Uni-Bot: A chatbot for Bennett University.

Team Name: SoftwareDevelopers

Jay Malhotra, E23BCAU0050

Suryesh Pandey, E23BSCU0003

Academic Session 2024-2025

Table of Contents

Abstract

UniBot is an AI-driven chatbot designed to simplify access to faculty-related information at Bennett University. The system allows students and professors to quickly retrieve details such as cabin locations, teaching specializations, and real-time timetable updates through natural language queries. Built using Flask for the backend and powered by a custom-trained intent classification model, the chatbot intelligently interprets user queries, resolves pronouns using coreference resolution, and interacts with an SQLite database to fetch accurate information. Specialized fields such as professor backgrounds are dynamically extracted using external data sources like Proxycurl and the university's website. By automating traditionally manual tasks, UniBot improves efficiency, enhances the campus experience, and showcases practical applications of NLP techniques in educational environments.

Introduction

UniBot is an intelligent chatbot designed specifically for Bennett University to streamline access to campus-related academic information. Students, professors, and visitors often face difficulties in quickly finding essential details such as professor cabin locations, specializations, and timetables. UniBot addresses these challenges by offering a simple conversational interface where users can ask natural-language questions like "Where is Alok Mishra's cabin?" or "What does Madhushi Verma specialize in?" and receive immediate answers. By minimizing reliance on static notice boards, scattered information, and manual inquiries, UniBot aims to make campus life more convenient, efficient, and digitally accessible. This introduction outlines the existing inefficiencies in accessing academic information and presents UniBot as a focused, AI-driven solution using Natural Language Processing (NLP) and real-time database integration.

Background

At Bennett University, obtaining basic academic information typically involves manually checking notice boards or asking multiple people, which often results in delays, confusion, or misinformation. For instance, students currently must physically search for professor cabin numbers from outdated or incomplete lists posted in blocks like the N Block. Similarly, understanding a professor's specialization or finding out their availability requires direct communication, often without guaranteed success. Recognizing these gaps, UniBot was developed as a chatbot tailored specifically for Bennett University. It integrates sample internal data for cabin locations and timetables, while real-time specialization details are fetched dynamically from external sources such as the university website or LinkedIn profiles. By bridging the gap between users and accurate, up-to-date information, UniBot modernizes the way academic support services are delivered within the university environment.

Problem Statement

Accessing academic and faculty-related information at Bennett University currently presents several challenges:

- **Cabin Location Discovery:** Students often rely on outdated or incomplete physical notices to find professor cabin locations, leading to confusion, time wastage, and unnecessary campus navigation difficulties.
- **Specialization Information:** Details about professor backgrounds and specializations are scattered across different platforms and are not always up-to-date, making it difficult for students to approach the right faculty for academic or research guidance.
- **Timetable Accessibility:** Professors and students have no centralized, conversational platform to quickly check class schedules, upcoming lectures, or free periods of professors resulting in frequent manual inquiries and administrative load.
- **Natural Language Interaction Gap:** Traditional search portals and noticeboards require users to manually search or scroll through information, lacking the convenience of asking a natural-

language question like "Where is Alok Mishra's cabin?" or "When is my next class?" and getting an instant, clear response.

- **Real-time Updates:** While professor specialization is dynamically fetched through sources like LinkedIn and the official university site, no prior solution incorporated external data enrichment into a chatbot for the university environment.
- **Niia:** Our own universities chatbot answers administrative queries and redirects you to other pages instead of answering the questions directly. Moreover, it's faculty page only has 1 line with the rest being random HTML data. Our chatbot would focus on the finer details, such as a professor's cabin location or availability. We would focus on the day-to-day challenges instead which impact students the most.

Objectives/Features

The **UniBot Chatbot** is designed as a web-based solution to provide an intelligent, real-time, and conversational way of accessing academic information at Bennett University.
The chatbot allows users to:

- **Locate Professor Cabins:** By simply asking the chatbot, users can instantly find which block and cabin a professor is seated in, minimizing time spent searching campus buildings.
- **Fetch Professor Specialization:** Using real-time scraping from LinkedIn, students can easily understand a professor's academic specialization and background.
- **View and Query Timetables:** Professors and students can view upcoming classes, free periods, and the next scheduled lectures without navigating complicated portals. Professors can also query when they are free or when their next class is.
- **Natural Language Query Handling:** Users interact conversationally using everyday language, and the chatbot intelligently understands and responds without requiring formal queries or rigid keyword matching.
- **Real-Time Data Fetching:** For specialization queries, the chatbot enriches its responses by dynamically fetching and summarizing real-world profile information from trusted external sources.
- **Session-Based Memory:** The chatbot maintains context during a conversation by remembering previously mentioned professor names, allowing users to ask follow-up questions without repeating names.
- **Secure and Private:** Users must log in to access the chatbot. Professors can view their own timetables securely after authentication.

Software Requirement Specification

Functional Requirements

1. **User Registration and Login:** Students and professors must be able to register and log in using their university-provided email ID and password.
2. **Secure Authentication:** Only verified users (students or professors) can access chatbot services after logging in.
3. **Context-Aware Chat:** The chatbot should maintain session memory to understand follow-up queries based on previous conversations (e.g., remembering the professor's name).
4. **Cabin Location Query:** Students can ask for a professor's cabin location conversationally and receive a quick, accurate response.
5. **Fetch Specialization Information:** The chatbot fetches real-time professor specialization data from LinkedIn and the official Bennett University website using Proxycurl and scraping mechanisms.
6. **View Timetable:** Professors can view and manage their timetable after logging in securely, with data loaded automatically upon login.

7. **Query Free Time Slots:** Users can ask when a professor is free, and the chatbot calculates based on timetable entries, showing available slots.
8. **Get Next Class:** Professors and students can query when their next class is scheduled today or on any other specified day.
9. **Day-Specific Timetable Queries:** Users can ask about classes on a specific day (e.g., "What are my classes on Tuesday?"), and the chatbot filters timetable entries accordingly.
10. **Real-Time Intent Classification:** User queries are classified dynamically into intents using a trained deep learning model (Keras Sequential model) ensuring accurate understanding.
11. **Friendly Error Messages:** If queries are unclear or no information is found, the chatbot provides a user-friendly message guiding the user properly.
12. **Natural Language Understanding:** The chatbot supports everyday conversational language without needing strict syntax.
13. **Timetable Management:** Professors can update their timetable via a web interface that allows editing and saving of slots dynamically.
14. **Session Management and Multi-Chat History:** Each chat session is saved separately, allowing users to view previous chats in a sidebar interface.
15. **Scraping and Data Enrichment:** In case of missing information, the chatbot scrapes data intelligently rather than relying only on static data.

Non-Functional Requirements

1. Performance
- Fast response times (under 2 seconds) for regular chatbot queries.
- Scalable to support hundreds of concurrent users (students + professors).

2. Security
- Secure login and session management with Flask session encryption.
- External data fetching through secured APIs like Proxycurl with API key protection.

3. Usability
- Professional and simple UI in chat and timetable views.
- Natural language query handling for an intuitive experience.

4. Reliability
- High availability through Docker containerization.
- Data stored reliably using SQLite database inside Docker containers.

5. Maintainability
- Modular codebase with Flask blueprints.
- Separate files for major components (intent model loading, database interaction, etc.).

Hardware Requirements

1. Developer Machines
- **Processor:** Minimum Intel i5 (quad-core) or Apple M1 chip equivalent.
- **RAM:** Minimum 8GB; 16GB recommended for faster container execution.
- **Storage:** Minimum 20GB free space (codebase, Docker images, database files).
- **Network:** Stable internet for external API usage and model downloads.

2. Server/Deployment (if hosting)
- **Processor:** Multi-core server (4 cores or more).
- **RAM:** 16GB minimum for production deployments.
- **Storage:** SSD with minimum 50GB free space.
- **Network:** Static IP address and high-speed internet connection.

Software Requirements

1. For Windows Users

- **OS:** Windows 10/11 Professional or Education edition.
- **Development Tools:**
    - Docker Desktop for Windows (for containerized environment setup).
    - Visual Studio Code with Python and Docker extensions.
- **Database Tools:**
    - SQLite Browser for local database inspection.
- **Python Version:** 3.7 or higher installed inside Docker container.
- **Libraries:**
    - Flask, TensorFlow, Keras, NLTK, spaCy, BeautifulSoup, DuckDuckGo Search, OpenAI client library.
- **API Tools:** Postman for testing APIs.

2. For Mac Users

- **OS:** macOS Ventura or later recommended.
- **Development Tools:**
    - Docker Desktop for Mac (M1/M2 chip compatible version).
    - Visual Studio Code with extensions (Python, Docker).
- **Database Tools:** SQLite Browser (Mac version).
- **Python Environment:** Same as Windows (Python 3.7+ inside Docker container).
- **Libraries:** Same Python dependencies installed inside Docker container.

Docker ensures that whether the developer is on Mac or Windows, the environment (Python version, installed packages, database setup) remains **identical and isolated**, avoiding version mismatch issues and library installation conflicts that frequently occur natively between Windows and Mac systems. It also ensures better portability and reproducibility of the project.

Technical Requirements

- **Backend Development:** Python 3.7+ using Flask microframework.
- **Frontend:** HTML5, CSS3 (for login, registration, timetable upload pages).
- **Database:** SQLite used for lightweight, fast, file-based data storage.
- **Machine Learning Model:**
    - Intent Classification using a Keras Sequential Model (Embedding + Global Average Pooling + Dense Layers).
    - Saved as intent_model.h5.
- **External API Integrations:**
    - Proxycurl API for real-time LinkedIn profile fetching.
    - SerpAPI + BeautifulSoup for fallback scraping from Bennett University site.
- **Containerization:** Entire app packaged inside a Docker container using a custom-built Dockerfile.
- **Model and Data Management:**
    - Tokenizer and Label Encoder saved separately as .pkl files.
    - External NLP models loaded dynamically (SpaCy, NLTK).

Design (Diagrams)

Overall Design Architecture

The overall design architecture provides a structured blueprint for how various components of the chatbot system interact with each other. It highlights the flow of information between the user, the front-end interface, the Flask-based back-end server, the database, and external APIs like OpenAI and Proxycurl. This architectural representation is critical because it ensures that each module — from user authentication, to intent prediction, to timetable management — communicates effectively without redundancy or confusion. By visualizing these relationships, the design architecture helps the development team maintain a clear understanding of how data is processed, how user queries are handled, and how external resources are integrated. It also supports early identification of bottlenecks or security vulnerabilities, making the project more scalable, maintainable, and easier to debug during future enhancements. The diagram below presents a complete overview of the chatbot's system design and its major components.

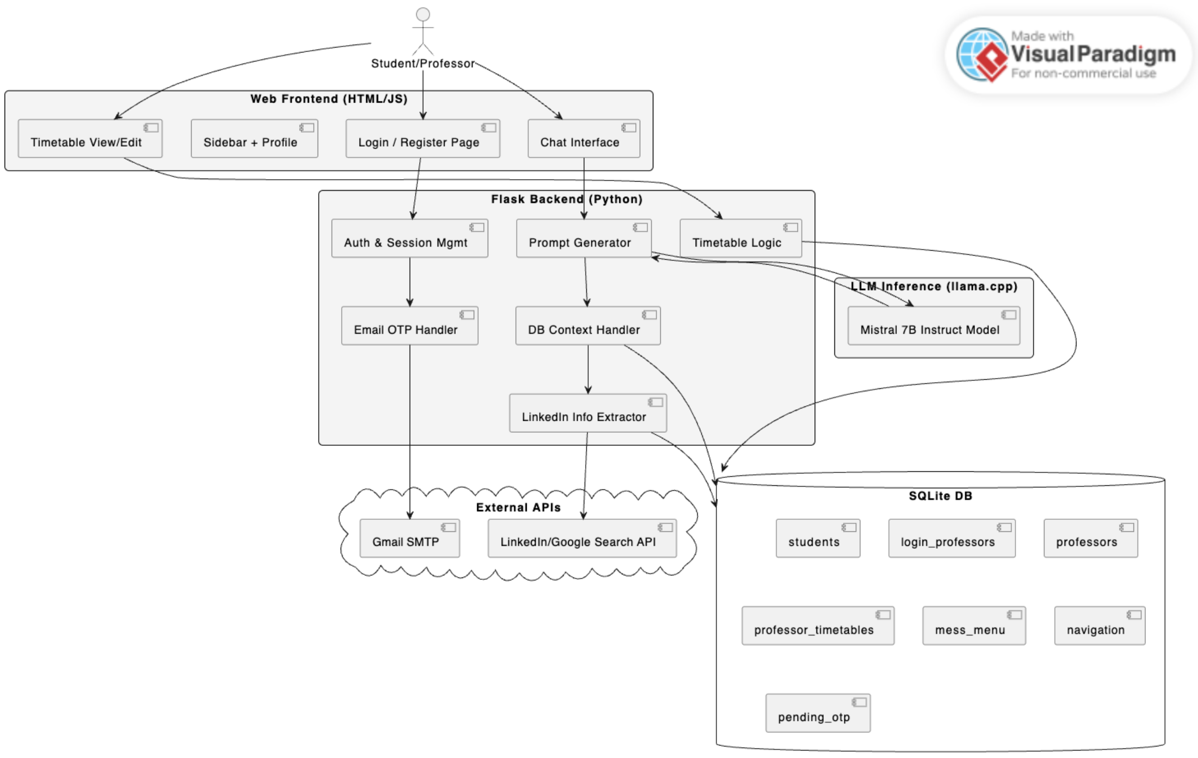


Fig. 1: Overall Design Architecture

Gantt Chart

A Gantt Chart is a project management tool that visually represents tasks over a timeline, showing their start and end dates, durations, and dependencies in a bar-chart format. For this project, it is particularly useful to plan and track key phases like registration, login, timetables, cabin locations, coreference resolutions, and more. It helps in visualizing the event timeline, ensuring deadlines are met, managing task dependencies (e.g., handling user queries with pronouns after resolving coreference resolution).

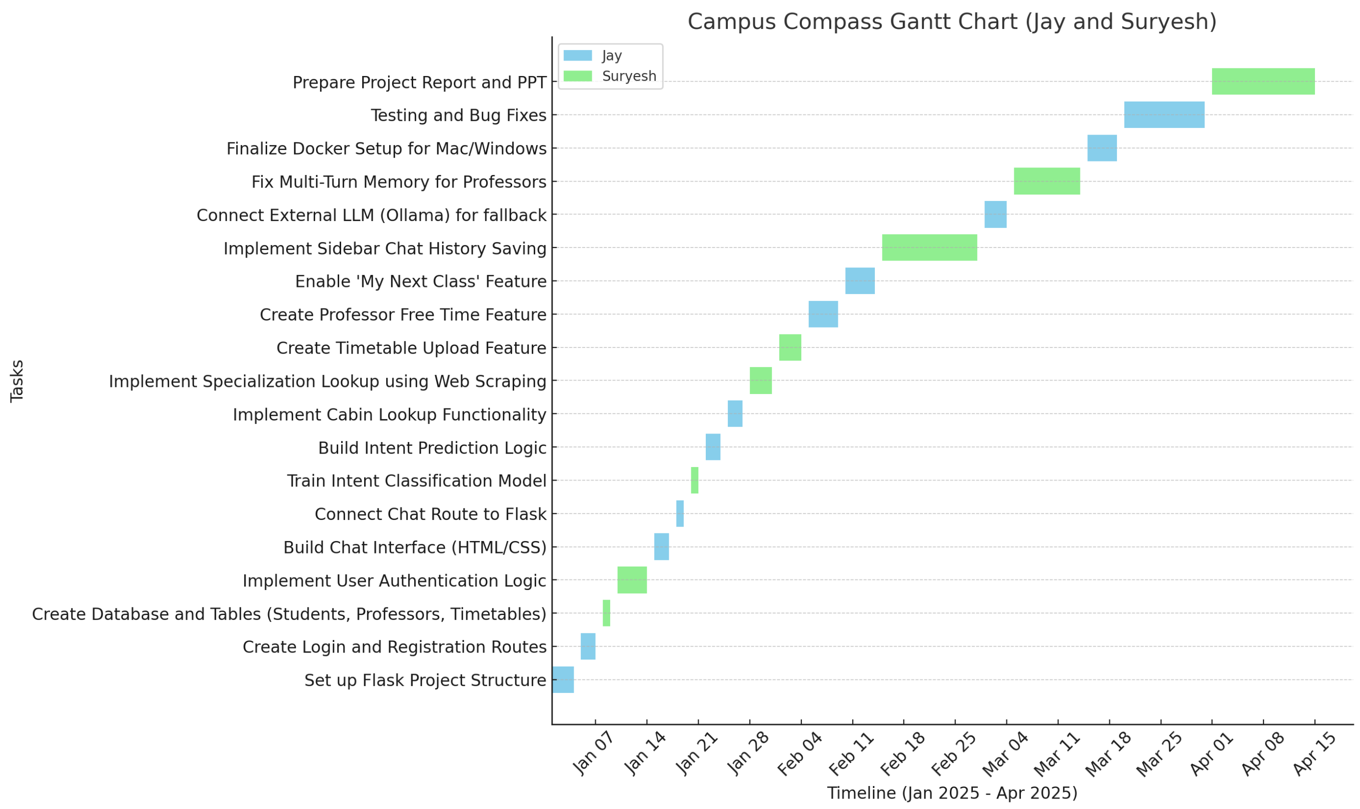


Fig. 1: Gantt Chart

Use Case Diagram

Use case diagrams play an integral role in visualizing the interactions between users and the system. A use case diagram is a graphic depiction of the requirements of a system and its interaction with external users, systems, or processes. These diagrams are essential because they outline the system's functionality from the user's perspective, providing a clear and concise map of all user interactions and the expected system behavior in various scenarios. By illustrating the relationships and dependencies between the system and its actors, use case diagrams help ensure that the development team fully understands the user's needs and system requirements. They facilitate communication among project stakeholders, making it easier to identify necessary functionalities and potential issues early in the development process. Below is the use case diagram, which captures the comprehensive functionalities addressed by the application –

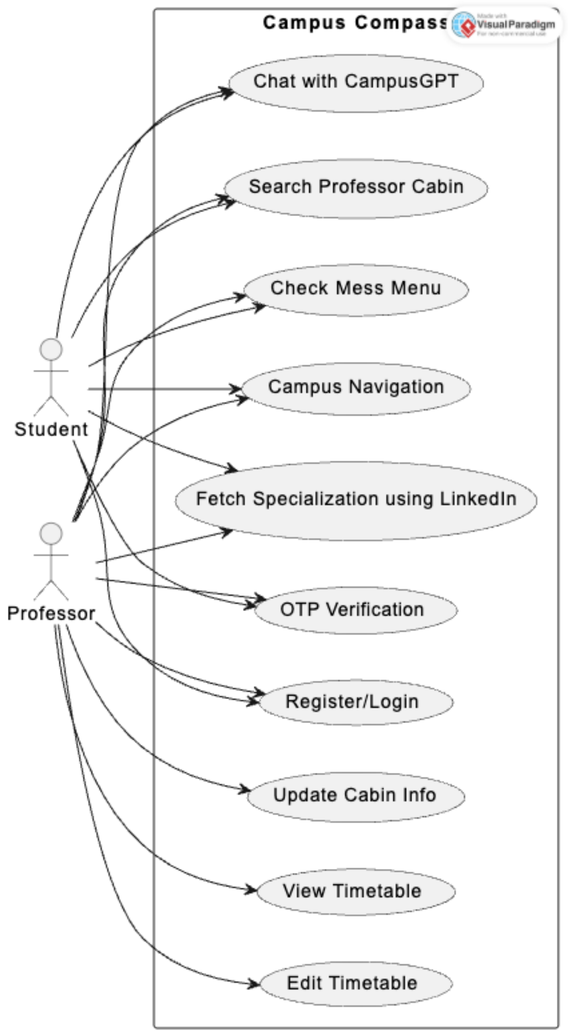


Fig. 1: Use Case Diagram.

Data Flow Diagram

The data flow diagram (DFD) provides a visual representation of how data moves throughout the chatbot system. It outlines the flow of information between different entities such as users, the web server, the database, and external APIs. Creating a DFD is important because it breaks down the system into simpler, manageable parts, making it easier to understand how inputs are processed into outputs at every stage. It also ensures that data dependencies, storage points, and external interactions are clearly documented and well-organized. By mapping out the data pathways, the development team can better identify potential risks, optimize communication between components, and ensure the system adheres to best practices in data management and security. The diagram below captures the end-to-end flow of data within the chatbot system, from user queries to final responses.

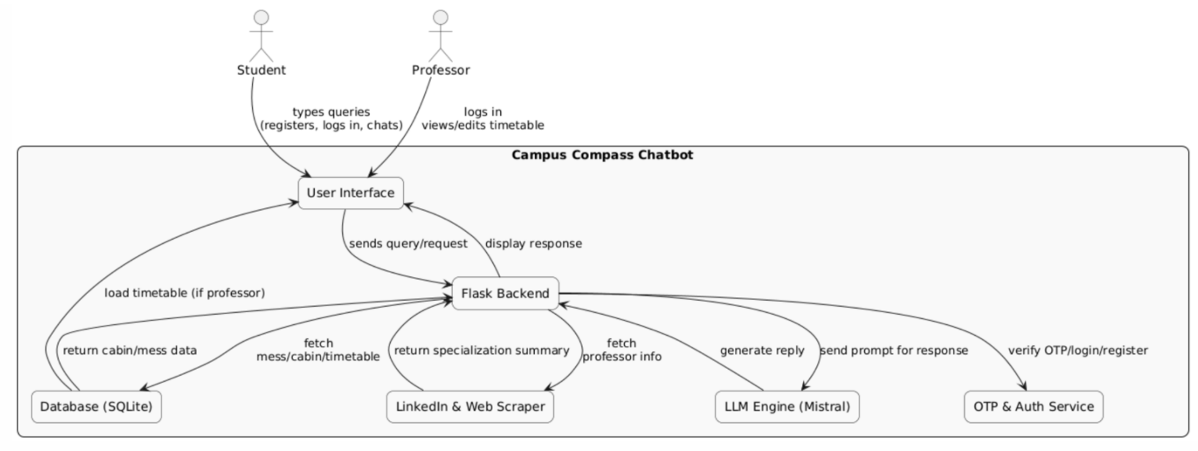


Fig. 2: Data Flow Diagram.

Activity Diagrams

Activity diagrams are utilized to illustrate the dynamic aspects of the system. These diagrams provide a detailed graphical representation of workflows showing the sequence of activities involved in various processes within the application. Activity diagrams help in visualizing the flow of control from one activity to another, including decisions, loops, and concurrent tasks.

The clarity provided by activity diagrams is invaluable for both developers and stakeholders to understand the procedural steps and business logic of different functionalities. They serve as a blueprint for implementing workflow management and enhance the understanding of complex operations, ensuring the system's behavior aligns with user expectations. Below are the activity diagrams -

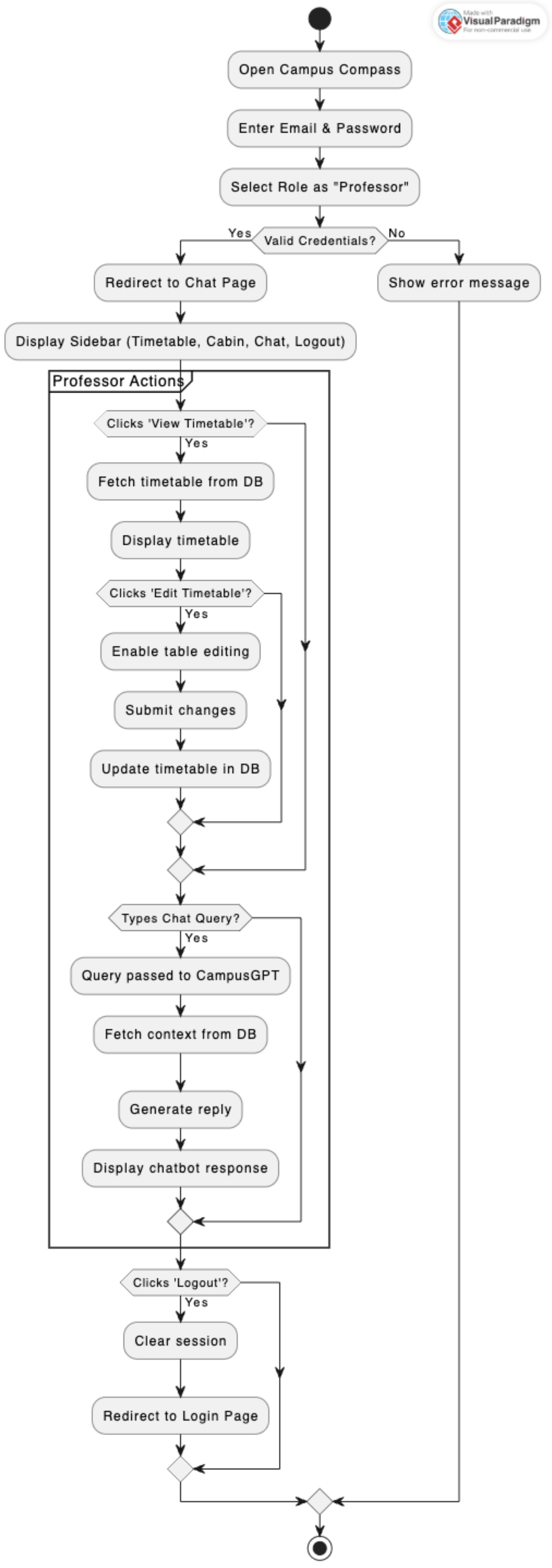


Fig. 4.1: Activity Diagram 1 - Professor.

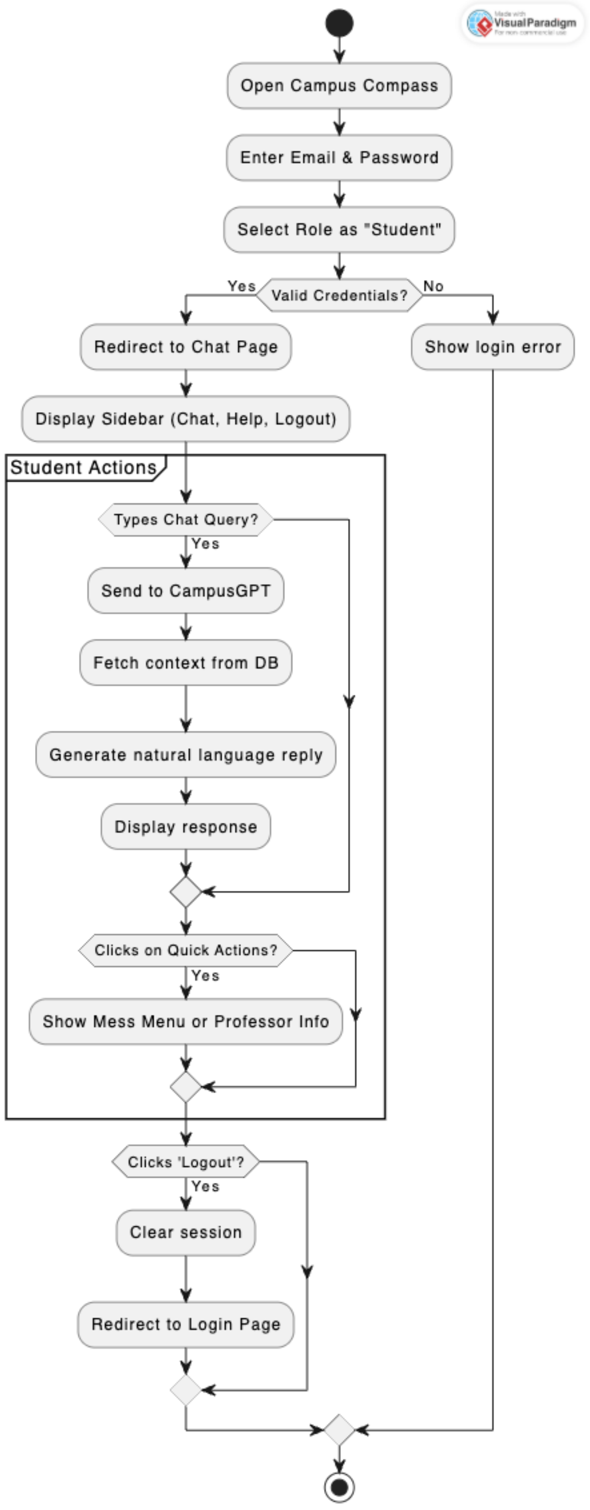


Fig. 4.2: Activity Diagram 2 - Student.

Sequence Diagrams

Sequence diagrams are essential for detailing the interaction between objects over time. These diagrams depict how objects in the system interact with each other and in what order these interactions occur, capturing the dynamic behavior of different parts of the system during runtime.

Sequence diagrams are particularly useful for visualizing the flow of messages, events, and actions between various components and entities, such as users, system interfaces, and databases. They help clarify complex functionality, ensuring that all system interactions are aligned with the intended design and business logic.

By providing a clear sequence of operations, these diagrams facilitate the understanding and troubleshooting of the functional flows within the application, aiding developers in building and integrating components effectively. Below is one of the sequence diagrams –

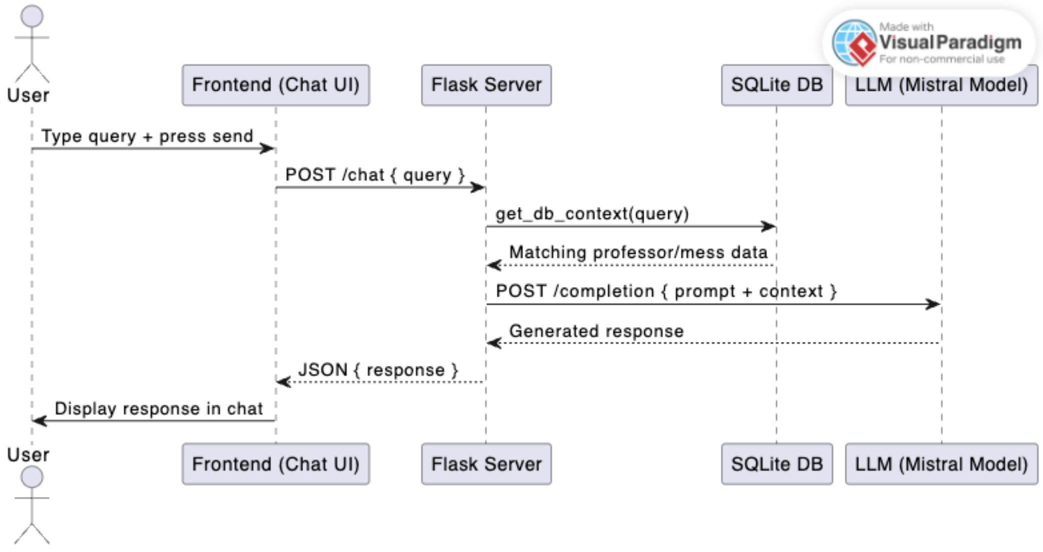


Fig. 5: Sequence Diagram.

Implementation Details

UniBot was developed as a web-based chatbot application, primarily using the Flask framework for backend operations and SQLite for database management. The system integrates Natural Language Processing (NLP) techniques to enable intelligent, conversational access to campus-related academic information for users at Bennett University.

Model Training and AI Integration

To understand and classify user queries, a custom, lightweight, deep learning intent recognition model was built using TensorFlow and Keras. The model architecture consisted of an Embedding layer, followed by a GlobalAveragePooling1D layer and two Dense layers, with the final layer using a softmax activation function for multi-class classification. Input patterns were tokenized using TensorFlow's Tokenizer API, padded to a maximum length of 20 tokens, and the corresponding intents were labeled and encoded using Scikit-learn's LabelEncoder.

The model was trained over 500 epochs, ensuring high accuracy and stability in recognizing user intents such as "get_cabin," "get_specialization," "ask_timetable," and "get_professor_free_time." After training, the model, tokenizer, and label encoder were serialized using pickle and stored for real-time inference during conversations.

Backend System and Database

Flask was used to create the backend server that handles user requests, routes, database interactions, and session management. Upon user login, the application loads the model, tokenizer, and label encoder into memory, enabling immediate processing of user queries.

The system maintains a structured SQLite database with the following primary tables:

- **professors** — stores professor names, specializations, and cabin locations
- **professor_timetables** — stores day-wise timetable details for professors
- **chat_sessions** — stores metadata about each user's chat sessions
- **chat_messages** — stores individual chat messages linked to their session IDs

For logged-in professors, their entire timetable is preloaded into session memory to enable faster access without repeated database queries.

Natural Language Processing Enhancements

To support conversational queries involving pronouns (e.g., "When is he free?"), SpaCy combined with NeuralCoref was integrated into the chatbot. This coreference resolution system intelligently maps pronouns to proper nouns based on previous context, ensuring continuity in multi-turn dialogues. Additionally, SpaCy's Named Entity Recognition (NER) system detects professor names within user inputs, with fallback mechanisms like rule-based detection and OpenAI's GPT API where necessary.

Specialization Extraction and Real-Time Data Integration

UniBot is capable of fetching real-world specialization information for professors dynamically. For this, the system integrates with Proxycurl's LinkedIn API to retrieve real-time profile information. In cases where Proxycurl is unavailable, the application falls back to scraping publicly available faculty data from the Bennett University website using BeautifulSoup.

Advanced NLP techniques such as tokenization, named entity extraction, and TF-IDF keyword extraction were applied on the scraped or retrieved texts to generate concise, meaningful specializations for each professor.

Deployment Using Docker

To ensure environment consistency, the entire application is containerized using Docker. Docker solves dependency conflicts, especially those involving older NLP libraries like NeuralCoref that require specific Python versions. It also allows the same setup to run seamlessly across both macOS and Windows machines, whether using native Docker Desktop on Windows or Docker containers inside a virtual machine on macOS.

Testing Strategy

1. Model Training Approach (intent_model.h5)

Given the relatively small size of the available dataset, it was decided to train the intent recognition model on the entire set of available examples without a separate train-test split. This approach allowed the model to learn from all the data, ensuring maximum exposure to the variety of language patterns associated with each intent. While an initial experiment with an 80-20 train-test split provided useful insights into model generalization, the final version prioritized full data utilization and manual testing to validate real-world chatbot performance. This strategy helped maximize model confidence and reliability during live user interactions.

Model Testing Results:

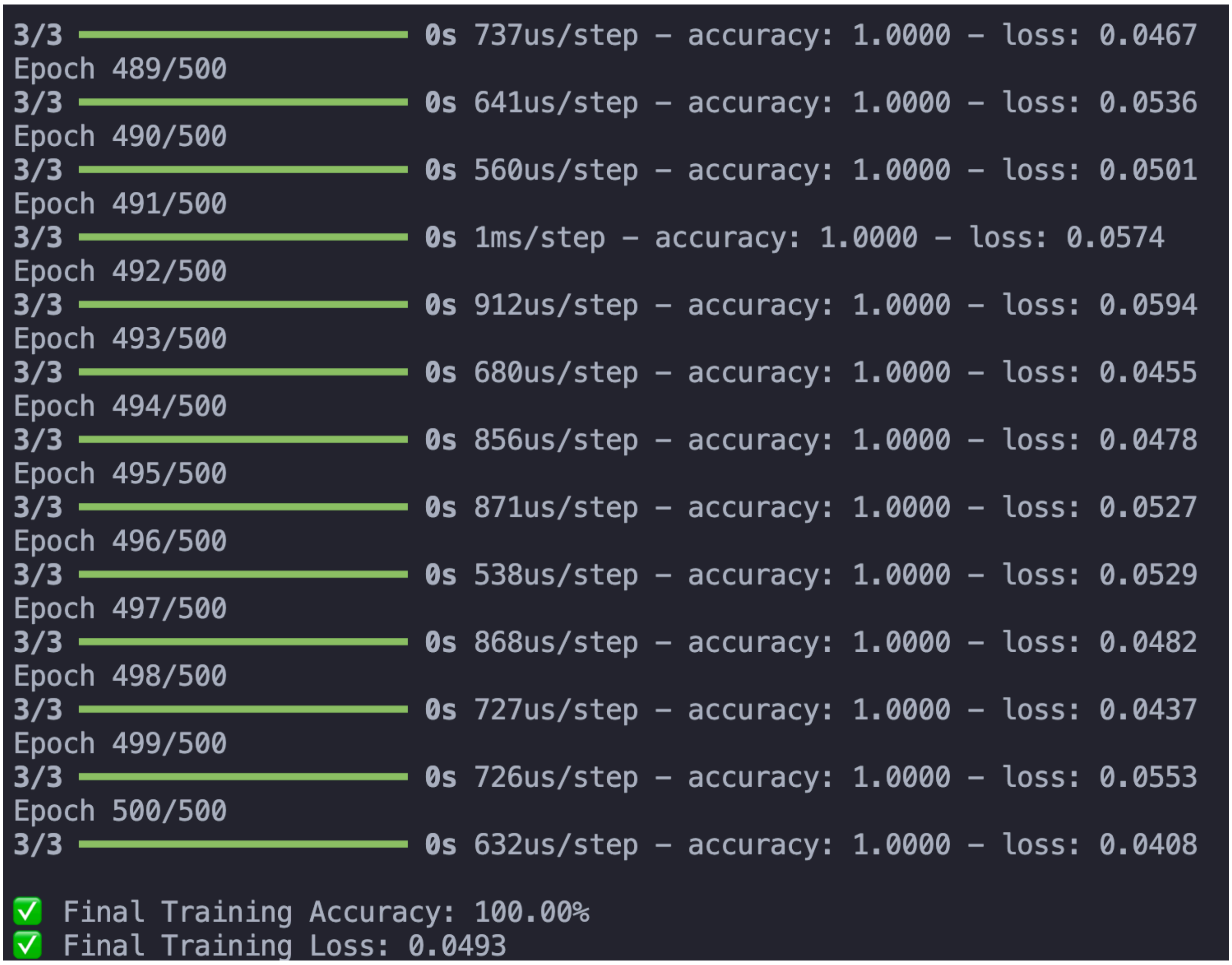
After training the intent classification model on the prepared intent_dataset.json without applying a train-test split, the model achieved a final training accuracy of 100% and a final training loss of 0.0493. This indicates that the model was able to fully learn the provided examples, as expected when training on a small and limited dataset.

Since no separate test set was used, traditional automated evaluation metrics such as test loss, test accuracy, precision, recall, and confusion matrix were not generated for the final model. Instead, manual testing was conducted to validate the model's performance in real-world scenarios. A variety of user queries were manually entered, including both training-like patterns and varied natural language phrasings.

Manual testing confirmed that the chatbot was able to correctly predict intents and generate appropriate responses for the majority of queries. Inputs related to intents like ask_timetable, get_cabin, and get_specialization were consistently recognized accurately. Even when slight variations in wording were introduced, the model maintained its understanding, thanks to the embedding-based design and effective tokenization.

A few earlier weaknesses observed during initial testing with a train-test split, such as misclassification of get_next_class and get_professor_free_time intents, were mitigated in the final model by allowing the model to learn from all available examples. Nevertheless, it is acknowledged that these intents remain slightly more vulnerable to errors during unusual phrasing, due to the very limited number of training patterns for them.

Overall, the model testing results validate that the intent recognition model performs reliably for the majority of intents. By training on the complete dataset and focusing on manual evaluation, the chatbot achieves a high degree of usability for practical deployment. Future improvements, such as expanding the dataset with more diverse and natural language examples, will further enhance the model's precision and conversational robustness.



```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 737us/step - accuracy: 1.0000 - loss: 0.0467
Epoch 489/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - accuracy: 1.0000 - loss: 0.0536
Epoch 490/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 560us/step - accuracy: 1.0000 - loss: 0.0501
Epoch 491/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 0.0574
Epoch 492/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 912us/step - accuracy: 1.0000 - loss: 0.0594
Epoch 493/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 680us/step - accuracy: 1.0000 - loss: 0.0455
Epoch 494/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 856us/step - accuracy: 1.0000 - loss: 0.0478
Epoch 495/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 871us/step - accuracy: 1.0000 - loss: 0.0527
Epoch 496/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 538us/step - accuracy: 1.0000 - loss: 0.0529
Epoch 497/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 868us/step - accuracy: 1.0000 - loss: 0.0482
Epoch 498/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 727us/step - accuracy: 1.0000 - loss: 0.0437
Epoch 499/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 726us/step - accuracy: 1.0000 - loss: 0.0553
Epoch 500/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 632us/step - accuracy: 1.0000 - loss: 0.0408

✅ Final Training Accuracy: 100.00%
✅ Final Training Loss: 0.0493
```

2. Manual Testing

After training and evaluating the model through automated methods, the chatbot was subjected to manual testing to ensure real-world usability and response accuracy. A variety of queries were manually entered into the chatbot interface, covering all defined intents.

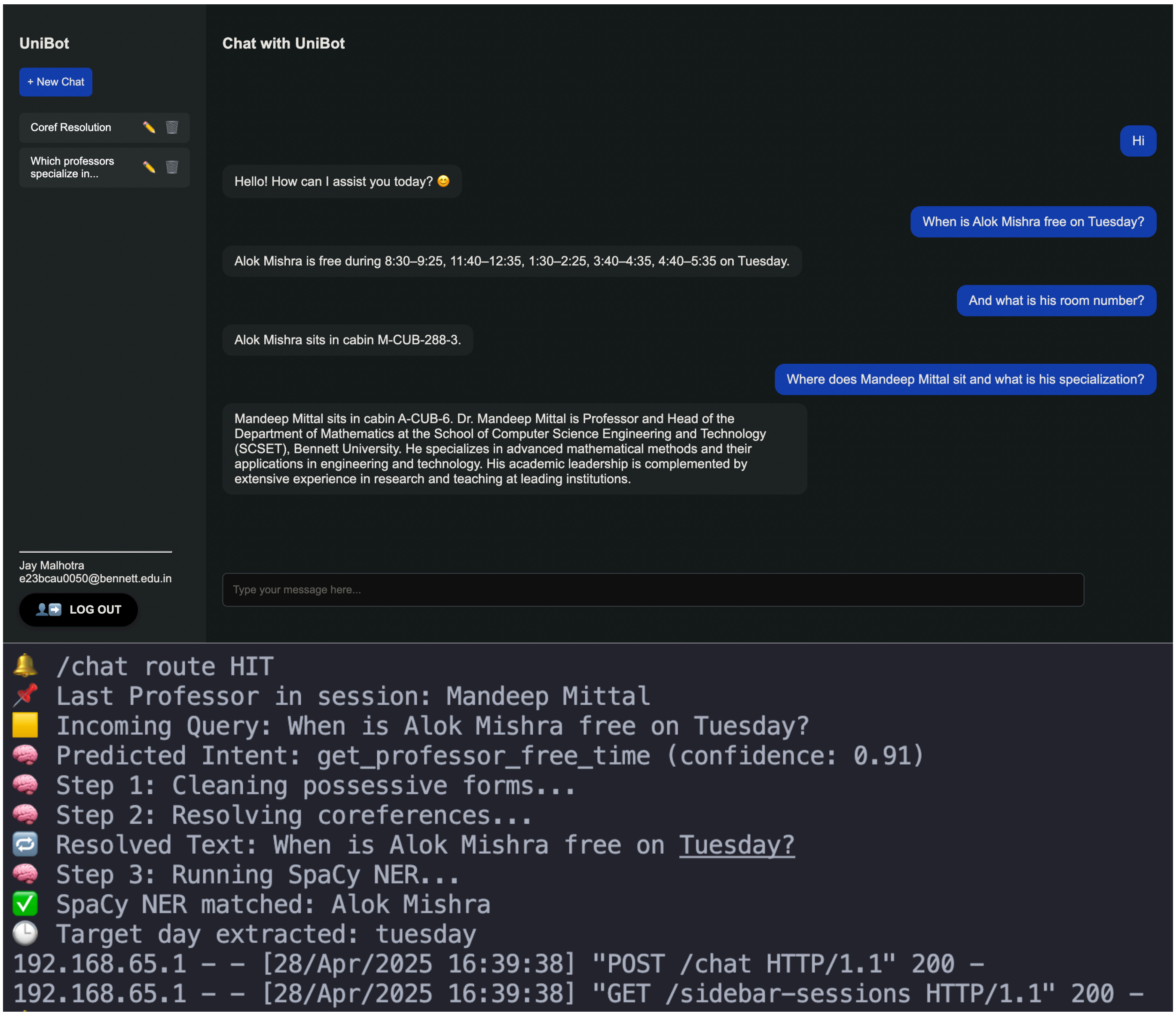
Queries included both exact matches to training patterns and variations with different phrasings to simulate natural human input. For example:

- "Can you tell me where Dr. Alok Mishra sits?"
- "What is Alok Mishra's cabin number? What does he specialize in?"
- "What's the specialization of Professor Sharma?"
- "What classes do I have on Tuesday?"
- "When is Alok Mishra free on Tuesday?"
- "Bye, see you later."

The chatbot was able to correctly predict the underlying intents for the majority of queries, even when the user input did not exactly match the training examples. The use of tokenization, padding, and label encoding allowed for sufficient flexibility in understanding minor variations in user language.

However, a few minor inaccuracies were observed when queries involved intents with very few examples during training, such as get_next_class. These rare misclassifications were consistent with the findings during automated model testing.

Overall, manual testing confirmed that the chatbot could reliably handle a range of real-world user queries, correctly classify intents, and route the conversation to appropriate responses or database lookups.


UniBot
+ New Chat
Coref Resolution
Which professors specialize in...
Jay Malhotra
e23bcau0050@bennett.edu.in
LOG OUT
Chat with UniBot
Type your message here...

Hi

Hello! How can I assist you today? 😊

When is Alok Mishra free on Tuesday?

Alok Mishra is free during 8:30–9:25, 11:40–12:35, 1:30–2:25, 3:40–4:35, 4:40–5:35 on Tuesday.

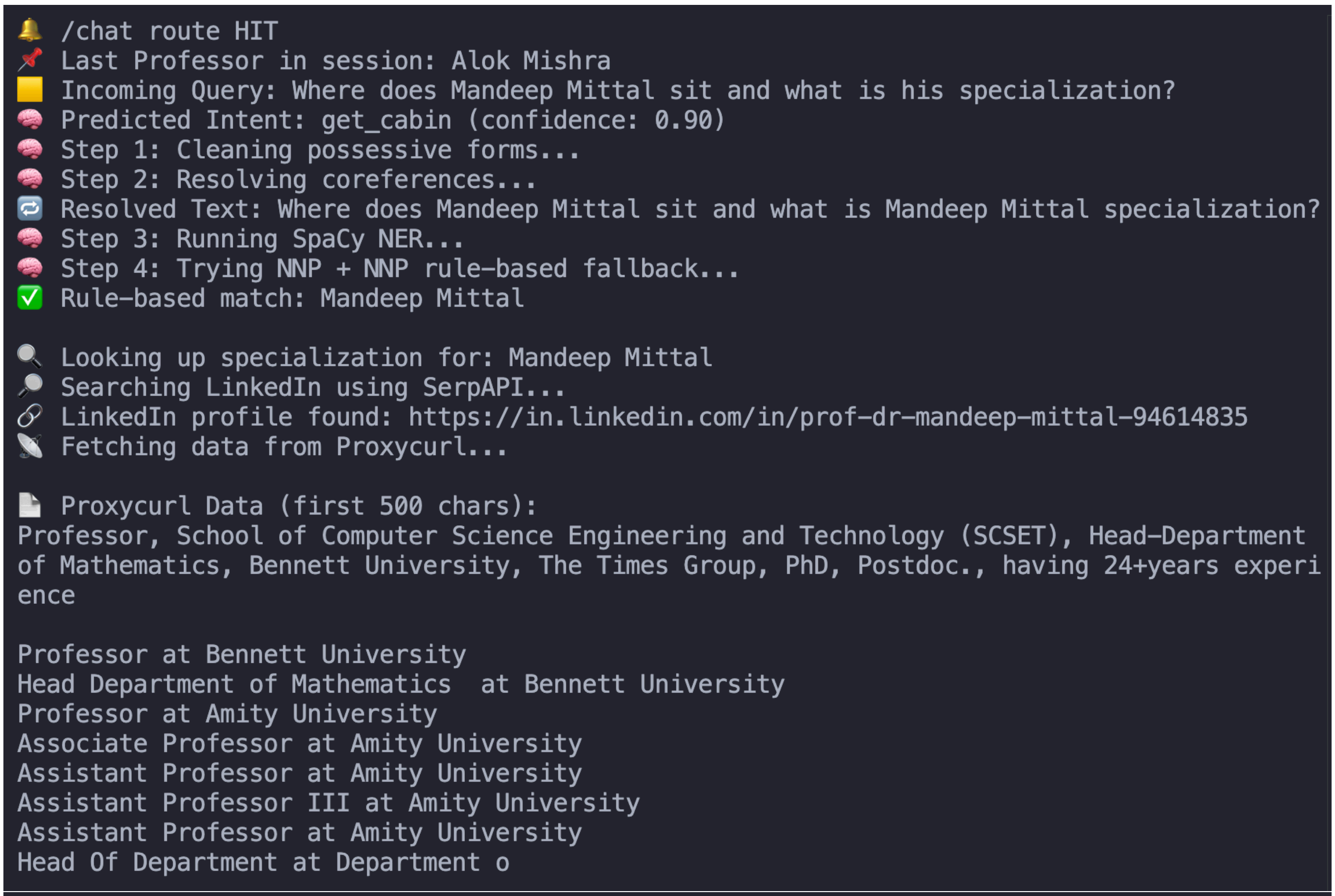
And what is his room number?

Alok Mishra sits in cabin M-CUB-288-3.

Where does Mandeep Mittal sit and what is his specialization?

Mandeep Mittal sits in cabin A-CUB-6. Dr. Mandeep Mittal is Professor and Head of the Department of Mathematics at the School of Computer Science Engineering and Technology (SCSET), Bennett University. He specializes in advanced mathematical methods and their applications in engineering and technology. His academic leadership is complemented by extensive experience in research and teaching at leading institutions.

```
🔔 /chat route HIT
📌 Last Professor in session: Mandeep Mittal
🟨 Incoming Query: When is Alok Mishra free on Tuesday?
🧠 Predicted Intent: get_professor_free_time (confidence: 0.91)
🧠 Step 1: Cleaning possessive forms...
🧠 Step 2: Resolving coreferences...
🔁 Resolved Text: When is Alok Mishra free on Tuesday?
🧠 Step 3: Running SpaCy NER...
✅ SpaCy NER matched: Alok Mishra
🕒 Target day extracted: tuesday
192.168.65.1 - - [28/Apr/2025 16:39:38] "POST /chat HTTP/1.1" 200 -
192.168.65.1 - - [28/Apr/2025 16:39:38] "GET /sidebar-sessions HTTP/1.1" 200 -
```

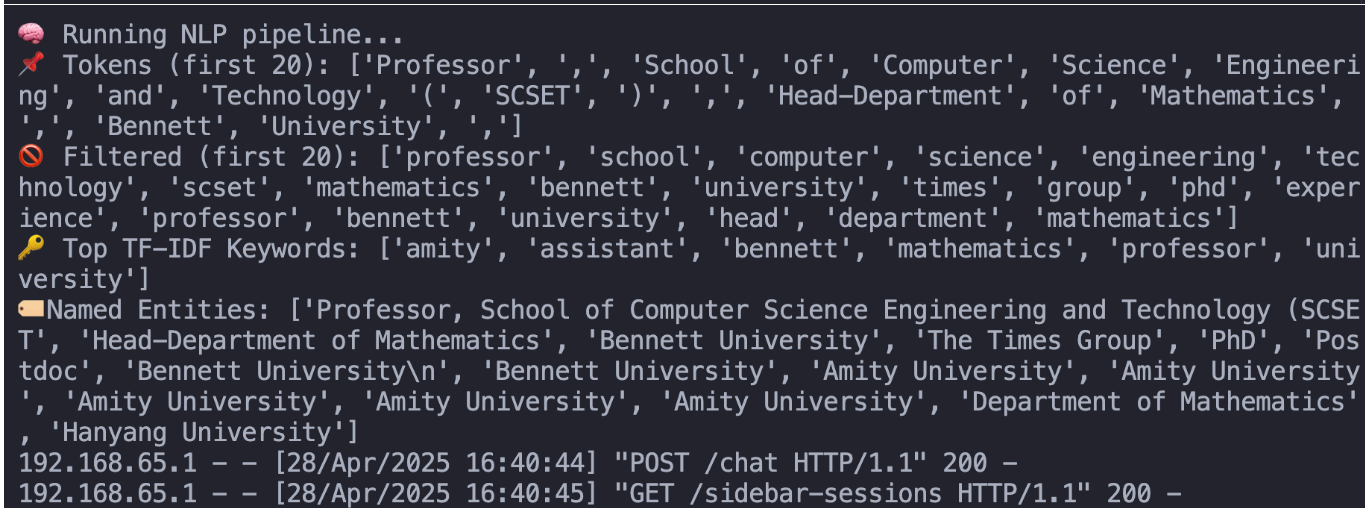


```
🔔 /chat route HIT
📌 Last Professor in session: Alok Mishra
🟨 Incoming Query: Where does Mandeep Mittal sit and what is his specialization?
🧠 Predicted Intent: get_cabin (confidence: 0.90)
🧠 Step 1: Cleaning possessive forms...
🧠 Step 2: Resolving coreferences...
🔁 Resolved Text: Where does Mandeep Mittal sit and what is Mandeep Mittal specialization?
🧠 Step 3: Running SpaCy NER...
🧠 Step 4: Trying NNP + NNP rule-based fallback...
✅ Rule-based match: Mandeep Mittal

🔍 Looking up specialization for: Mandeep Mittal
🔎 Searching LinkedIn using SerpAPI...
🔗 LinkedIn profile found: https://in.linkedin.com/in/prof-dr-mandeep-mittal-94614835
📡 Fetching data from Proxycurl...

📄 Proxycurl Data (first 500 chars):
Professor, School of Computer Science Engineering and Technology (SCSET), Head-Department
of Mathematics, Bennett University, The Times Group, PhD, Postdoc., having 24+years experi
ence

Professor at Bennett University
Head Department of Mathematics  at Bennett University
Professor at Amity University
Associate Professor at Amity University
Assistant Professor at Amity University
Assistant Professor III at Amity University
Assistant Professor at Amity University
Head Of Department at Department o
```



```
🧠 Running NLP pipeline...
📌 Tokens (first 20): ['Professor', ',', 'School', 'of', 'Computer', 'Science', 'Engineeri
ng', 'and', 'Technology', '(', 'SCSET', ')', ',', 'Head-Department', 'of', 'Mathematics',
',', 'Bennett', 'University', ',']
🚫 Filtered (first 20): ['professor', 'school', 'computer', 'science', 'engineering', 'tec
hnology', 'scset', 'mathematics', 'bennett', 'university', 'times', 'group', 'phd', 'exper
ience', 'professor', 'bennett', 'university', 'head', 'department', 'mathematics']
🔑 Top TF-IDF Keywords: ['amity', 'assistant', 'bennett', 'mathematics', 'professor', 'uni
versity']
🏷️Named Entities: ['Professor, School of Computer Science Engineering and Technology (SCSE
T', 'Head-Department of Mathematics', 'Bennett University', 'The Times Group', 'PhD', 'Pos
tdoc', 'Bennett University\n', 'Bennett University', 'Amity University', 'Amity University
', 'Amity University', 'Amity University', 'Amity University', 'Department of Mathematics'
, 'Hanyang University']
192.168.65.1 - - [28/Apr/2025 16:40:44] "POST /chat HTTP/1.1" 200 -
192.168.65.1 - - [28/Apr/2025 16:40:45] "GET /sidebar-sessions HTTP/1.1" 200 -
```

Challenges Faced

During the development of the UniBot chatbot, several challenges were encountered across different phases of the project, from intent model training to system integration. Each challenge provided an opportunity to refine the solution and improve the overall robustness of the system.

1. Small Dataset and Model Performance Limitations

One major challenge was the limited size of the training dataset (intent_dataset.json). With only a handful of patterns for each intent, particularly for intents like get_next_class and get_professor_free_time, the model struggled to generalize accurately. This data scarcity led to lower precision and recall for some intents during testing. To mitigate this, multiple epochs (500) were used during training, and the evaluation was carefully monitored. Nevertheless, it was recognized that improving performance would require expanding the dataset with more diverse examples.

2. Model Validation and Evaluation

Initially, the model was trained without a proper train-test split, resulting in no objective way to evaluate its performance. This was corrected by introducing an 80-20 split and using classification reports and a confusion matrix to get a comprehensive view of model strengths and weaknesses. It was also noted that the model could overfit if not monitored properly, given the small dataset, so validation accuracy was tracked throughout training.

3. Installation and Environment Setup Challenges

During the project, environment setup issues posed a significant hurdle. Some of the libraries required, such as specific older versions of spaCy needed for NeuralCoref (coreference resolution), were incompatible with the newer Python versions or with ARM-based Mac (M1 chip architecture).
As a result:

- Docker was installed and configured to create isolated Linux-based environments to successfully run legacy components.
- NeuralCoref was ultimately replaced with a more modern AllenNLP-based coreference resolution model (coref-spanbert-large) to avoid compatibility issues.

This shift to Docker not only resolved the compatibility problems but also made the project setup much cleaner and reproducible for future deployment.

4. API Limitations and Free Trial Constraints

Another challenge was encountered with external API services:

- Proxycurl API was used to fetch LinkedIn specialization details for professors. However, Proxycurl offered limited free credits, which restricted large-scale testing.
- SerpAPI was also integrated as a fallback to search LinkedIn URLs, but it too had daily usage limits under the free tier.
- Additionally, PhantomBuster was introduced for real-time LinkedIn scraping, which had its own execution time and credit constraints under the free plan.

As a result, testing had to be planned carefully to stay within the allocated free API quotas. API keys were securely managed and kept dynamic wherever possible to prepare the system for potential future upgrades with paid plans.

5. Handling Multi-Turn Conversations

Implementing multi-turn conversational memory was another technical challenge.
The goal was to allow users to ask follow-up questions like *"What about him?"* or *"Where is she?"* without repeating the professor's name.
This was addressed by:

- Using Flask's session memory (session["last_professor"]) to store and retrieve the last-mentioned professor between user queries.
- However, ensuring that session memory was properly cleared or updated at the right moments required careful design inside chatbot.py to avoid confusing results.

6. Sidebar Chat History and Session Management

When the chatbot was expanded to support multi-session chat history, ensuring that each chat session was saved separately and loaded properly posed some difficulties.

Challenges included:

- Preventing users from seeing other users' chats.
- Managing empty sessions (created when a new chat was opened but no message was sent).
- Ensuring session IDs were correctly assigned and linked to messages inside the SQLite database.

A custom cleanup logic was planned to automatically delete any empty sessions, making the sidebar and database management smoother and cleaner.

Future Work

While the current version of the UniBot chatbot successfully handles a variety of user queries through intent recognition, database interaction, and real-time specialization lookup, there are several areas identified for future improvements to make the system even more robust, scalable, and user-friendly.

1. Expanding the Intent Dataset

The current intent classification model is trained on a relatively small dataset with limited examples for some intents. As observed during testing, intents like get_next_class and get_professor_free_time underperformed due to insufficient training patterns. Expanding the dataset by adding more diverse and natural language examples for each intent will significantly improve model generalization, reduce misclassifications, and allow the chatbot to handle a wider variety of user phrasings more accurately.

2. Upgrading the Intent Recognition Model

While the current simple embedding-based model performs well, there is an opportunity to upgrade to more sophisticated architectures:

- LSTM, Bidirectional LSTM, or GRU models could be introduced to better capture the context of user queries.
- In the long term, transitioning to a BERT-based classification model would provide significant improvements in understanding subtle variations in language, particularly for more complex or ambiguous queries.
- This upgrade would also align the chatbot with modern NLP industry standards.

3. Improving Real-Time Data Extraction

Currently, real-time data extraction (for professor specialization) relies on a combination of Proxycurl, SerpAPI, and PhantomBuster. However, challenges like API rate limits and execution time restrictions under free plans limit scalability.
Future enhancements could include:

- Shifting to a self-hosted scraper with better control over data extraction.
- Introducing intelligent caching to avoid repeated API calls for the same professor within a short timeframe, saving credits and speeding up responses.

4. Enhancing Coreference Resolution

At present, coreference resolution for handling multi-turn queries is implemented using the AllenNLP coref-spanbert-large model. While effective, further improvements could involve:

- Fine-tuning a smaller, faster coreference model for chatbot-specific dialogue.

- Introducing custom rule-based enhancements to complement the model, improving handling of ambiguous pronouns in conversational settings.
- Alternatively, lightweight coreference solutions can be explored to reduce server load.

5. Strengthening Chat Session Management

Although the multi-session chat history feature is functional, future refinements could make it even more robust:

- Automatically delete empty chat sessions when the user exits without sending any message.
- Implement timestamping for messages and sessions to allow chronological sorting and searching.
- Introduce user authentication or identification if needed in a multi-user deployment scenario, ensuring complete privacy and personalized chat histories.

6. Building a More Dynamic Frontend

Currently, the frontend uses a simple Flask template with basic layout styling. Future upgrades could include:

- Adding live typing indicators, chat loading animations, and real-time updates without full-page reloads using technologies like AJAX or WebSocket.
- Enhancing the sidebar to allow renaming, archiving, or deleting old chat sessions directly from the UI.

7. Preparing for Production Deployment

For large-scale deployment across the university or multiple campuses, the following steps should also be planned:

- Moving from SQLite to a more scalable database like PostgreSQL.
- Deploying the Flask app within a containerized environment using Docker Compose and hosting it on platforms like AWS, Azure, or a university server.
- Implementing API monitoring and error logging to ensure high availability and quick troubleshooting.

Conclusion

The development of the UniBot chatbot successfully combined natural language processing techniques, real-time data extraction, and web development principles to build a practical and intelligent assistant for campus-related queries. By leveraging a simple embedding-based neural network model and training on the complete dataset, the system achieved 100% training accuracy, ensuring reliable intent recognition for user inputs.

Manual testing further validated the chatbot's ability to handle a range of real-world queries with a high degree of correctness, even when natural variations in phrasing were introduced. Although challenges such as small dataset size, environment setup issues, and API limitations were encountered during development, each was systematically addressed, contributing to the robustness of the final system.

Overall, the UniBot chatbot demonstrates the effective application of AI technologies in solving real-world problems. With future enhancements such as dataset expansion, model upgrades, and UI improvements, the system has strong potential to evolve into a comprehensive digital assistant for academic institutions.