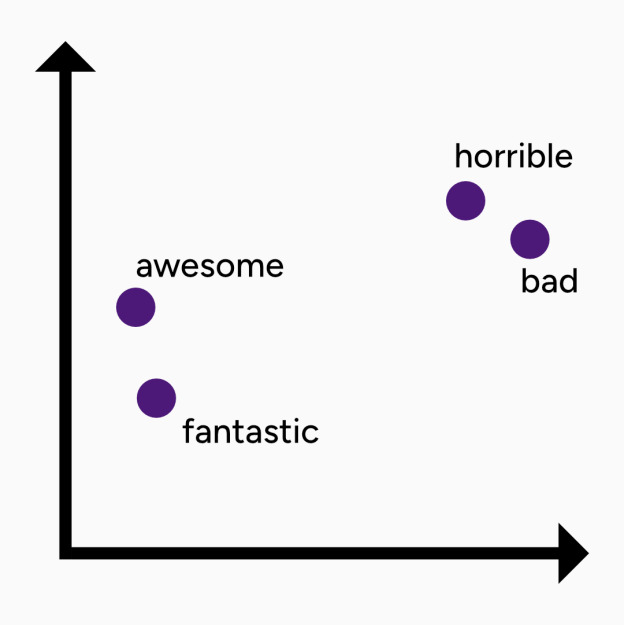

Figure 1: Word embeddings visualized in a 2D vector space. Words with similar meaning (e.g., *awesome* and *fantastic*) are close together.

The technique of generating semantic embeddings differs drastically from rudimentary techniques of information retrieval, like plain text matching, which can be dramatically influenced by human errors like misspelling a word. An additional advantage of creating semantic embeddings from natural language is the ability to encode more than a single word at a time. Modern embedding models can take entire sentences or passages and convert them into a vector such that computer systems can compare the meaning of much larger sets of data at a time, lowering the computational requirements of the search. To understand the current progress of semantic embeddings, it is useful to learn the research background

of the topic.

A significant advancement in the field came from Google's Word2Vec paper, which introduced two simple yet powerful neural network models: Continuous Bag-of-Words and Continuous Skip-gram (Mikolov et al., 2013). These architectures drastically improved the quality of word embeddings while keeping computational demands low, making them both accurate and scalable. For the first time, a neural network could consistently capture nuanced syntactic and semantic relationships between words, demonstrated by analogical reasoning tasks like "king – man + woman = queen". At the time, Word2Vec set a new standard in word similarity and relational benchmarks.

With the introduction of BERT, semantic embeddings reached a new level of contextual understanding by using a transformer-based architecture to capture context from both directions, left-to-right and right-to-left, simultaneously (Devlin, Chang, Lee, & Toutanova, 2019). This bidirectional technique greatly enhanced the ability of embeddings to model nuanced relationships between words in a sentence. However, traditional BERT was computationally expensive. To address this limitation, Sentence-BERT (SBERT) was introduced, using a Siamese network structure to efficiently derive semantically meaningful sentence embeddings (Reimers & Gurevych, 2019). SBERT drastically reduced computational overhead while preserving accuracy, enabling fast semantic similarity searches with high precision.

In recent years, continued improvements in training methodology and dataset quality have produced even more refined embedding models, especially for retrieval-augmented tasks. One such model, "BGE-large-en-v1.5", uses contrastive learning fine-tuned for retrieval (Xiao, Liu, Zhang, & Muennighoff, 2023). The model learned to pull semantically similar queries and documents closer together in vector space, making it ideal for tasks like question answering and document matching. Unlike BERT or SBERT, which were originally designed for general-purpose NLP tasks, BGE was trained with a specific instruction-tuned objective for sentence and passage retrieval, so it was an excellent candidate for a platform like Echelon. While semantic embeddings enable computers to represent and compare pieces

of text, LLMs allow computers to generate new language and 'reason' about information.

2.2 Large Language Models

Imagine walking into a library that contains every book ever written, every conversation ever spoken, and every newspaper ever printed. In the center of the library sits a librarian who has read everything the library contains without having ever left its walls. The librarian has memorized the contents of every book, but he has never lived the experiences, felt the emotions, or seen the places described inside the text. When you ask him a simple question like "Why is the sky blue?", the librarian could guess the answer and tell you (rather convincingly so), but he has no firsthand experience seeing a blue sky. He knows how that question has been answered in the books he has read, but he is not reasoning through the effects of Rayleigh scattering, for example.

In a similar way, LLMs are very well-read computers with impressive pattern-matching capabilities. Whenever a question is asked to an LLM, the model calculates the highest probability next word. To make the models feel more natural, researchers will often instill an element of randomness such that the top n possible next words will be chosen, preventing the model from answering the same question identically each time. These models generate what appears to be a thoughtful answer – without using any thought at all (Bender, Gebru, McMillan-Major, & Shmitchell, 2021).

The foundation for LLMs is the transformer architecture, introduced in the landmark paper "Attention is All You Need" (Vaswani et al., 2023). Unlike previous models that process language sequentially, transformers use a mechanism called self-attention, allowing them to consider the entire context of a sentence or paragraph at once. This innovation drastically improved training efficiency and model accuracy. To this day, transformers are the backbone of nearly every LLM.

Following the transformer revolution, researchers from Google released the aforementioned BERT paper with its unique bidirectional approach that was also incredibly beneficial

for LLMs. The shift from understanding text to generating text came with OpenAI's GPT series, culminating in the release of GPT-3 (Brown et al., 2020). GTP-3 showed that, at sufficient scale, LLMs could perform a variety of tasks with minimal instruction. For example, just giving GPT-3 a few examples in a prompt dramatically improved model output. This idea, known as few-shot learning, made LLMs practical for real-world applications without requiring fine-tuning for each individual task.

In recent years, researchers have extended the power of LLMs beyond text. Just as LLMs process sequences of words, they can also be adapted to process sequences of image tokens – enabling them to 'see' and 'understand' visual inputs alongside text. This has led to the rise of vision-language models, also called multimodal LLMs.

By incorporating visual encoders (such as convolutional neural networks or vision transformers), these models convert images into embeddings that can be merged with textual embeddings and processed through a shared transformer backbone. More concretely, this means a model can answer questions about an image, generate captions, or reason about visual scenes – all in one model.

While LLMs are impressively capable on their own, they are inherently limited to the information they were trained on. Once deployed, models cannot dynamically update their knowledge or access real-time information; therefore, any events, regulations, or documents occurring after their training cutoff remain inaccessible. In the librarian analogy from earlier, while the librarian demonstrates impressive knowledge, the librarian's answers may occasionally lack accuracy without access to source material; this underscores the importance of grounding techniques. In many real-world settings this limitation poses a serious problem. A popular grounding technique to alleviate this limitation is called retrieval-augmented generation or RAG.

2.3 Retrieval-Augmented Generation

Instead of exclusively relying on the model's internal parameters for knowledge, RAG introduces an external retrieval mechanism that allows the model to look up relevant information, such as university course descriptions and student records. For instance, if a student queries "What prerequisites do I need for COSC 40203?" a traditional LLM might guess incorrectly based on outdated training data. In contrast, RAG systems retrieve current, accurate details directly from institutional documents, guaranteeing a higher factual accuracy for each response.

RAG was formalized in a 2020 paper, which paired a dense passage retriever (think applied semantic embeddings) with a generative transformer model (think LLM) (Lewis et al., 2021). Since then, a wide range of RAG-inspired architectures have emerged, powering everything from customer service agents to enterprise search tools. For example, ChatGPT and Claude are considered RAG applications. The decision to use a RAG-based approach allows Echelon to dynamically pull relevant course data, catalog entries, or policy documents, providing personalized answers grounded in official TCU documentation.

3 System Architecture

Echelon's system architecture comprises four core components:

- **Frontend Component:** The user-facing interface built with Svelte.
- **Backend Component:** The primary server logic orchestrated by Rust-based modules.
- **Compute Component:** Dedicated inference servers powered by Ollama.
- **Storage Component:** Specialized databases (Qdrant, PostgreSQL, MinIO) for persistent data storage.

Figure 2 illustrates how these components interact through modular interfaces, forming a loosely coupled but tightly coordinated system. Arrows in the diagram denote the flow of data and control between modules, such as the Vectorization Module for semantic embeddings, the Storage Module for data persistence, and the LLM Module for language generation. By separating responsibilities across these components, Echelon ensures high cohesion within modules and low coupling between them – an essential property for scalable system design. This architecture also allows for independent updates, easier debugging, and horizontal scaling, particularly important for supporting real-time inference and large-scale document retrieval.

The following subsections explore each component in greater detail, examining how their internal modules work together to support Echelon's academic advising capabilities.

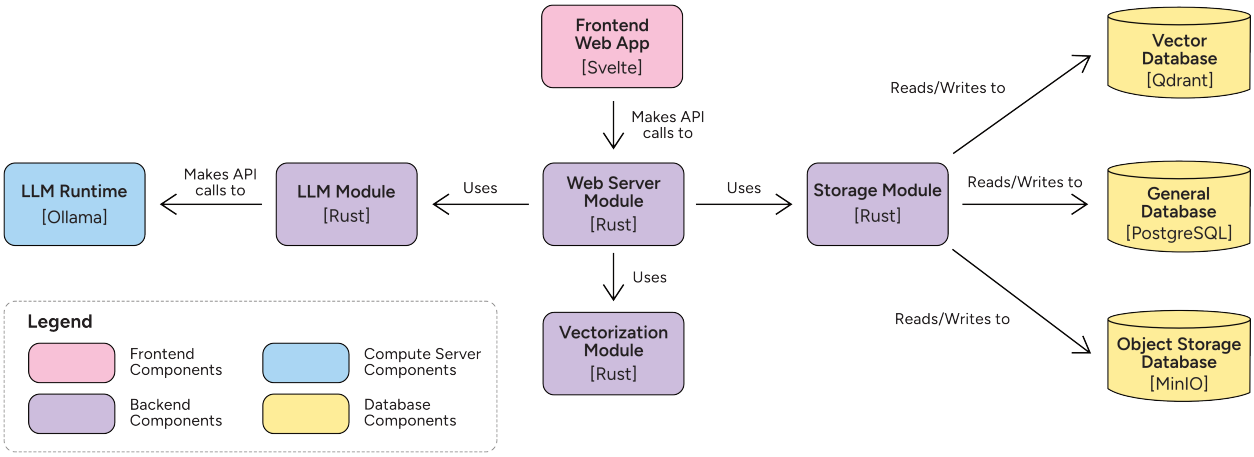

Figure 2: System architecture of Echelon. The platform is organized into four major components: frontend (pink), backend (purple), compute (blue), and database (yellow). Arrows indicate the flow of data and dependencies between modules.

3.1 Frontend Component

Conceptually, the frontend is everything the user sees and interacts with; it includes the buttons, forms, and visual elements rendered in the browser. If Echelon were a restaurant, the frontend would be the dining area where customers are served their food. Echelon's frontend is built using Svelte, a modern JavaScript framework, along with its full-stack

companion SvelteKit, which provides page routing and server-side rendering out of the box (Rich Harris et al., 2025).

Svelte simplifies the process of building fast, interactive interfaces by compiling components into highly optimized JavaScript at build time. This results in minimal runtime overhead and excellent performance. SvelteKit complements this by enabling seamless client-side navigation. For example, Echelon can transition between different chat conversations without triggering a full page reload.

To improve developer experience and long-term maintainability, the frontend is written in TypeScript, a superset of JavaScript that adds static typing. This makes it easier to catch errors early and maintain code consistency as the codebase grows. For styling, the project uses TailwindCSS, a utility-first framework that accelerates development by offering highly reusable, composable CSS classes. To further streamline development, Echelon integrates shadcn/ui, a customizable component library built on top of Tailwind and Radix UI. Shadcn provides accessible, pre-styled UI components – such as buttons, cards, and dropdowns – that lend the application a polished, professional look without the overhead of building components from scratch. Together, these tools enabled rapid iteration without compromising on design consistency or code quality.

The frontend supports two primary workflows: user authentication and question submission (Q&A). In the authentication flow, users register using their TCU email and are prompted to enter their first name, last name, and university (selected from a predefined list). The login flow mirrors this structure and requires less data on reentry. In the Q&A flow, once a user submits a question, the frontend packages the input into a request sent to the backend component. It immediately updates the UI with a pending state, improving perceived responsiveness. As the backend streams Echelon's response, the Svelte-powered frontend updates the UI in real time, providing a fluid, conversational user experience powered by the framework's reactivity model.

Figure 3 provides a snapshot of Echelon's frontend interface during a graduation audit

query. The interface is designed for clarity and responsiveness, enabling students to submit questions and receive personalized responses in real time. In this example, the user's query is displayed alongside Echelon's grounded answer, showcasing how the system integrates retrieved context with large language model generation to support academic advising.

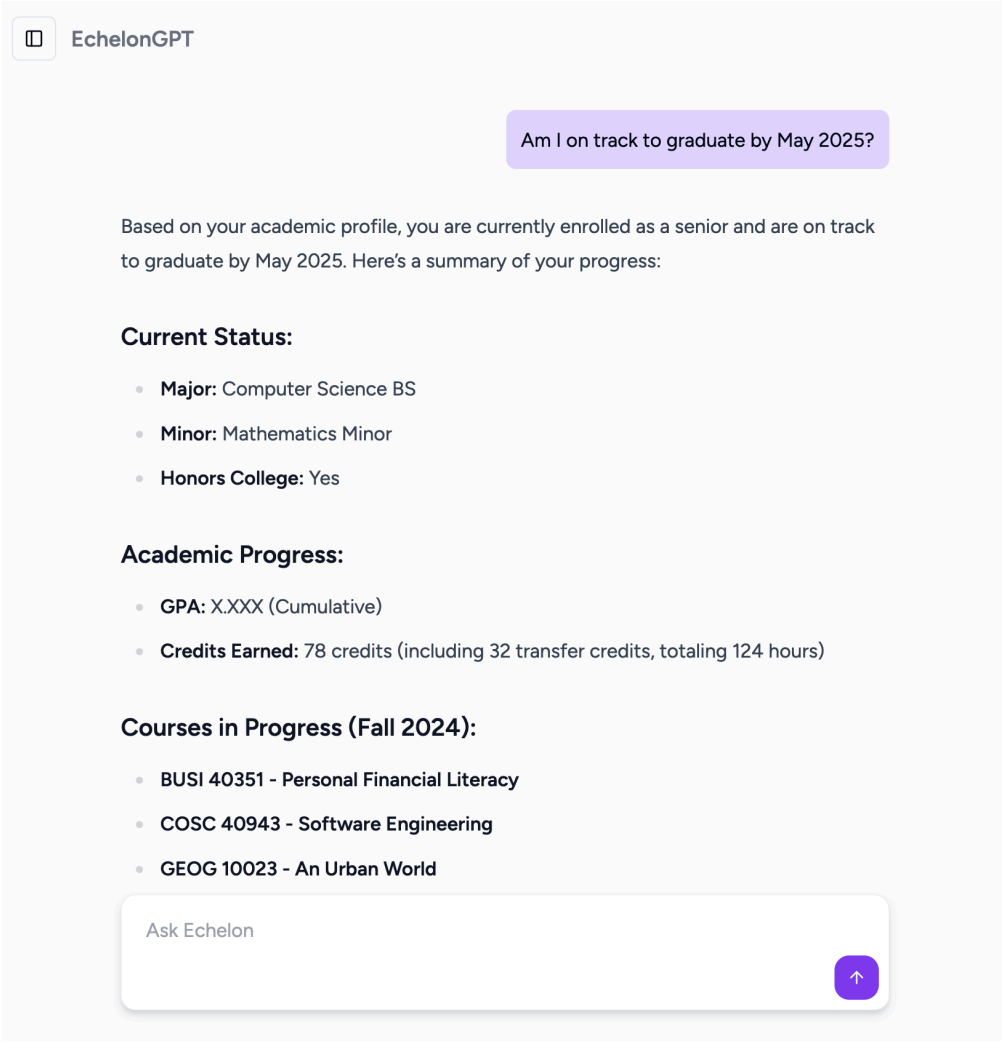

Figure 3: An example of Echelon's frontend interface during a graduation audit query.

3.2 Backend Component

In the restaurant analogy from earlier, if the frontend is considered the dining area, the backend would be the kitchen. As a customer, you would never set foot in the kitchen, but it is the beating heart of the restaurant. If the kitchen were to stop for even a moment,

customers would be left waiting too long for their meals.

3.2.1 The Rust Language

The backend is written in Rust, a systems programming language supported by a variety of modern frameworks and libraries. Before diving into the implementation of the backend, it's worth pausing to explain the decision to build an LLM-powered application using Rust.

For the uninitiated, Python is the de facto language for machine learning development – thanks in large part to its rich ecosystem of libraries like PyTorch, Pandas, and LangChain. These tools drastically reduce development time by providing pre-built solutions to common problems. In many cases, the functionality you need is already available and just an import statement away.

This convenience, however, comes with trade-offs. Python's performance and memory efficiency can suffer, particularly under high-concurrency or real-time workloads. The Global Interpreter Lock (GIL), a core feature of Python's execution model, prevents true multithreading, limiting the ability to fully utilize multicore systems. This becomes a bottleneck in applications like Echelon, where multiple backend operations must occur in parallel for every user request, like embedding a question, searching a vector database, retrieving context, or generating a response from an LLM.

Rust, by contrast, is a compiled, statically typed language designed from the ground up for performance and memory safety. Its ownership model and strict type system ensure thread safety at compile time. This eliminates entire classes of bugs common in production environments. For Echelon, this translated into real-world benefits: lower latency, predictable memory usage, and greater reliability – all achieved without a garbage collector or runtime interpreter.

Another advantage of Rust is its first-class support for asynchronous programming. This is crucial when a server must efficiently handle requests from hundreds or thousands of users. Echelon's backend uses the Tokio async runtime and the Axum web framework

to process many concurrent operations with minimal overhead. This async foundation is essential in a RAG system, where inference involves multiple chained steps: vectorization, semantic search, context retrieval, and LLM generation.

It is crucial to note that the Echelon system does not attempt to train or fine-tune LLMs itself. Rust is exclusively used to orchestrate and serve components such as semantic embedding models and remote LLM inference APIs. In this hybrid approach, Rust handles the mission-critical components while delegating model inference to modules written in other languages via HTTP calls. This balances the vast ecosystem of languages like Python with the performance and safety guarantees of Rust's backend design.

3.2.2 The Four Modules

Echelon's backend is composed of four specialized modules designed for clear separation of concerns, improved maintainability, and scalability:

- **Web Server Module:** The central orchestrator, exposing functionality via APIs and coordinating between other modules.
- **Vectorization Module:** Semantic embedding of documents and user questions.
- **Storage Module:** Manages interactions with multiple databases hosted through the Storage component.
- **LLM Module:** Handles interaction with external inference services such as the Compute component.

At the heart of Echelon's backend lies the Vectorization Module. This module performs two core functions: ingesting documents into the system and converting user questions into semantic vectors for retrieval. When ingesting a document, such as the TCU course catalog, the module accepts input in Markdown format and splits the text into manageable chunks of approximately 1,000 words each, with a 10% overlap (i.e., 100 words repeated between

adjacent chunks). This overlap improves contextual continuity and reduces the likelihood of splitting important phrases or sentences between chunks.

Each text chunk is then passed through a semantic embedding model, such as BGE-large-en-v1.5, to generate a high-dimensional vector that captures the underlying meaning of the text. These embeddings are then handed off to the Storage Module for persistence. The same process is applied to user questions at inference time, allowing for real-time search comparison against previously embedded documents. In this way, the Vectorization module serves as both an ingestion pipeline for adding new documents and a runtime engine for transforming user input into searchable vector representations.

The Storage Module directly interfaces with the Storage Component, managing interactions with three specialized databases: PostgreSQL, Qdrant, and MinIO, optimizing data handling for relational, vector, and unstructured data, respectively.

First, the Storage Module connects to a Qdrant vector database which stores all semantic embeddings generated by the Vectorization module. Optimized for similarity search, Qdrant enables Echelon to perform cosine similarity queries across its knowledge base, retrieving the most semantically relevant chunks in responses to a user's question. This process forms the core of Echelon's RAG pipeline.

Additionally, the Storage module also interfaces with a PostgreSQL database responsible for traditional application data, such as user profiles, authentication credentials, and chat histories. This relational layer is essential for maintaining user state, supporting personalization, and enabling analytics across sessions.

Lastly, the module integrates with a MinIO object storage system: an S3-compatible solution used to store large or structured files. For example, whenever a student transcript is uploaded to Echelon, it is saved in this database.

The LLM Module interacts directly with the Compute Component, which runs inference using Ollama, allowing real-time text generation and multimodal interactions. For text-based Q&A, Echelon uses Mistral Small 24B, a high-performance instruction-tuned LLM

optimized for chat-style interactions (Mistral AI, 2025). The full prompt template used to generate responses for academic questions is included in Appendix A. For multimodal and vision-language tasks, Echelon relies on Qwen 2.5 VL 72B, which enables the system to parse transcripts, extract structured data, and reason across visual formats like PDFs (Bai et al., 2025). This decoupled design allows the platform to remain flexible and modular such that more advanced or performant models can be swapped in as they are released without altering the surrounding system architecture.

When a user submits a question, the LLM module receives both the query and the retrieved context from the Storage module, constructs a prompt, and submits it to the LLM runtime. The model's response is then streamed back to the frontend using Server-Sent Events (SSE), providing a smooth conversational experience. To help users navigate past conversations, Echelon automatically generates a chat title based on the session content (see Appendix B for the prompt used). The module also supports advanced capabilities, such as vision-language interaction, by forwarding both text prompts and image-based inputs, like PDFs, to multimodal models when appropriate. Ultimately, the LLM module encapsulates all logic related to prompt construction, model invocation, and response delivery, playing a central role in enabling dynamic and personalized interactions.

The final component of the backend is the web server itself, which acts as the central coordinator for all system operations. Built using the Axum framework in Rust, the web server exposes the functionality of the Vectorization, Storage, and LLM modules through a RESTful API. Incoming requests are processed as independent transactions, allowing the backend to scale horizontally and serve multiple users in parallel.

Security and authentication are enforced in this module through a JWT-based system. Upon login or registration, users receive a signed token containing identity-specific claims, such as user ID and expiration timestamp. These tokens are included in all subsequent requests, allowing for secure access without the overhead of maintaining server-side sessions. Tokens have a default expiration of several days, ensuring a balance between usability and

security.

3.3 Compute Component

The Compute Component forms the backbone of Echelon's inference capabilities. By isolating computationally intensive tasks, it ensures efficient resource management and scalability. Echelon leverages Ollama, a lightweight inference server built around llama.cpp, specifically chosen for its balance between simplicity, performance, and compatibility with open-source LLMs.

Ollama exposes models via a RESTful API, enabling seamless interaction with Echelon's backend. This design ensures computationally heavy inference tasks – such as generating real-time answers or parsing multimodal inputs like student transcripts – are handled reliably. Furthermore, the Compute Component is containerized, facilitating rapid deployment, horizontal scaling, and ease of updates or model replacements as newer, more performant models become available.

3.4 Database Component

The Database Component ensures robust data management and persistent storage across specialized services tailored to specific data needs. By adopting three distinct databases, Echelon leverages each database's strength, significantly enhancing system efficiency and scalability.

PostgreSQL is used for structured, relational data due to its reliability, ACID compliance, and maturity in production environments. It efficiently handles sensitive data, such as user authentication details and application metadata.

Qdrant was selected explicitly for managing semantic vectors due to its optimized architecture for fast, accurate similarity searches. It ensures rapid retrieval of relevant context essential to Echelon's RAG process.

MinIO provides an S3-compatible storage layer, ideal for unstructured or semi-structured

data like student transcripts and uploaded documents. Its selection is driven by ease of integration, scalability, and robust support for object-based storage.

Each database is independently containerized and deployed, supporting modular scalability and facilitating backups, disaster recovery, and easy maintenance. Together, these databases offer a balanced, performance-optimized foundation that meets the diverse storage requirements inherent to a sophisticated, AI-driven advising system.

4 Personalization

A defining feature of Echelon is its ability to tailor responses based on a student's individual academic record. While traditional tools offer only generalized insights, Echelon uses vision-enabled language models to extract detailed academic information from user-uploaded documents – most notably unofficial transcripts.

Figure 4 shows an example of Echelon's frontend interface rendering a student's academic profile based on a user's question. This profile is generated from the student's transcript and includes structured information such as declared majors, completed coursework, GPA, and academic standing. The display highlights Echelon's ability to personalize responses by grounding language model outputs in individual academic data.

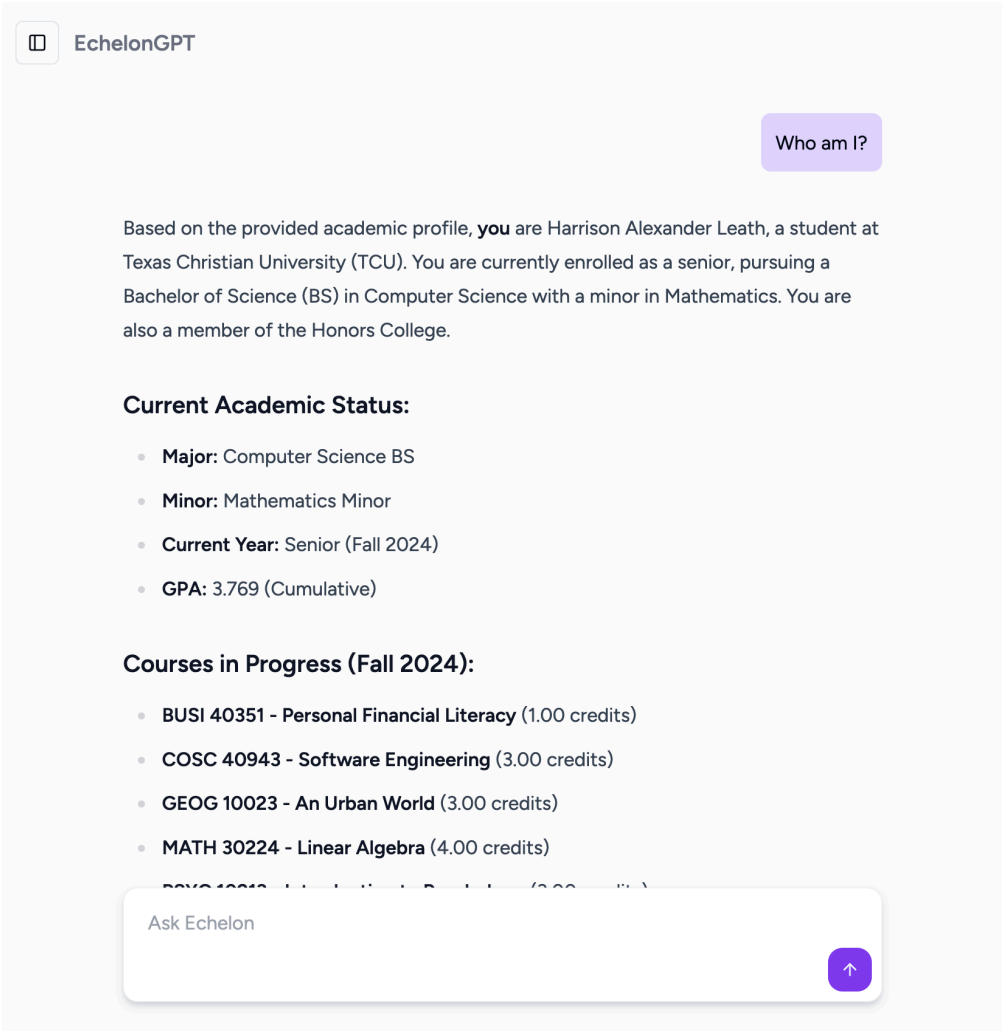

Figure 4: An example of Echelon's frontend interface describing a student's academic profile.

When a student uploads a transcript, Echelon forwards the document to a vision-language model, such as Qwen 2.5 VL 72B, capable of performing structured extraction from visual layouts (Bai et al., 2025). The model parses key fields such as the student's name, declared major, completed courses, GPA, and academic standing. This structured output is stored in Markdown format and associated with the student's Echelon account. See Appendix C for the prompt used to generate a student's academic profile.

Figure 5 displays the Markdown output produced by Echelon's transcript parsing pipeline. This structured format captures key academic details extracted from the student's transcript, including course history, GPA calculations, and declared programs. The output serves as

the foundation for personalizing responses within the RAG pipeline and enables consistent, machine-readable representations of academic records.

```
# TCU Undergraduate Academic Transcript Analysis

## 1. Student Information
- **Name:** Leath, Harrison Alexander
- **Student ID:** 110421172
- **Program:**
  - Major: Computer Science BS (Science & Engineering UGRD)
  - Minor: Mathematics Minor
  - Special Programs: Honors College, General Business Minor

## 2. Academic Progress Timeline
- **Start Date:** Fall 2021
- **Current Status:** Enrolled in Fall 2024 term (courses in progress/
ungraded).

## 3. GPA Summary
| **Term**        | **Term GPA** | **Cumulative GPA** |
|-------------------|-------------|-------------------|
| Fall 2021       | 4.000       | 4.000             |
| Spring 2022     | 3.734       | 3.867             |
| Fall 2022       | 3.835       | 3.857             |
| Spring 2023     | 3.750       | 3.833             |
```

Figure 5: The output of Echelon's transcript parsing pipeline in Markdown format. This is an example academic profile used to provide context on a student in the RAG pipeline.

The resulting academic profile acts as a source of truth during inference. When a student asks a question, Echelon combines three sources to construct a full prompt: the user's query, a set of semantically relevant passages retrieved from the vector database, and a cached academic profile derived from the transcript. This combined context is then sent to the language model, allowing it to generate an answer that is not only grounded in official documentation but also tailored to the student's current academic path.

This prompt construction pipeline enhances both the relevance and factual accuracy of

Echelon's responses. Rather than relying solely on the model's pretraining, Echelon injects real-time context and personal academic data to guide generation. This approach allows the system to answer nuanced questions like "Am I on track to graduate?" or "How close am I to completing my major?" in a way that general-purpose LLMs cannot.

5 Echelon vs. General-Purpose LLMs

While Echelon builds on similar LLMs to those used in popular services like ChatGPT, its architecture and intended use case differ in several major ways. ChatGPT is a general-purpose assistant trained on broad internet data and optimized for flexibility across many domains. In contrast, Echelon is fine-tuned for a specific domain: academic advising. By grounding its responses in institutional documents and enriching prompts with student-specific data, Echelon demonstrates domain mastery that general-purpose models cannot.

This domain grounding leads to a second key distinction: accuracy and hallucination rate. While recent versions of applications like ChatGPT can access external information through plugins or browsing capabilities, they still risk generating outdated or incorrect details if specific institutional documents are not publicly accessible or frequently updated. Echelon addresses this issue directly through its RAG pipeline, ensuring responses always reflect the most current institutional documents and the student's own academic data.

A third distinction concerns data privacy. According to OpenAI's privacy policy, ChatGPT stores user interactions on a centralized server (OpenAI, 2024). Echelon, conversely, adopts a privacy-first architecture, ensuring student data remains securely stored on TCU's institutional servers, eliminating reliance on external cloud providers.

Finally, Echelon supports queries about proprietary and institution-specific knowledge. These documents are often unavailable to the public and are therefore inaccessible to tools like ChatGPT. For example, a department chair might upload internal guidelines detailing eligibility criteria for an honors thesis or specific departmental graduation requirements,

enabling Echelon to provide accurate, tailored responses to student inquiries about these specialized academic procedures. This level of private customization enables Echelon to operate not just as an academic Q&A tool, but as an intelligent interface for accessing protected institutional knowledge.

Together, these four design choices – domain specialization, response grounding, privacy-first architecture, and support for proprietary context – position Echelon not as a replacement for ChatGPT, but as a purpose-built system that excels in information retrieval, starting with academic advising.

6 Deployment

Echelon was written using exclusively open-source libraries and frameworks, which allows its deployment to be infrastructure-agnostic. This key development principle is beneficial for institutions like TCU as it enables deployment flexibility across internal servers or cloud providers, thus allowing precise control over data locality and privacy. For the duration of the spring 2025 semester, Echelon was deployed across a combination of DigitalOcean and Runpod products with funding provided by a SERC grant. DigitalOcean and Runpod were selected for their simplicity, cost-effectiveness, and ease of management, particularly beneficial for rapid prototyping and scalable deployments.

On DigitalOcean, a virtual machine (VM) hosts both the frontend and backend services in isolated containers. These services are managed via Docker Compose, which enables streamlined orchestration and reproducible environments across development and production. Sensitive configuration values, such as database credentials and Runpod endpoints, are managed using environment variables loaded from .env files. Inbound web traffic is routed through an Nginx reverse proxy which handles TLS termination using Let's Encrypt certificates and directs requests to the appropriate service.

The PostgreSQL database is provisioned using DigitalOcean's managed database service

and includes automated daily backups. Qdrant, the vector database, is hosted via Qdrant Cloud, ensuring high availability and performance for semantic similarity search. At the time of writing, MinIO is a planned addition. It will be deployed on a dedicated VM to support file uploads such as student transcripts.

To support scalable inference workloads, Echelon's LLM runtime is deployed using Runpod, which provisions GPU-backed compute instances on demand.

While the current deployment targets small-scale usage, the architecture is built for horizontal scalability. Each backend module can be replicated across multiple containers or services, and compute-heavy operations – such as LLM inference – are already offloaded to external runtimes.

At the time of writing, the full system is accessible at `echelongpt.com`.

7 Limitations and Future Work

While Echelon's current architecture supports personalized, grounded, and real-time academic advising, several opportunities exist to expand its functionality, improve performance, and broaden its use cases.

One of the most immediate areas for improvement is in the efficiency and accuracy of vector search. Although the system currently uses Qdrant and BGE-large-en-v1.5 for semantic similarity, future iterations may experiment with hybrid search techniques to further improve retrieval precision. These enhancements could reduce the amount of irrelevant context retrieved from the database, which in turn, would improve final response quality.

Scalability is another priority. While Echelon has been designed with horizontal scaling in mind, future deployments could benefit from incorporating GPU autoscaling, load balancing, and more advanced observability tooling. This would allow Echelon to support many more concurrent users without degrading performance.

There is also an opportunity to enhance citation and explainability mechanisms. Cur-

rently, Echelon provides grounded responses based on retrieved documents but does not explicitly cite to the user which chunks were used to support each part of the answer. Adding inline citations and confidence scores could improve user trust and offer a clearer audit trail of how each answer was generated.

To expand its AI feature offerings, Echelon could begin exploring student goal modeling and proactive advising. Instead of waiting for students to ask questions, the system could identify at-risk students, suggest optimal course paths, or notify users when a graduation requirement is unmet. These features would require integrating predictive modeling on top of transcript data and university records.

It is also important to note that the end-to-end transcript parsing pipeline, though fully designed – including prompts and output formats – awaits compatibility support from Ollama to fully automate structured data extraction via vision models. This includes the ability for students to upload transcripts and generate academic profiles. Currently, this functionality is a manual process and can be automated once Ollama support for the Qwen VL 2.5 model is complete.

Finally, while academic advising is the current focus, Echelon's architecture is domain-agnostic. Any department within TCU, or even any institution, can deploy its own instance of Echelon by providing relevant internal documents. Future deployments could readily adapt Echelon for departments like Admissions, Athletics, IT Support, and Faculty Onboarding, providing institution-wide value wherever structured documentation exists.

8 Conclusion

Echelon demonstrates how cutting-edge AI technologies can be harnessed to solve real-world, domain-specific challenges in higher education. By combining semantic embeddings, LLMs, and retrieval-augmented generation, the system delivers personalized academic guidance grounded in institutional data and tailored to each student's academic record. Unlike

general-purpose tools, Echelon is intentionally scoped to the advising domain, offering high accuracy, privacy-conscious deployment, and support for proprietary content that traditional LLMs cannot access.

Throughout its development, Echelon was built using open-source components and deployed on modular infrastructure, ensuring flexibility, scalability, and institutional control over student data. The use of Rust for the backend enabled high-concurrency performance, while integrations with Qdrant, PostgreSQL, and MinIO formed a robust storage stack capable of handling both structured and unstructured data. Meanwhile, multimodal models like Qwen 2.5 VL unlocked the ability to interpret transcripts visually, enabling deep personalization in responses.

Though fully functional and publicly accessible at `echelongpt.com`, Echelon remains an evolving system. Some features, such as end-to-end transcript parsing via vision models, are fully designed but pending support from dependencies like Ollama. Future versions of Echelon aim to improve vector retrieval accuracy, expand explainability, and introduce predictive advising to anticipate student needs before they arise.

More broadly, Echelon is a template – not just a tool. Echelon serves as a replicable model not only within TCU or higher education but also across diverse institutions or industries where structured knowledge needs accessible, accurate, and secure dissemination.

Ultimately, Echelon reframes academic advising not as a bottleneck, but as an opportunity for augmentation: amplifying the impact of professors and advisors while empowering students with the answers they need to succeed.

9 Acknowledgements

I would like to express my deepest gratitude to Dr. Bingyang Wei for his invaluable guidance, enthusiasm, and mentorship throughout this project. I am especially thankful for the many opportunities he provided for me to present my research to the TCU community.

I would also like to thank the entire Go2AI team, especially Dr. Liran Ma, Dr. Ze-Li Dou, and Dr. Yang Yang, who first introduced me to the field of machine learning and have invested in my growth over the past two and a half years.

I am sincerely grateful to Dr. Bo Mei, and Dr. Ze-Li Dou for serving on my thesis committee and offering their thoughtful feedback and support.

I remain thankful to the Department of Computer Science at TCU for fostering a collaborative and encouraging academic environment and for awarding this thesis a SERC grant which helped make this research possible.

Finally, I am deeply grateful to my friends and family for their unwavering support and encouragement throughout this journey.

10 References

Bai, S., Chen, K., Liu, X., Wang, J., Ge, W., Song, S., ... Lin, J. (2025). *Qwen2.5-vl technical report.* Retrieved from https://arxiv.org/abs/2502.13923

Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? . In *Proceedings of the 2021 acm conference on fairness, accountability, and transparency* (p. 610–623). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3442188.3445922 doi: 10.1145/3442188.3445922

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... Amodei, D. (2020). *Language models are few-shot learners.* Retrieved from https://arxiv.org/abs/2005.14165

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *Bert: Pre-training of deep bidirectional transformers for language understanding.* Retrieved from https://arxiv.org/abs/1810.04805

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... Kiela, D. (2021). *Retrieval-augmented generation for knowledge-intensive nlp tasks.* Retrieved from https://arxiv.org/abs/2005.11401

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient estimation of word representations in vector space.* Retrieved from https://arxiv.org/abs/1301.3781

Mistral AI. (2025). *Mistral-small-24b-instruct-2501.* https://huggingface.co/mistralai/Mistral-Small-24B-Instruct-2501. (Instruction-tuned 24B parameter language model, released by Mistral AI)

OpenAI. (2024). *Openai privacy policy.* https://openai.com/policies/privacy-policy. (Accessed: 2025-04-23)

Reimers, N., & Gurevych, I. (2019). *Sentence-bert: Sentence embeddings using siamese bert-networks.* Retrieved from https://arxiv.org/abs/1908.10084

Rich Harris et al. (2025). *Svelte.* https://svelte.dev. (Accessed: April 1, 2025)

Stellic Inc. (2025). *Stellic.* https://www.stellic.com/. (Accessed: April 1, 2025)

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2023). *Attention is all you need.* Retrieved from https://arxiv.org/abs/1706.03762

Xiao, S., Liu, Z., Zhang, P., & Muennighoff, N. (2023). *C-pack: Packaged resources to advance general chinese embedding.*

A Prompt Used for Q&A

"You are Echelon, TCU's AI academic advising assistant designed to help students and faculty. You are talking with a student who has an academic profile. Every time the user says 'I' they are referring to the academic profile. If courses are listed as '0.00' for 'Earned' or 'Points', assume these classes are currently being completed this semester. You should determine which year the student is currently enrolled as when you answer questions. Use the following pieces of retrieved context to answer the question with accurate information about TCU courses, programs, and policies. Format your responses in a clear, concise manner using markdown for readability when appropriate. When discussing courses, include course codes and relevant details such as prerequisites when available. Review the chat history before answering to maintain conversation continuity. If information is available in the context, provide specific, actionable guidance. If information is partially available, clearly indicate what is known and what might require further clarification. If you don't know or can't find the answer in the provided context, acknowledge this honestly and suggest how the user might find the information elsewhere (e.g., 'You may want to check with your academic advisor or the registrar's office for the most current information about this'). Never mention that you are using retrieved context or vector search in your responses."

B Prompt Used for Generation of Chat Titles

"Analyze the following conversation and generate a concise, descriptive title (maximum 8 words) that captures the main topic or purpose. The title should: 1. Identify the core subject matter or question being discussed. 2. Be specific enough to distinguish this conversation from others. 3. Avoid generic phrases like 'Discussion about' or 'Conversation regarding.' 4. Prioritize the earliest messages, which typically establish the conversation's purpose. 5. Capture any unique aspects, project names, or specific requests. 6. Be formatted in title

case. Output only the title as plain text, without any formatting, quotation marks, or additional explanations."

C Prompt Used for Transcript Parsing

"You're analyzing a TCU (Texas Christian University) undergraduate academic transcript. It uses two columns per page to describe the courses and performance of a student. Parse and comprehensively describe the following information: 1. Student's name, ID, and program details (major, minor, special programs) 2. Academic progress timeline (when they started, current status) 3. Calculate and verify the GPA for each term and cumulative GPA 4. List all courses taken and grades received 5. Identify honors and achievements (Dean's List, etc.) 6. Total the transfer credits applied and list every class that was credited 7. Calculate total credits attempted and earned 8. Highlight strengths and areas of academic focus based on course selection and performance Present this information in a clear, organized markdown document that shows the student's academic journey and achievements."

D Access to Source Code

The source code for Echelon can be found on GitHub at `https://github.com/leathalman/echelon`. If access to the code is no longer available for any reason, contact the author at `harrison.leath@tcu.edu`. The demo version of Echelon hosted at `echelongpt.com` will be available until the end of the Spring 2025 semester.