**Introduction**

This project is a desktop-based AI chatbot application named **“Ask AU – AI Chatbot”**. It is designed specifically to help students and visitors of **Athabasca University** by providing quick and accurate answers to frequently asked questions. The chatbot answers general questions about the university, including course information, admission procedures, registration deadlines, exam policies, and technical support.

The chatbot works by first trying to match the user's question with a list of common FAQs stored in a local JSON file. If there is no close match, the system sends the question to the **OpenAI GPT API** and returns an intelligent response. The goal of the chatbot is to improve access to university information and reduce the time users spend searching the website.

This project is suitable for Athabasca University students, staff, or any user who prefers a simple, easy-to-use desktop application to get instant answers. The chatbot is also helpful for students who are new to the university and want quick guidance without waiting for email or phone replies.

The chatbot was developed using the Python programming language. It uses several important libraries, such as tkinter for the graphical user interface (GUI), ttkbootstrap for modern styling, and openai to connect to the GPT-4 API. The chatbot can be used in both offline and online modes, as it first checks the local data before going to the internet.

This report explains how the chatbot was planned, built, tested, and evaluated. It covers every part of the development process, from the problem being solved to the final user experience. Screenshots of the actual chatbot in use are also included to support each section.

By building this chatbot, I learned how to design user-friendly applications, how to combine local and cloud-based intelligence, and how to test and improve a real software system. The result is a working chatbot that is practical, smart, and helpful to users.

**Problem Statement**

Universities often receive a high number of common and repeated questions from students, such as “How can I apply?”, “What is the contact information?”, or “Can you tell me about a specific course?” These questions are usually answered by email, phone calls, or through website navigation, which can be time-consuming and sometimes frustrating for users—especially if they cannot quickly find what they need.

At Athabasca University, where many students study remotely, getting support or information fast is very important. Students may be in different time zones or may not always have access to staff. The university’s website does have helpful information, but it is often spread across different pages and sections. This can make it difficult for new students to find the answers they need, especially when they are under time pressure.

Many students just want quick, short, and clear answers to basic questions. If they don’t find the answer fast, they might become discouraged or confused. Waiting for support through email or phone calls can take hours or days. This delay affects the user experience and may result in students feeling unsupported.

Additionally, universities spend a lot of time and resources answering the same simple questions again and again. Automating this process with a chatbot can save time for both students and staff.

This project solves that problem by creating a desktop chatbot application that can answer most common questions instantly. The chatbot is simple, fast, and available anytime. It improves access to information and gives users a better experience without needing to contact someone manually. Also, it helps reduce the workload on university staff who answer student questions.

The chatbot also supports both offline and online modes. If it finds an answer locally, it shows it right away. If not, it connects to the OpenAI GPT API and generates a smart, accurate response. This makes the chatbot more flexible and intelligent, helping users even when their questions are not in the FAQ list.

**Project Objectives**

The goal of this project was to design and implement a simple, user-friendly chatbot that could assist students and users of **Athabasca University** by answering their common questions quickly and accurately. This chatbot was created as a **desktop application** so it could be used without needing a browser or mobile device, offering a direct and focused user experience.

The main objective was to improve access to important university information, such as contact details, how to apply, course descriptions, and university policies. The chatbot should be able to provide instant answers to common questions without the need for users to search the university website manually or wait for email replies.

The following were the key objectives of the project:

**1. Design an Easy-to-Use Chat Interface**

One of the first goals was to build a clean and simple chat interface that looks professional and is easy to understand. The interface needed to include a text area to show the conversation, an input field to write questions, and buttons to send messages or select quick questions. This would help all types of users, including those with limited technical knowledge, to interact with the chatbot easily.

**2. Provide Answers to Frequently Asked Questions**

The chatbot needed to have access to a local JSON file that contains a list of frequently asked questions (FAQs) and their answers. The system should be able to search this data and return the most relevant answer to the user’s question using fuzzy matching, even if the question was not typed perfectly.

**3. Use AI to Answer Complex or Unlisted Questions**

If a user asks a question that is not in the local data, the chatbot should use the OpenAI GPT API to get a smart answer. This allows the chatbot to provide answers for a wide range of topics, even those not planned in advance. This also helps the system grow over time and handle more types of questions.

**4. Allow Quick Access to Common Questions**

To save time, the chatbot should provide a list of quick question buttons such as:

* “Contact Info for Athabasca University”
* “How can I apply to Athabasca University?”
* “Tell me about Athabasca University”

When users click on these buttons, the chatbot should display an answer immediately, without needing the user to type anything. This saves time and improves the user experience. In particular, the button “Tell me about Athabasca University” should provide a general overview of the university, including important topics such as exam booking, course fees, deadlines, and other essential information relevant to all students. This helps summarise the university’s policies in one convenient answer.

**5. Include a “Clear Chat” Feature with Confirmation**

Another goal was to include a "Clear Chat" button that lets users delete the entire conversation history. However, to prevent users from accidentally deleting the chat, the system should ask for confirmation with a popup message before clearing everything. This adds a layer of safety and improves usability.

**6. Use Simple and Reliable Technologies**

The chatbot should be built using technologies that are easy to install, understand, and maintain. Python was chosen as the main language, and tkinter was used for the GUI because it is built into Python and does not require additional setup. Libraries like ttkbootstrap, dotenv, and rapidfuzz were used to add modern design, secure API access, and intelligent question matching.

**7. Test the System with Real User Input**

Finally, the chatbot should be tested by entering different types of questions, including spelling mistakes and unusual formats. The goal was to make sure the chatbot could still understand the questions and give helpful answers. Testing was also done to check the speed and accuracy of both local and AI responses.

In summary, the main goal of this project was to create a working AI-powered chatbot that could serve as a helpful assistant to students and users of Athabasca University. The chatbot needed to be reliable, easy to use, fast in responding, and flexible enough to handle both known and unknown questions. These objectives were carefully considered and guided the design and implementation of the full system.

**Tools and Technologies**

This project was developed using several reliable and popular tools, programming libraries, and technologies. Each tool was carefully chosen to support a specific part of the project, such as building the user interface, processing user input, connecting to the internet, and designing a modern, professional appearance for the application. The aim was to select tools that are easy to learn, widely supported, and suitable for a Python-based desktop application.

Visual Studio Code was used as the primary code editor throughout the development of this project. It provided an efficient and user-friendly environment for writing, testing, and debugging the Python code, with useful features like syntax highlighting, code completion, and integrated terminal support.

Below is a list of the main tools and technologies used in the development of the chatbot, along with a simple explanation of how each one contributed to the project:

**1. Python Programming Language**

Python was used as the main programming language for this project. It was selected because it is easy to read and write, and it has a large number of libraries that are useful for building applications. Python is also very popular in both academic and professional settings, which makes it a good choice for students and developers. The chatbot logic, user interface, file handling, API requests, and message processing were all written using Python.

**2. Tkinter – Graphical User Interface (GUI)**

tkinter is Python’s built-in library for creating desktop applications with graphical user interfaces. It was used to build the main window of the chatbot, including the chat display area, input box, send button, and quick question buttons. Tkinter is simple to use and does not require extra installation, which makes it perfect for student projects and small desktop tools. The chatbot window is created using tkinter widgets such as Frame, Label, Entry, and ScrolledText.

**3. ttkbootstrap – Styling the Interface**

While tkinter works well for building interfaces, its default appearance can look old-fashioned. To improve the look and feel of the chatbot, the project used a library called **ttkbootstrap**. This library brings modern styles and colors to tkinter applications by using Bootstrap-like themes. It helped make the buttons, labels, and entry boxes look clean and professional without writing complex CSS or HTML. The use of the “flatly” theme gave the application a calm and professional blue color scheme.

**4. OpenAI API – Artificial Intelligence Answers**

When the chatbot cannot find an answer in the local FAQ data, it sends the user’s question to the **OpenAI API** to get a response from the GPT model. This feature allows the chatbot to give smart and natural responses even to questions that are not pre-written. The GPT model used is gpt-4o, which is capable of understanding and answering questions in a way that feels human. The API key is stored securely and loaded using environment variables to keep it private.

**5. python-dotenv – Secure API Key Management**

To protect the OpenAI API key, the project uses a library called **python-dotenv**. This library lets the developer store private data, like API keys, in a .env file that is not shared publicly. When the chatbot runs, it reads the key from the environment file safely. This keeps the API key secure and separate from the main code, which is an important practice in modern software development.

**6. rapidfuzz – Fuzzy String Matching**

The chatbot uses rapidfuzz to match the user's question to the closest question in the local JSON FAQ file. If the question typed by the user is slightly different, or has spelling mistakes, rapidfuzz still finds the closest match based on similarity score. This makes the chatbot more flexible and helpful, especially for users who make typing errors. The extractOne function returns the best match along with a score, and if the score is above 80%, the chatbot returns the local answer.

**7. JSON – Storing FAQ Data**

The project uses a **JSON file** to store frequently asked questions and their answers. JSON (JavaScript Object Notation) is a simple and easy-to-read format used for storing structured data. It allows the chatbot to quickly look up answers based on the user’s question. The JSON data is loaded at the start of the program and flattened to make it easier to search.

**8. threading – Background Tasks**

The chatbot uses the threading module to run tasks like typing simulation and GPT response fetching in the background. This allows the chatbot to show “Typing…” while preparing a response without freezing the user interface. This improves user experience by making the chatbot feel more interactive and smoother.

**9. webbrowser – Opening URLs**

When the chatbot provides a clickable link, the project uses Python’s built-in webbrowser module to open the link in the default browser. This allows users to visit official university pages easily by clicking links directly in the chat window.

**10. messagebox – Confirmation Dialogs**

The message box module from tkinter is used to show a confirmation popup when the user clicks the “Clear Chat” button. This prevents accidental chat deletion and makes the system more user-friendly and safer.

Together, these tools and technologies allowed for the creation of a complete, well-functioning chatbot application that is simple, smart, and secure. By using only Python-based libraries, the entire project remains easy to run on most systems without the need for complex setup. Each tool played an important role in the successful development of the chatbot.

**System Design and Architecture**

Before starting the implementation of the chatbot, the system was carefully planned and designed to ensure that all the required features could be supported. A clear design helps make the development process easier and avoids confusion during coding. This section explains how the chatbot system works, how it was structured, and how different components interact with each other to produce the final results.

The chatbot was designed to follow a modular architecture, meaning that each part of the system is separated into smaller sections that handle specific tasks. This makes the program easier to test, understand, and maintain. The system consists of several main components: the user interface (GUI), the local data search module, the AI (OpenAI) integration module, and the chat logic controller.

**Development Environment**

The code for the chatbot was written and tested using **Visual Studio Code (VS Code)**. This is a free and powerful code editor that supports Python and many other programming languages. VS Code was chosen because it provides useful features like syntax highlighting, debugging tools, extensions, and an integrated terminal, which made the development process faster and more organized. It also made it easier to test the application and manage Python packages.

**Input–Processing–Response Flow**

The core of the chatbot design is based on a simple **input–process–response** model.

1. **Input:** The user types a question into the input box or selects a quick question button.
2. **Processing:** The chatbot first checks the local JSON data for a close match using fuzzy matching. If found, it returns the local answer. If no good match is found, the system sends the question to OpenAI GPT for a smart response.
3. **Response:** The chatbot formats the answer, checks for URLs, and displays the message in the chat window. If any links are present, they are made clickable.

**Local and Online Response System**

The chatbot uses a two-layer system to provide answers:

* **Layer 1 – Local Data Matching:**  
  The chatbot uses a local JSON file that contains common questions and answers. It searches this file using the RapidFuzz fuzzy matching algorithm. This step is fast and does not require internet access.
* **Layer 2 – OpenAI GPT API:**  
  If the chatbot cannot find a strong match locally, it sends the user’s message to the GPT-4 model using the OpenAI API. This provides a fallback for less common or more complex questions.

This layered approach ensures fast response time and flexibility while saving on API usage when local answers are available.

**Chat Logic and Flow Control**

The main function that controls the flow of the conversation is called send\_message(). This function collects the user input, checks the local data using fuzzy matching, and displays the result. If no answer is found, it uses threading to call ask\_gpt() in the background to avoid freezing the interface. Once the response is ready, the message is added to the chat window, replacing the “Typing…” placeholder.

**Security and Data Management**

To manage sensitive data like the OpenAI API key, the system uses the .env file. This file is not shared publicly and is used with the python-dotenv library to load the API key securely during runtime. This protects the key from being exposed in the codebase.

**User Interface Behavior**

The chat window is scrollable and displays both user and chatbot messages in different colors for easy reading. It also supports special styling for clickable links. Messages typed by users appear in blue bold text, while chatbot responses appear in dark gray. The system also includes a “Clear Chat” button that triggers a confirmation popup using messagebox.askyesno() before deleting the chat history.

**Architecture Diagram (Text-Based):** Here is a simplified view of how the system components interact:

[User Input]

↓

[Chat Controller] → [Local JSON Search (RapidFuzz)] → If match found → [Display Local Answer]

↓

If no match found

↓

[OpenAI GPT API Request]

↓

[Display AI Response]

↓

[Format Links, Update Chat Window]

The chatbot system is built in a modular and layered way, allowing it to function efficiently and be easily improved in the future. The combination of local data and AI support ensures accurate and flexible responses. The use of Visual Studio Code as the development tool also made it easier to write, test, and manage the code effectively.

**Interface Design**

The design of the user interface (UI) is one of the most important parts of this chatbot application. Since the chatbot is meant to be used by students, visitors, and possibly staff members of Athabasca University, the interface must be clear, professional, and easy to use for people of all technical skill levels. A good interface encourages users to ask questions confidently, and it also helps them navigate the system without confusion or errors.

The chatbot interface was built using the tkinter library, which is the standard GUI (Graphical User Interface) library that comes with Python. To give the interface a more modern and stylish look, I also used ttkbootstrap, a special library that adds color themes and layout improvements to tkinter. These tools allowed me to design a professional desktop window with a simple layout and clean design elements.

**Layout Overview**

When the chatbot application is opened, users see a clean window titled "Ask AU – AI Chatbot" at the top. This title is placed using a Label widget and styled with a large, bold font to grab attention. The chatbot window is sized to fit most standard screens (750x800 pixels) and centered to make sure it is easy to read and interact with.

Just below the title, there is a scrollable chat window where the conversation takes place. This window shows all user and chatbot messages in a vertical order, making it feel like a real-time chat. This is implemented using the ScrolledText widget, which allows users to scroll up and down if the conversation becomes long.

The messages are styled in different colors and fonts to make them easy to understand:

* User messages appear in bold blue text to clearly show what the user asked.
* Chatbot responses appear in dark gray for readability and a calm tone.
* Clickable links appear in orange and underlined, so users can easily tell they are links.

**Input and Send Area**

At the bottom of the chatbot window, there is a horizontal input area. On the left side, users see a **text entry box** where they can type their question. This entry box is styled using ttkbootstrap's "light" theme to match the rest of the interface. On the right side of the input area, there is a **“Send” button**, which the user clicks to send their message to the chatbot.

For convenience, the **Enter key** is also bound to the send function. This means users do not have to click the button — they can simply press Enter on the keyboard after typing their message.

**Quick Access Buttons**

Below the main chat window, there is a section titled **“Quick Questions – Click to Learn Instantly.”** This section contains three buttons that users can click to instantly ask a common question:

1. “Contact Info for Athabasca University”
2. “How can I apply to Athabasca University?”
3. “Tell me about Athabasca University”

These quick-access buttons are useful for new users who may not know what to type. When clicked, they send a pre-written question to the chatbot and display the result in the chat window. This feature saves time and improves the experience, especially for students who just want basic information without typing.

**Clear Chat Button**

At the very bottom of the application, there is a red button labeled **“Clear Chat.”** This button allows users to delete the entire chat history from the window. To avoid mistakes, clicking this button opens a **confirmation popup** using the messagebox.askyesno() function. If the user confirms “Yes,” the chat is cleared. If the user chooses “No,” nothing happens, and the message is preserved.

This safety feature makes the application more user-friendly and protects against accidental clicks.

**Responsive Design**

The interface also includes **responsive behavior** to improve usability:

* The chat window automatically scrolls to the latest message after each response.
* The "Typing..." message is shown when the chatbot is preparing an answer.
* Links provided by the chatbot (e.g., to Athabasca University’s website) are **clickable** and open in the user’s default browser.

These small additions make the chatbot feel more interactive and modern, even though it is a simple desktop application.

**Screenshot Support:**

A screenshot of a chatbot

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a chatbot

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

The screenshots provided in this report serve as visual proof of the chatbot’s design, functionality, and real-time interaction with users. Each figure demonstrates a specific feature of the application, including the main interface layout, typing simulation, clickable links, course-specific answers, polite responses, and the clear chat confirmation system.

These images not only show that the chatbot was fully implemented and tested, but they also highlight its usability and responsiveness. The chatbot responds smoothly to both quick-access questions and custom typed questions, and it returns helpful, well-formatted replies. All interface elements are styled consistently, and the application behaves as expected in all tested scenarios.

Together, these screenshots support the written explanations in the report and provide visual confirmation that the system works correctly. They also show that the project goals were successfully achieved and that the chatbot offers a reliable and user-friendly experience for students and users of Athabasca University.

**Implementation**

The implementation of the chatbot involved writing and organizing code to handle all of its features, including the user interface, input processing, message responses, and communication with external services like the OpenAI API. The chatbot was fully developed using the **Python programming language** inside the **Visual Studio Code** editor. This section explains how the main parts of the chatbot were built and how they work together to provide a smooth experience for the user.

**Starting the Application**

When the Python file chatbot.py is run, the program first imports all the required libraries at the top. These include:

* tkinter and ttkbootstrap for the interface
* openai for connecting to the GPT-4 API
* json for reading FAQ data
* dotenv for secure environment variables
* threading and time for managing typing simulation
* rapidfuzz for matching user questions with local data
* webbrowser for opening clickable links
* messagebox for showing confirmation popups

After importing the libraries, the program loads the .env file using load\_dotenv() to access the API key securely.

**Loading and Preparing FAQ Data**

The chatbot uses a local file named athabasca\_faqs.json to store a list of frequently asked questions and answers. The JSON file may contain data in a nested format, so a function called flatten\_json() is used to convert it into a flat dictionary. This makes it easier to search and match the user’s question against the keys.

The function load\_faq\_data() reads the JSON file and flattens it. The flattened data is stored in memory so it can be quickly searched during each conversation.

**Matching Local Questions Using RapidFuzz**

When the user types a question, the chatbot first tries to find a local answer. This is done using the get\_best\_match() function, which uses the process.extractOne() method from the **RapidFuzz** library. This method compares the user’s question to all available questions in the local JSON data and returns the one with the highest similarity score.

If the score is 80% or higher, the system accepts it as a good match and returns the local answer to the user. This allows the chatbot to still understand questions even if they are typed with different wording or small spelling mistakes.

**Using OpenAI GPT API**

If no strong match is found locally, the chatbot calls the **OpenAI GPT API** to generate a response. The function ask\_gpt() handles this by sending the user’s question to the GPT-4 model. The response is then returned as plain text.

This AI-based answer helps the chatbot handle more complex or rare questions that were not included in the local JSON file. It also gives the chatbot the ability to answer follow-up questions or anything outside the original scope.

The openai.api\_key is set using the environment variable to keep it secure and hidden from public view.

After receiving a response — either from the local data or the GPT API — the chatbot displays it in the chat window. The ScrolledText widget is used to show the conversation. Each message is styled to help users clearly understand who said what.

* User messages are shown in **bold blue text**.
* Chatbot replies are shown in **dark gray**.
* Links are styled in **orange and underlined**, so users can tell they are clickable.

This formatting is done using the tag\_config() and tag\_bind() functions in tkinter.

**Typing Simulation**

To make the chatbot feel more realistic, a line that says **"Chatbot: Typing..."** is shown after a user sends a message. This message is only temporary. Once the real response is ready, it replaces the typing message.

For local responses, the delay is very short.  
For GPT responses, a longer delay is used to give the model time to respond. The threading module is used here so that the interface does not freeze while waiting.

**Handling Clickable Links**

When a chatbot response includes a website link (such as one to the university’s homepage), the system uses a regular expression to detect the link in the text. If a link is found, it is displayed as a clickable part of the message.

Clicking the link opens it in the user’s default web browser. This is handled using Python’s built-in webbrowser module.

**Quick Question Buttons**

Below the chat area, the chatbot has three buttons with common questions:

* **“Contact Info for Athabasca University”**
* **“How can I apply to Athabasca University?”**
* **“Tell me about Athabasca University”**

These are set up using a simple for-loop to reduce repeated code. When clicked, the button sends the question automatically, and the answer is shown just like a normal chat.

**Clear Chat Feature**

The chatbot also includes a **Clear Chat** button at the bottom. When the user clicks it, a confirmation popup appears. The chatbot uses the messagebox.askyesno() function to ask:  
**“Are you sure you want to clear the chat?”**

* If the user clicks **Yes**, the chat is cleared.
* If the user clicks **No**, the conversation remains.

This helps prevent accidental message deletion and improves the user experience.

The implementation of this chatbot included several parts working together: loading data, checking for answers, connecting to AI, and managing user input and output. The chatbot uses both offline and online data to answer questions quickly. Its simple layout, helpful features, and responsive design make it easy to use and reliable.

**Strengths and Limitations**

This section explains what the chatbot does well and also the things that could be improved in the future. Every software project has its good parts and some areas where it can get better. Understanding both helps improve the quality of the application.

**Strengths**

1. **Easy to Use**  
   The chatbot is very simple to use. Users can type their questions or click on quick question buttons. The chat window is clean and easy to read, with clear text and proper colors. Even users with little technical knowledge can use it without problems.
2. **Fast Responses**  
   When answering questions from the local data file, the chatbot responds instantly. This is helpful for users who want answers quickly without waiting. The use of fuzzy matching allows it to understand questions even with small spelling mistakes.
3. **Smart AI Answers**  
   If the chatbot cannot find an answer in the local file, it uses the OpenAI GPT API to provide an answer. These responses are smart, accurate, and often detailed. This makes the chatbot more intelligent and useful.
4. **Clickable Links**  
   When answers include links to official university pages, they are shown as clickable text. This makes it easy for users to visit websites without needing to copy and paste the link. It saves time and improves the experience.
5. **Clear Chat Confirmation**  
   The "Clear Chat" button includes a confirmation popup. This protects the user from deleting the conversation by mistake. It adds safety and shows that the chatbot was designed with care.

**Limitations**

1. **Dependence on Internet for GPT Answers**  
   If the user asks a question that is not in the local file, the chatbot must use the internet to connect to the OpenAI GPT API. If there is no internet connection, it cannot give an answer to these questions.
2. **Limited Local Knowledge**  
   The local JSON file only contains a few questions. This limits what the chatbot can answer offline. To improve this, more questions and answers could be added to the local data file.
3. **Basic Interface**  
   The user interface is clean and works well, but it is not as advanced as a modern web or mobile app. It may not support screen readers or accessibility tools. Also, it does not save chat history.
4. **No Voice Input or Output**  
   The chatbot only works with text. There is no option to speak to the chatbot or hear it speak. This could be added in future versions to help users with different needs.
5. **No User Feedback System**  
   There is no way for users to rate the answers or give feedback. A feedback feature could help improve the chatbot over time.
6. **AI Hallucinations and Trust**

Sometimes, the AI can give answers that are not true or are made up. These are called hallucinations. This is a risk when using AI models. To reduce this problem, the chatbot first checks the local FAQ file for answers before asking the AI. In the future, we could also add a way to warn the user when the answer comes from the AI or give links to double-check the answer.

**Ethical Considerations**

Using AI in chatbots brings many benefits, but there are also some important ethical concerns. One concern is that the AI (GPT) may sometimes give answers that are biased or unfair, because it was trained on a large amount of online data that may not always be balanced.

Another issue is that the AI might give wrong or made-up answers, even if it sounds confident. This is called AI hallucination. To help with this, the chatbot uses a local FAQ file first, and only sends the question to the AI if it cannot find a good answer. This helps give more accurate responses. Still, users should check important answers on the official Athabasca University website if they are unsure.

Finally, this chatbot does not save any user input. This protects user privacy, but it also means the chatbot cannot remember previous questions. In the future, we could look at adding a secure way to remember questions, if privacy can still be protected.

**Future Improvements**

This section explains what can be added or changed in the future to make the chatbot even better. Although the chatbot works well now, there are many ideas to improve it and give users an even better experience.

**1. Add More Local Questions and Answers**

Right now, the chatbot has a small number of questions in the local JSON file. In the future, this file can be expanded to include more topics. This will help the chatbot answer more questions even without internet. It will also reduce the need to connect to the OpenAI GPT API, which can help save time and costs.

**2. Save Chat History**

At the moment, the chatbot does not save past conversations. A future version could add a feature to save chat history. This would help users go back and review previous answers. It would also make the chatbot more useful for students who want to keep important information.

**3. Add a Feedback Button**

Adding a feedback feature would let users rate the chatbot’s answers. For example, a simple thumbs up/down or a short text box for suggestions. This feedback could help developers know which answers are helpful and which ones need improvement.

**4. Add Voice Support**

In the future, the chatbot could include voice input and output. This means users could speak their question and hear the answer out loud. This would help users with visual impairments or those who prefer using voice instead of typing.

**5. Make It Work on Mobile Devices**

Currently, the chatbot is a desktop application. In the future, a mobile version could be created using tools like Kivy or a web-based version using Flask. This would allow more students to use it on their phones or tablets anytime and anywhere.

**6. Add User Login (Optional)**

A future update could include a login system. This way, students could have a more personalized experience. For example, the chatbot could remember previous chats or give answers based on their courses and study program.

There are many possible ways to improve the chatbot in the future. These include adding more questions, saving chat history, using voice features, supporting mobile devices, and improving accessibility. These updates would help more users and make the chatbot more powerful and helpful.

**Conclusion**

This project was about creating a simple desktop chatbot called Ask AU – AI Chatbot to help students and visitors of Athabasca University. The goal was to make it easier for users to get quick answers to common university-related questions such as how to apply, course information, contact details, and policies.

The chatbot was built using the Python programming language along with helpful libraries like tkinter, ttkbootstrap, openai, dotenv, and rapidfuzz. The application runs on the desktop, and it has a friendly interface that allows users to type questions, click buttons, and receive clear, useful responses.

The chatbot works in two ways. First, it checks a local JSON file for a matching question and answer. If no match is found, it connects to the OpenAI GPT-4 API to get a smart and detailed answer. This makes the chatbot fast, flexible, and intelligent. Users can get answers right away, even if the question is not stored in the system.

The chatbot also includes features like:

* A confirmation popup before clearing the chat
* Clickable links in answers
* A typing animation that feels like a real conversation
* Quick question buttons for instant access

All of these features were tested, and the chatbot worked well in different situations. It was able to understand different types of questions, including ones with spelling mistakes. The answers were accurate and easy to understand.

This project helped solve a real problem. Students at Athabasca University often study online and may not have fast access to staff or support. The chatbot helps fill that gap by giving instant help 24/7. It can reduce waiting time, improve the user experience, and lower the number of repeated support requests sent to staff.

Through this project, I learned how to:

* Build a working desktop application
* Design a user interface using Python
* Connect to an AI service (OpenAI API)
* Handle both offline and online data
* Test and debug software
* Think about real users and how to make their experience better

In the future, this chatbot could be improved by adding more local questions, saving chat history, allowing voice input, or making it work on mobile phones. But even now, the chatbot is functional and helpful. It can be used by any student who wants fast answers in a simple way.

Overall, this was a successful and meaningful final-year project. It used real tools, solved a real problem, and showed how technology can make university support more helpful and available.

**References:**

Python Software Foundation. (n.d.). *tkinter — Python interface to Tcl/Tk*. Retrieved from <https://docs.python.org/3/library/tkinter.html>

ttkbootstrap. (n.d.). *Modern themes for tkinter*. Retrieved from <https://ttkbootstrap.readthedocs.io/>

Python dotenv. (n.d.). *python-dotenv: Reads key-value pairs from a .env file and can set them as environment variables*. Retrieved from <https://pypi.org/project/python-dotenv/>

Athabasca University. (n.d.). *Home*. <https://www.athabascau.ca/>

Python Software Foundation. (n.d.). *webbrowser — Convenient web-browser controller*. Retrieved from <https://docs.python.org/3/library/webbrowser.html>