UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Final Paper 2021

Title: Virtual Student Advisor

Author: Avhusaho Ramalala

Project Abbreviation: ADVICE

Supervisor(s): Aslam Safla

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	20
Theoretical Analysis	*0*	*25*	0
Experiment Design and Execution	*0*	*20*	0
System Development and Implementation	*0*	*20*	20
Results, Findings and Conclusions	*10*	*20*	10
Aim Formulation and Background Work	*10*	*15*	10
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	
Total marks		**80**	

ADVICE: Virtual Student Advisor

Avhusaho Ramalala
rmlavh0011@myuct.ac.za
University of Cape Town
South Africa

ABSTRACT

Student advisors offer a great deal of assistance to students and can sometimes get overwhelmed with managing the large number of queries from students. There is a need for a solution that would help lessen the burden on Students Advisors and assist more students at the same time. An automated system that can handle some of the simpler queries would be very useful. It would be beneficial to both Student Advisors and students as students get the help they need, and Student Advisors will have additional time to manage complex queries. A Virtual Student Advisor website would include a chatbot available to assist students round-the-clock (24/7). This paper focuses on the chatbot aspect of the system. The aim of the project is to implement a working conversational agent that assists students with answering academic queries.

CCS CONCEPTS

• **Natural Language Processing**; • **Machine Learning**; • **Pattern Matching**; • **Software Development** → iterative Development Cycle; • **Web Development**;

KEYWORDS

Chatbot, neural networks, natural language processing, text processing

1 INTRODUCTION

Navigating university can be a daunting task for any student. Student Advisors play a vital role in assisting students throughout their time at university [10]. It is an important role that, when performed efficiently and effectively, will yield success for both the students and the university. The University of Cape Town (UCT) enrolls over 25000 students each year. Student Advisors assist all of these students, directly or indirectly, over the course of the year, for example, during registration. This poses a challenge as the sheer number of queries is a great deal for them to process.

Students can get assistance from Student Advisors through email, virtual meetings, or physical meetings (when it is safe to do so). Given the sheer number of students being enrolled, it is evident that each year Student Advisors are faced with an overwhelming number of queries which have to be attended to. Due to the number of queries and sometimes complexity of queries, students do not get responses immediately and Student Advisors are overworked. This inefficiency can have a detrimental effect on a student's academic career [11]. Student advisors also may not have enough time to attend to all queries as they also have to complete their academic duties like lecturing [11].

We proposed a Virtual Student Advisor system as a solution to the problem. Such a system would offer an alternative for students, rather than email a Student Advisor, the student could simply visit the website and find the information they need. Several individuals at other universities have attempted to create such an automated system for students with varying success [13, 17, 20]. Such a website would assist students with credit calculations, pre-admission guidance, establishing prerequisites, offering summaries for courses, and explaining policies and procedures. The website would also provide summaries for information found in the UCT handbooks. Students would be able to navigate the website to find the information or use the chatbot to get answers.

This paper will be focused on the Chatbot aspect of the website. A chatbot is a software system capable of imitating a conversation with a human using natural language [2, 14, 15]. Implementing such a system includes building the chatbot, training the chatbot and developing the chatbot to provide the best answer possible to most simple student queries.

1.1 Problem statement and Aim.

Student Advisors cannot efficiently and effectively attend to queries submitted by students. The number of Student Advisors and the time they are available is not proportional to the number of students in the university and number of queries. The current form of communication, email or virtual meetings, is not sufficient for students. Depending on the availability of the Student Advisor, response times vary regardless of how simple or complex the query is. Student queries can range from simple queries like what the prerequisites for a course are to complex queries like add/drop protocols for modules.

The aim of the chatbot is to eliminate the unpredictability of response times. It should also be available to provide answers to students at all times. The chatbot would be able to answer most of the simple queries, short questions that most students ask, with short answers. The chatbot should be as accurate as possible. It would provide general information, information mostly found in handbooks and UCT's various websites. A chatbot to answer queries, available at all times, would offer relief for Student Advisors. Students get answers immediately and Student Advisors will have additional time to attend to complex queries.

The following sections have been arranged as follows: Section 2 will provide the background of chatbots in general, chatbots in the context of academic institutions and state their issues. Section 3 will provide a requirements analysis of the system and the design plan of the implemented system. Section 4 shows the development process, architecture design and how the system was implemented. Section 5 will provide the findings made from user testing, discuss the feedback and provide suggestions for future work. Conclusions of the project are provided in Section 6.

2 BACKGROUND AND RELATED WORK

Computer programs capable of communicating in natural language (chatbots) have existed for several decades. The first "chatbot" was a program named ELIZA created by Joseph Weizenbaum in 1966 [20]. The program identified key words in input sentences and generated answers through rules that where coded into the program [19]. There have been many chatbot programs created since then, Elizabeth and ALICE are some of the prominent programs in existence.

Artificial Linguistic Internal Computer Entity (ALICE) is an Artificial Intelligence chatbot capable of natural language processing (NLP). ALICE is an adaptation of ELIZA created by Dr Richard Wallace in 1995 [4]. The program uses pattern-matching algorithms to match sentence inputs to outputs [2], which makes it easier and less complicated than other NLP chatbot programs.

Elizabeth is also an adaptation of ELIZA but with multiple improvements in selection, substitution, and storage mechanisms [14]. Storage functionality meant the program could store input for further use in the conversation at a later point. Unlike ALICE, Elizabeth still used the basic concepts of pattern matching and sentence decomposition used in ELIZA, although with improvements in adaptability and flexibility [14].

2.1 Related Applications.

There is not much literature on applications that implement a chatbot in the context of an academic institution or an online student advising system.

Carolis et al [8] implemented a program that is similar to the application this project aims to achieve. Their implementation attempted to mimic human conversation together with the use of an animated character with human resemblance. Bhavika R. et al. [13] and Herry D Wijaya et al. [20] have also implemented applications similar to a student advising chatbot. Suvethan N. et al. also implemented a Virtual Student Advisor chatbot using NLP, a Schedular and a feedback analyzer [17]. A feedback analyzer processes inputs by users to the chatbot to find common topics.

Bhavika R. et al. designed a chatbot using Artificial Intelligence Markup Language (AIML) and Latent Semantic Analysis (LSA). The chatbot was created to answer Frequently Asked Questions in the University [13]. This is close to what the chatbot in this project aimed to achieve but limited mostly to a database of frequently asked questions (FAQs). The program could respond to basic queries but had difficulties understanding more complex queries.

Herry D. Wijaya et al. aimed to create a virtual assistant to be integrated to the existing website of the Mercu Buana University, Jakarta, Indonesia. The virtual assistant would aid in answering questions and thus decreasing student traffic. The previous system at the university had challenges in assisting students with queries. The virtual assistant was created with inspiration from ELIZA [20]. They concluded their chatbot was successful although more research would have been needed. A drawback of the project was the little research done for the design. More research and testing could have greatly improved the design and implementation of the program.

Suvethan et al. created a mobile and web application students can interact with. The application used NLP and pattern-matching to manage students queries and provide appropriate answers. The system could assist students automatically or give students the option to meet with a human Student Advisor [17].

Jack Cahn of the University of Pennsylvania presented detailed literature on how each step of the design process can be executed [5]. One can choose between using Bayesian models or non-Bayesian models to process user input, each with their advantages and disadvantages. Bayesian models are mainly based on the probability of the input sequence given the language base- language used to train the chatbot. Non-Bayesian models apply neural networks, text classification using machine learning, perceptrons and decision trees. Machine learning is more suited for a chatbot as it provides context based on past instances and generates better communication. The downside of this being that machine-learning based algorithms have increased complexity and would take longer to implement.

2.2 Issues/Difficulties with current software.

2.2.1 Not Designed for South African Universities. Previous implementations of chatbots have mostly been only capable of general conversation. Current implementations in an academic setting have been built specifically for their respective universities, e.g., Herry D. Wijaya et al. implemented a chatbot specifically for Mercu Buana University [20]. Existing chatbot libraries like Python's ChatterBot (ChatterBot) are designed for general "small-talk" conversation and therefore have to be specialized for one's needs [7]. In this case, the chatbot's training data would have to be altered for the University of Cape Town. There seems to be no student advising chatbot that has been built for a South African university in record. It is possible, however, that a chatbot has been done before but not officially documented.

2.2.2 Language. The University of Cape Town enrolls a diverse pool of students in terms of race, culture, ethnicity, etc. The chatbot should be implemented with this in mind. It should utilize a common language that most, if not all, students understand. The main language of communication at the university is English. Other chatbot programs have been implemented for different languages like Kuan-Hua, Chinese standard speech known as Mandarin [6]. It is worth considering that the English language has several versions, United Kingdom, South African, Australian, etc., [18]. Training data for this chatbot will have to be edited to the South African English dialect [12].

2.2.3 Confidentiality and Data privacy. Current literature does not offer much detail on what security measures, if any, were put in place to ensure that user's data is protected. This is vital because the chatbot is placed in an online platform and thus could be vulnerable to malicious software online. Previous chats must be securely stored in the database, in encrypted format if possible.

2.2.4 Accessibility. Students from low-income backgrounds should also be able to access the website. Current implementations do not state how the applications were built for optimum data usage to cater for various types of users. Accessibility also refers to who will be able to use the chatbot on the website. High school students (prospective students) are potential users. This potentially brings with it several ethical and legal challenges due to the storage of data belonging to minor person(s). Related applications do not address

this issue. Storing information of high-school students or persons not registered to the university, whilst still abiding by the POPI Act [1], is a challenge. A solution to this would be restricting storage of previous chats to users that are registered to UCT only. Other users can still use the chatbot, however, no information will be stored.

2.2.5 Ease of Use and UI Design. An important aspect of any user-centred system is how easy it is to use. There is significant literature on designing a user interface for a chatbot. For example, S. Shyam Sundar and Eun Go provide literature that gives insight into the advantages and disadvantages of humanizing a chatbot [9]. This project mainly focuses on the back-end on the chatbot, that is, producing responses in text format rather than displaying. However, humanizing a chatbot is not only limited to its appearance, but it also refers to the types of responses, length of responses and time taken to reply. Making a chatbot more human brings with it more challenges as it means the responses will have to be edited to be more "human-like".

3 REQUIREMENTS ANALYSIS AND DESIGN

3.1 Requirements Analysis

3.1.1 Functional Requirements. Main functionalities of the chatbot are providing answers to simple short queries, providing short explanations about courses, providing contact details for relevant staff or departments, providing links to websites, and storing previous chats.

- The chatbot should be capable of various kinds of greetings and goodbyes. One of the aims of the chatbot is to be human-like, to a point. It should be able to respond to informal greetings and goodbyes. Each course has a short description in the handbooks. The chatbot should provide a short summary of all these descriptions. Messages should be short and concise. This information will be manually extracted from handbooks and inserted into the training data for the chatbot.
- Sometimes students simply want contact details of a staff member or department. The chatbot should be able to provide those contact details, if not then the chatbot should be able to provide information on where the information can be found. Such information changes frequently as different staff members may have been appointed to different roles with each year, so it should be possible to change this information with ease. Providing information of where to find the information may be a safer option as it is less likely that the location of where the information is placed changes. This is an area of improvement for future iterations.
- Each registered student will have a history of their chats stored on the database. Chats are stored in an array of strings as one of the attributes of the Users relation. Chats are stored prefixed with "usr_" or "bot_" to differentiate between input by the user and chatbot responses. These chats should be secured to ensure data privacy.

3.1.2 Non-Functional Requirements.

- **Quick response time** . Users should get replies immediately, they should not have to wait, as they must with emails. Every response should take no more than 3 seconds. However, several factors should be considered, response time could be affected by speed of the internet or network coverage as well.
- **Short responses** . Users should not feel discouraged to read responses sent back by the chatbot. Ensuring that the responses are of reasonable length is imperative for how often users engage with the bot. if a query requires a long response such as a whole page in a handbook then the user will be given the link to the UCT website where the information is stored or be referred to the handbook. The chatbot is meant for short simple queries.
- **Capacity** . The number of previous chats stored is dependent upon the size of the server as all the chats are stored in the database. No information is stored locally (in user's device). Currently all chats are in text format therefore they occupy relatively little space. It is not expected that each user will be conversing with the chatbot frequently, therefore there will be enough space for chats sent within the students' time at the university.
- **Grammatical correctness** . Text input by the user does not have to be spelled correctly or grammatically correct. The input also does not have to match sentences in the training data exactly in order for responses to be correct. The chatbot will use pattern matching (further described in the following section), so if the query has the expected key words then a response will be generated.
- **Reliability, maintainability, and availability** . The bot will be available for users at all times ("24/7"). It may not be available when the website is undergoing maintenance or data is being updated. Maintaining the chatbot is a challenge because one must ensure that the dates and information the system uses are accurate and relevant to the current year.

3.2 Design Plan

3.2.1 Architecture. The system follows a Model-View-Controller architecture. The class diagram in figure 1 shows the objects that make up the chatbot system, back-end. The actual view of the system is what is displayed on the website UI, in this context, the view is the object that is responsible for accepting user input from the front end and returning responses, Chatbot. The Chatbot object is the Controller (in context of the website), it is only responsible for accepting texts and returning the response generated by the model to the front end.

The Bot, PreProcessor and Trainer objects make up the Model aspect of the system. information is passed to the Bot, which then uses the trained neural network model to generate a response. PreProcessor cleans user input off punctuation marks or any acronyms or derived words. The Trainer object loads the trained neural network, processes the cleaned input, and provides a response that is then sent to the Chatbot object (Controller) to be sent to the view. DatabaseManager is responsible for any queries made to the database, either to save chats or get information.

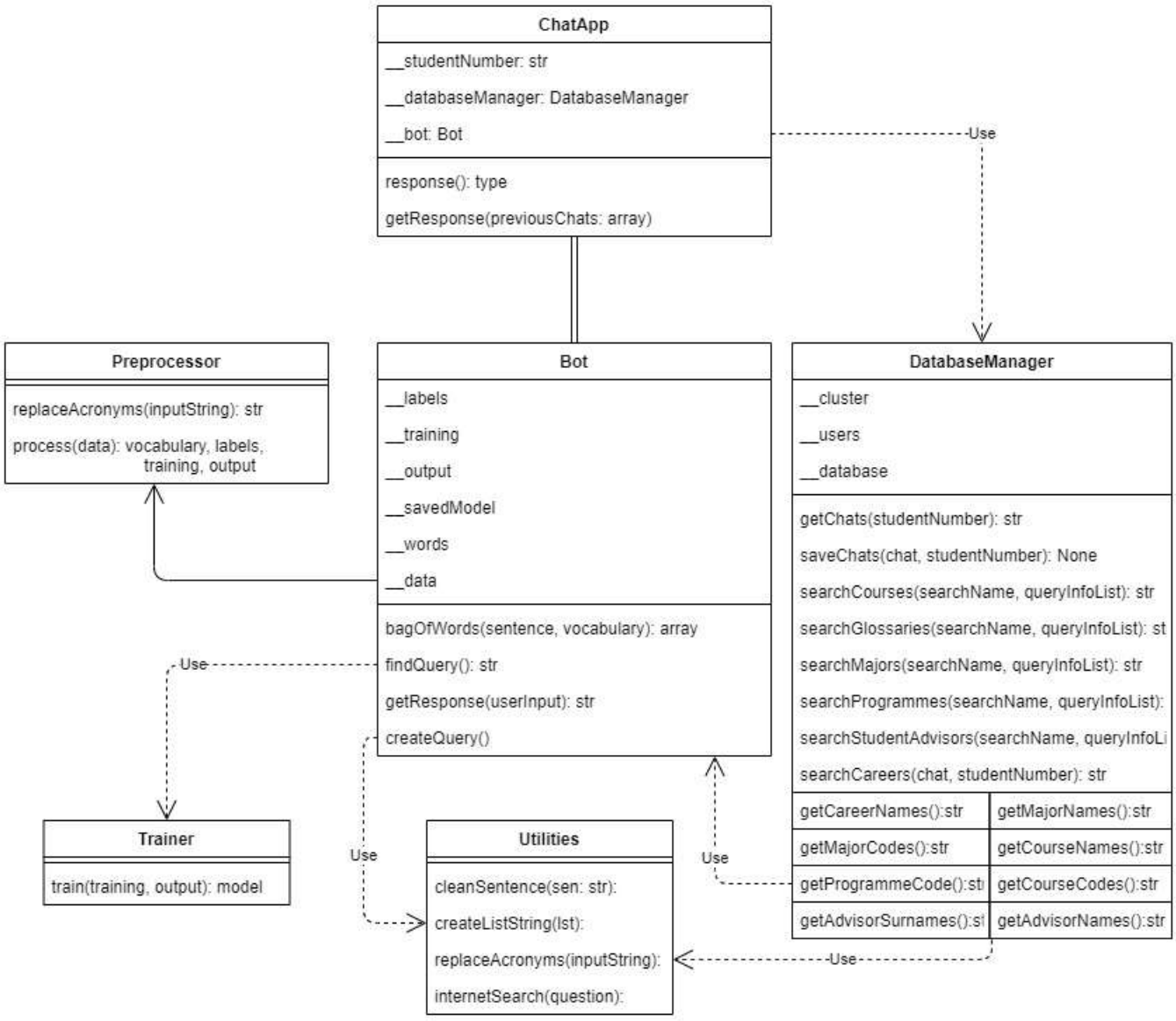


Figure 1: Class Diagram presenting architecture of the chatbot.

The user is abstracted from the logic of the system. All information required by the view can be acquired through the controller. Each object has one responsibility. Trainer is only responsible for creating or loading the neural network for language processing. PreProcessor is only responsible for processing user input before it is passed to the neural network. Using a layered architecture makes the system more maintainable as changing one layer (i.e. View, Model or Controller) has less consequence over the whole system. For example, the view can be changed with little to no consequence to the model.

3.2.2 Iterative Development Cycle. The system was developed through an iterative process. It is an evolutionary system which evolved with each iteration. An iterative, agile process was more favorable to accommodate for changing requirements during the development process. Developing the system iteratively also makes testing easier, the system is tested after each feature is implemented to ensure that correctness is maintained.

An evolutionary prototype was created for the first demonstration. The demonstration had positive feedback. An evolutionary prototype was preferred to a throw away prototype because it would be easier to refine a working system based on the feedback rather than starting from scratch again [3].

There are six steps in the iterative development cycle. Planning, Implementation and Testing are repeated with each iteration and Initial Planning, Deployment, Evaluation are only done at the beginning and end respectively [16].

- **Requirements gathering/Initial Planning** : The initial phase entailed gathering requirements from the supervisor

(client) through Microsoft Team Meetings. A virtual student advisor is a user-centred system therefore interviews were held with stake holders, i.e., Student Advisors. This helped with the initial understanding of how the system is expected to behave. It was assumed that requirements would change so the system was started with that in mind. Requirements were reviewed with each iteration. The following three steps were carried out with each iteration.

- **Planning** : This included choosing the appropriate features to be implemented in that iteration. Priority was placed on features based on how useful they would be to students. The main goal of the system is to create a chatbot that can produce results (ability to converse). Each iteration was a week long. Time was fixed and limited, so it was important to stay on schedule.
- **Implementation** : The feature(s) chosen in the planning step are then coded into a working program then integrated with the chatbot system. Some features are more complex therefore sometimes an iteration would have one feature and some iterations have multiple. For example, implementing a working Deep Neural Network model occupied an entire iteration compared to data pre-processing which was implemented with other features.
- **Testing** : Testing was done both manually and automatically. New unit test cases had to be written with new classes and behaviours. However, there needed to be testing of the overall system to ensure no new errors were introduced into the system with new features. The chatbot is mostly back-end therefore white-box testing was most appropriate as it tests the internal intricacies of the system. testing was done using unit tests for all methods. After successful testing, planning for the following iteration begins.
- **Deployment** : Deployment to a work environment in this case means producing a working system that will be demonstrated to the supervisor and second reader. Thus, there are two deployments for this project. First deployment was to demonstrate software feasibility and second is to demonstrate a fully functional system.
- **Evaluation** : The product is evaluated by users, students. the system may have to be updated based in user's feedback and experience. Evaluation is also by the supervisor to see if the system meets the expected requirements.

4 SYSTEM DEVELOPMENT AND IMPLEMENTATION

4.1 Development Software

For implementation of the chatbot, several factors had to be considered when choosing the technologies to be used. The chatbot would have to access the database therefore the language must be efficient with databases. The chatbot would use Machine Learning so it would be beneficial to choose a language with good support,an established community and various libraries, for Natural Language Processing models. Python was chosen as the language to implement the chatbot. It is an established programming language with a large community and libraries. There are several libraries that have been created for Natural Language processing purposes. The main NLP libraries used in this project are NLTK, TFLearn, and NumPy. The Natural Language Toolkit (NLTK) provides libraries and programs for processing human language. NLTK also provides test data that can be used to test the chatbot. Another library used is TFLearn, a TensorFlow Deep Learning Library. TFLearn is an extension of the tensorflow framework, a software library for machine learning and artificial intelligence mostly used for implementing deep neural networks. TFLearn provides a higher-level Application Programming Interface (API) which makes is easier to use. These libraries make it easier to implement the neural network model that will be used to match a text to a pattern and provide the appropriate response. Python's Flask will be used to build the chatbot into a web application with an endpoint that the main website can make POST requests to.

In addition to the language model, the application will also use regular expressions to match queries to responses. This is mainly used for answers that can be fetched directly from the database such as course names, prerequisites, course codes or course descriptions. Using regular expressions has advantages and disadvantages. It ensures more correct answers are generated as information is fetched from a database rather than using the model which attempts to provide the correct answer. The downside being that it provides a potential extra step for input which will not match the regular expressions which leads to slower responses.

Training data is stored in JavaScript Object Notation (JSON) format. This provides an easy way to access the data. Python has a JSON library which provides an efficient and simple way of loading and processing a JSON document. JSON uses human-readable text to store and transmit data. This choice was also better so the document could be stored in the MongoDB database, a document database that uses JSON to represent data. The files are lightweight and thus make it efficient to load and process large amounts of data. This is favorable as the chatbot needs thousands of sentences to learn questions and the appropriate answers. Questions and answers are stored in a pattern-response format. Each pattern has a tag, e.g., "greetings", "registrationForm", "uctLocation", etc. There are hundreds of tags and thousands of questions which the bot is trained on. The JSON file is arranged as follows:

```
{
  "intents" : [
    {
      "tag" : "<intentName>" ,
      "patterns" : ["<questions_pattern>"] ,
      "responses" : [ "<responses_pattern>"]
    }
  ]
}
```

Listing 1: JSON representation of the training data

MongoDB, a NoSQL database, was chosen to store information. A NoSQL database is preferred because data does not have to be strictly structured as in SQL, which makes it easier to store, update or query. It is highly efficient and enables easy updates to the schema design if the need arises.

Potential vulnerabilities of the development process are integration of the chatbot into the overall virtual student advisor website. Training the chatbot could take a really long time which would then affect the progress of the project. Changing requirements could also have affected the progress, however, an agile approach made it possible to adjust to those changing requirements.

4.2 Classes

4.2.1 ***ChatApp***. It is the controller class that connects with the View and Model. However, relative to this paper ChatApp also behaves as the display. Information which would be sent to the View is displayed on terminal. The ChatApp object accepts user input passed in from the View (Website User Interface). It sends the input to the Model and receives a response back. The response would then be passed on to the View to be displayed to the user. The main behaviour of the ChatApp class is to display the previous chats and the current chats. It also uses functions of the DatabaseManager class to get previous chats to be displayed and to save chats of the current session.

4.2.2 ***Bot***. Bot uses the trained model to create a response for that particular sentence using the saved neural network model. The saved model is fetched from memory when a chat session is started. Main behaviour of the class is getResponse which accepts a sentence (user input) and returns a response (bot response).

4.2.3 ***PreProcessor***. It is responsible for cleaning up training data before it is used to train the model. Cleaning up the training data refers to removing trailing spaces, removing some punctuation like question marks, replacing words with their root/stem words, and replacing any acronyms with the complete words. This helps with removing any inconsistencies in sentences, it also makes it easier for the model to recognize patterns and match user input to appropriate responses.

4.2.4 ***Trainer***. This object's main responsibility is using the TFLearn library to initialize the deep neural network, with its hidden layers and output layer. The hidden layer is tested with different number of neurons and batch sizes to see which yields the best results. As the training data is still growing, adding possible questions and answers, the actual numbers are still being tested. Currently the hidden layer has 8 neurons. If the model has not been trained yet or data had changed then it will be trained again, else the model is loaded from memory.

4.2.5 ***DatabaseManager***. Responsible for establishing a connection with the database and querying the database for information or previous chats. Current operations of the DatabaseManager are getChats, saveChats, getMajor, getCourse, getProgram. The chatbot uses these operations to get information from the database.

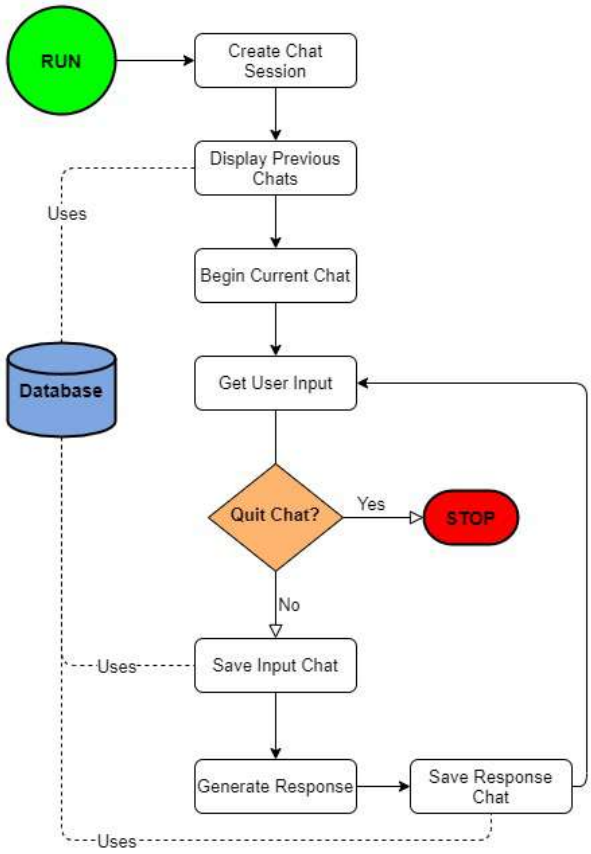


Figure 2: Flow chart showing the flow of the chatbot system.

Figure 2 shows the flow of the program. When a user opens the program, a new chat session is started (Appendix A.2). The previous chats are fetched from the database and displayed for the user to see. Previous chats can only be fetched if the user is registered, else the chats are not saved at all. The user can begin with the current chat session. The user gets a response after each entry. Inputs and responses are saved immediately, this ensures that if the program is unexpectedly stopped the chat history will still be available. If the user wants to end the chat session then the user enters "quit". This is for the terminal version. On the website the user would simply cancel the chat window or bubble.

Appendix A.2 is a flow chart of what happens when a chat session is created. The PreProcessor loads processed data from memory or processes the data if the file is not found and saves it. Training data is then use to either train a new model or load a saved model from memory.

Appendix A.1 is a screenshot of the working chatbot on the website. Each chat is displayed with a timestamp to assist users in recalling when texts were exchanged. To fetch chat history from the database, MongoDB is queried to search for the user's chats using the student number as the unique key. The query is shown in Figure Example of query made to the Mongo database..

```
def getChats(self, studentNumber):
    """
    Retrieve previous chats for a specified student number from the database.
    :param studentNumber: student number being searched.
    :return: the chat history.
    """
    return self.__users.find_one({"username": studentNumber})["chats"]
```

Figure 3: Example of query made to the Mongo database.

4.3 Integration

The goal of the chatbot is to be used in website therefore it has to be integrated into the website. For the front-end to be able use the chatbot, POST requests are sent to the chatbot endpoint (see figure Code snippet of how POST requests are processed and what is returned.) using a unique Unified Resource Allocator (URL). The chatbot app was deployed on Heroku which then generated a unique URL, https://advicechatapp.herokuapp.com/chat that can be made to the chatbot. The URL accepts a student number, if there is one, and the user input in JSON format and returns the generated response together with the student number in JSON format. Requests are transported over a secure communication (Hypertext Transfer Protocol Secure- HTTPS) which offers more data protection compared to regular Hypertext Transfer Protocol (HTTP).

```
@app.route('/chat', methods=['POST'])
@cross_origin()
def response():
    """..."""
    content = request.json
    stuNumber = content['studentNumber']
    userText = content['userText']
    answer = chatSession.getResponse(userText, stuNumber)
    textResponse = flask.jsonify(
        {'studentNumber': stuNumber, 'response': answer}
    )
    return textResponse
```

Figure 4: Code snippet of how POST requests are processed and what is returned.

4.4 Testing

The chatbot was tested together with the virtual student advisor website by users, students and student advisors. users were asked to give feedback on the appearance, quality of responses and ease of use of the system. The system was also tested for average response time of each user input. As this is a chatbot meant for people, whether or not time was satisfactory was dependant upon each individual. The Python time library was used in automated testing as it yields more accuracy compared to a person with a timer. The chatbot was also tested for accuracy with unseen questions.

Users completed the tasks and filled in the google form or answered questions based on the experience. The tasks and questions that people had to complete are listed under the Findings section along with the results.

5 RESULTS AND DISCUSSION

5.1 Findings

Ethics clearance from the university was first acquired prior to testing. The clearance permitted for testing that can only be done remotely (virtual meetings) and only on individuals of the university of cape town. User's identity was to remain anonymous as it would not largely affect the results of the system. None of the interviews would be reproduced or published along with the research.

Testing was conducted on nine individuals. User testing was carried out over virtual meetings on MS Teams. Number of questions and tasks had to be limited to due to time. It is important that users stay focused fatigue could affect their view of the system. The tasks and questions were as follows:

(1) Please try to find the chat bubble that lets you talk to the chat bot.

Table 1: Task 1 results.

Response	Percentage (%)
I found the chat bubble	88.9
I don't know where to look	11.1

(2) Please Rate the Difficulty of the Task Above (Finding the chat bubble). Ranked from very easy (1) to very difficult (5).

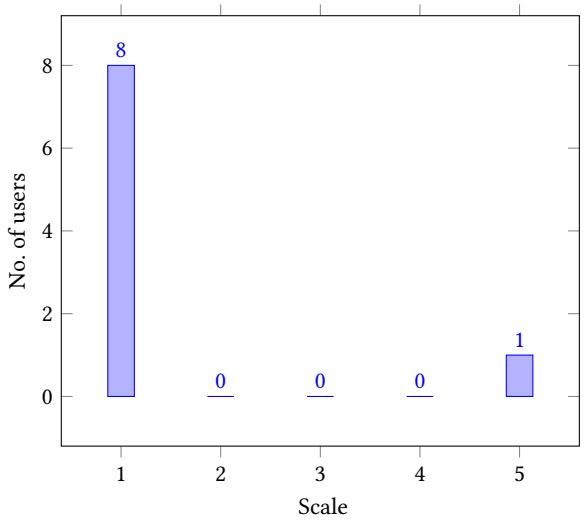


Figure 5: Bar graph showing results of the above question

(3) Is the chat bubble where you expected to find it?

Table 2: Question results

Response	Percentage (%)
Yes	66.7
No	33.3

(4) Try asking the chatbot using a fixed structure (Appendix A.4).

Table 3: Suggested structure task results

Response	Percentage (%)
Could use the suggested structure	66.6
Could not use the structure	33.3

(5) Try to ask the bot about the NQF credits for MAM1000W. (Check boxes)

Table 4: table showing results of finding MAM1000W credits.

Option	Percentage (%)
The bot gave the correct answer	66.6
Did not get the correct response	44.4
Found it difficult to ask question correctly.	33.3

(6) Now ask the chat bot any question(s) of your choice. (Did you get a response to the question you asked?).
- Some of the random question included: "What is astrophysics", "Who is the course convener for Computer Science" and "contact student advisor".

Table 5: Results, when a random question was asked.

Response	Percentage (%)
Received a correct response	77.8
Did not get a correct response	22.2

(7) Are you satisfied with how fast the chat bot answered?.

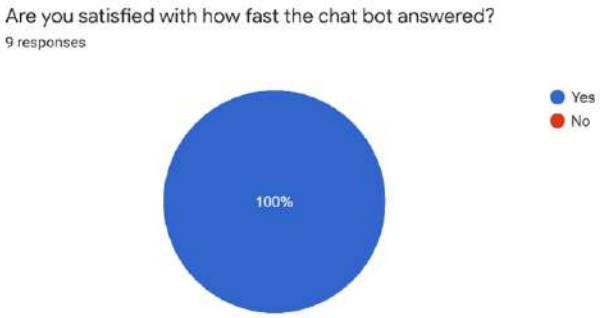


Figure 6: percentage of users satisfied with response time

(8) Now please log out of your account and then log back in. (Can you see your previous chats with the bot?).

Figure 7: success rate of chat history recovery

(9) Do you have any suggestions for the chatbot?
- Suggestions included:
  - Auto-correct.
  - Auto-Complete.
  - Considering Typing errors.
  - Keeping track of frequently asked questions and automatically updating the FAQ page on the website.
  - Better suggestions.

Appendix A.3 shows results of the training phase. With the current model, the best results came from small training data with 8 neurons of the hidden layer. The ideal accuracy is above 95% for a chatbot. It is expected that the accuracy will improve when all the data has been gathered and curated. This section will be expanded and tested more when user testing is complete.

5.2 Discussion

User testing and automated testing yielded a variety of interesting results. The User Interface design of the chat bubble was successful as users could easily identify the chatbot icon and were able to use it. It is worth noting that making the icon bigger could improve the experience as more users would be drawn to it and utilize its functionalities.

The chatbot's response time was excellent. All users were satisfied with the speed at which the chatbot responded. It is worth noting that this could also be affected by the user's quality of internet as the website sends a POST request to get a response from the chatbot. If a user has poor internet it may result in the chatbot giving slower responses.

Users were provided with suggestions on types of questions to ask and how to ask those questions. This is not mandatory; it is simply suggestions to ensure users get the best out of the chatbot. Some users found using the chatbot was easier after looking at the document whilst some found that looking at the suggestions was an extra obstacle to using the application. It is possible that the suggestions may have added a layer of complexity which made using the application even harder. This would need more extensive user-testing to find out if users find the suggestion helpful, if the suggestions could be simplified or, if the suggestions could be provided in a different manner. Suggestions could be provided as auto complete, as a user begins asking a question that could be answered by the database then the suggestion could be displayed to make it easier for the user.

Providing the correct answer is one of the most vital, if not the most vital, aspects of any auto responsive application. Automated testing showed that the chatbot was 90% accurate with unseen data (Appendix A.3) but, this was significantly less with user testing. The chatbot could answer 77% of random questions and only 44.4% of the questions asked using the suggested structure (NQF Credits for MAM1000W). However, accuracy of results improved as the users continued to interact with the chatbot and began to understand the suggestions more. The poor results can be attributed to several factors: incorrect input (i.e., spelling), misunderstanding of the instruction document and inputs including special characters.

Admittedly, these problems could have been accounted for in development with features such as auto correct and recognizing special characters. These problems can be addressed in future iterations of the system. User input should be processed more thoroughly before it is passed to the model and the model should also account for a range of possible user errors.

The chatbot was also successful in storing users' previous chats and displaying the history of chats whenever the user accessed the chatbot. A suggested feature related to chat history is the ability to "clear history". Users should have the ability to delete a chat or all previous chats if they want to. The chats could also be displayed with time stamps to help users keep track of when they sent queries.

It is a secure application. This is made possible by the using a reputable and secure cloud platform hosting system, Heroku. Additional security is offered through using HTTPS to transport information between web applications rather than HTTP. Whenever the chatbot makes web searches, the user will only receive links limited to the UCT domain, this protects against potential malicious software in third-party websites. Data protection is also provided in the database, user's passwords are stored in encrypted format meaning even an administrator cannot view the password.

In comparison to existing virtual student advisor automated system like those by Herry D.Wijaya et al. [20] and, Bhavika R. et al. [13], and Suvethan et al. [17] it was a successful implementation. The application is not only capable of small and short "small-talk" like that by Herry D.Wijaya et al [20] but also capable of answering simple queries meant for University of Cape Town. The language base was also implemented in the South African dialect and thus catered for most UCT students. It is an efficient chatbot that students can use to get results immediately. the drawbacks of the system are the accuracy of the responses given- needs much improvement, managing incorrect grammar or spelling and currently the application is limited to Science and Commerce. the system also needs an application to manage information used by the chatbot and keep it up to date.

The use of a database to store information from handbooks made it easy to understand queries and efficiently retrieve information from the database through pymongo functions.

With a few more iterations and improvements this chatbot could be a very effective feature of the virtual student advisor website. Users would be able to ask any question and get instant responses rather than searching for the information throughout the site. Relative to the site, the chatbot could also be a navigation tool of where to find information or sections of a site. The chatbot has great potential, not only within the site, but also as a stand-alone application that students can use. The application could have a mobile version which would further simplify interaction with the chatbot, especially for students. Use of the chatbot could significantly decrease the number of queries Student Advisors receive and thus give them additional time for complex queries.

5.3 Future Work

Several improvements can be implemented to make it faster and easier to use. The chat box on the website could be editable by the user to fit their own preferences. The chat bubble could be expandable to fit the entire screen, this would make it easier to read current and previous chats at the same time. The regular expressions used to match queries to responses could be improved to account for incorrectly spelled words, to a degree, and sentences with similar key words or structure. More training data would lead to better responses. The more data that the chatbot could train on, the more queries the chatbot will be able to answer. Data gathering is an intensive process that requires structure and time, if there is a dedicated section then more questions and questions can be gathered.

6 CONCLUSIONS

A chatbot is an automated conversational agent that can be utilized to assist users instantly. In the beginning we identified that student advisors are faced with the burden of responding to too many queries and students do not get responses immediately. The project aimed to develop a conversational agent capable of immediately responding to simple student queries about academics and potentially university as a whole. The developed chatbot was successful in responding to simple queries about academics effectively and efficiently. The chatbot could not effectively respond to random queries about university as a whole and thus needs much improvement in this area. It can best be used for simple frequently asked queries. Using the chatbot as per the suggestions would also greatly improve the experience. Testing showed that the chatbot was relatively successful. However, it is still in need of much improvement.

Suggestions for the future would be to split such a project into two sections, a data gathering section and an agent development section. This would allow for more time to be spent on each section and thus yield better results. User testing should also be replicated multiple times as it provides quality and actionable feedback. More research on previous implementations of chatbots and how to optimize them would also improve it. Also develop a User Interface that would make it easy for administrators or Student Advisors to input requirements that have changed from previous years. Extend the system to work for other faculties, departments at UCT and potentially other universities.

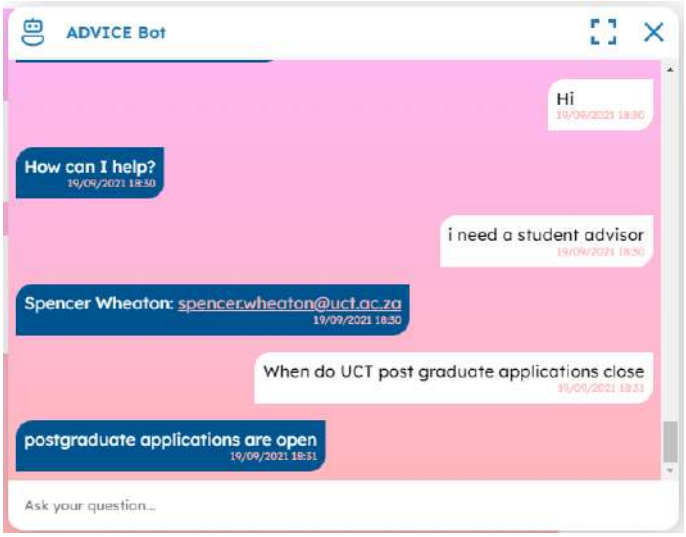
REFERENCES

[1] 2013. Protection of Personal Information Act (POPI Act). (2013). Retrieved June 23, 2021 from https://popia.co.za/
[2] Bayan AbuShawar and Eric Atwell. 2015. ALICE chatbot: Trials and outputs. *Computación y Sistemas* 19, 4 (2015), 625–632.
[3] S A Asri, I N G A Astawa, I G A M Sunaya, K A Yasa, I N E Indrayana, and W Setiawan. 2020. Implementation of Prototyping Method on Smart Village Application. *Journal of Physics: Conference Series* 1569 (jul 2020), 032094. https://doi.org/10.1088/1742-6596/1569/3/032094
[4] Balbir Singh Bani and Ajay Pratap Singh. 2017. College Enquiry Chatbot Using ALICE. *International Journal of New Technology and Research* 3, 1 (2017).
[5] Jack Cahn. 2017. CHATBOT: Architecture, design, & development. *University of Pennsylvania School of Engineering and Applied Science Department of Computer and Information Science* (2017).
[6] Yuen Ren Chao. 1943. Languages and Dialects in China. *The Geographical Journal* 102, 2 (1943), 63–66. http://www.jstor.org/stable/1790133
[7] Gunther Cox. 2020. About ChatterBot. (2020). https://chatterbot.readthedocs.io/en/stable/

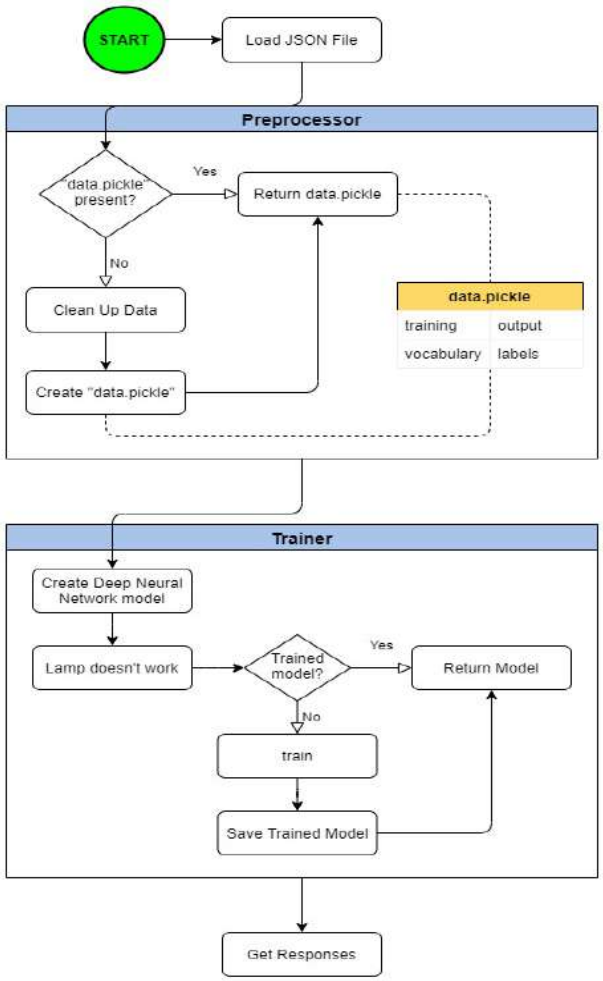
[8] Berardina De Carolis, Sebastiano Pizzutilo, Giovanni Cozzolongo, Pawel Drozda, and Francesca Muci. 2006. Supporting students with a personal advisor. *Journal of Educational Technology & Society* 9, 4 (2006), 27–41.
[9] Eun Go and S. Shyam Sundar. 2019. Humanizing chatbots: The effects of visual, identity and conversational cues on humanness perceptions. *Computers in Human Behavior* 97 (2019), 304–316. https://doi.org/10.1016/j.chb.2019.01.020
[10] Oded Gurantz, Matea Pender, Zachary Mabel, Cassandra Larson, and Eric Bettinger. 2020. Virtual advising for high-achieving high school students. *Economics of Education Review* 75 (2020), 101974. https://doi.org/10.1016/j.econedurev.2020.101974
[11] Mollie Krupp. 2014. The Student-Advisor Disconnect. *Women in Higher Education* 23, 11 (2014), 18–18. https://doi.org/10.1002/whe.20133 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/whe.20133
[12] Roger Lass. 1995. South African English. *Language and social history: Studies in South African sociolinguistics* (1995), 89–106.
[13] Bhavika R. Ranoliya, Nidhi Raghuwanshi, and Sanjay Singh. 2017. Chatbot for university related FAQs. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 1525–1530. https://doi.org/10.1109/ICACCI.2017.8126057
[14] BA Shawar and E Atwell. 2002. *A comparison between Alice and Elizabeth chatbot systems.* University of Leeds, School of Computing research report 2002.19. https://eprints.whiterose.ac.uk/81930/
[15] Bayan Abu Shawar and Eric Atwell. 2007. Different measurement metrics to evaluate a chatbot system. In *Proceedings of the workshop on bridging the gap: Academic and industrial research in dialog technologies.* 89–96.
[16] Steve McRobb Simon Bennett and Ray Farmer. 2010. *Object-Oriented Systems Analysis and Design Using UML.* McGraw-Hill Education, New York, New York, USA.
[17] N Suvethan, K Avenash2 MAQ Huzaim, R Mathusagar, MPAW Gamage, and A Imbulpitiya. 2016. Virtual Student Advisor using NLP and Automatic Appointment Scheduler and Feedback Analyser. (2016).
[18] Clive Upton and John David Allison Widdowson. 2013. *An atlas of English dialects: region and dialect.* Routledge.
[19] Joseph Weizenbaum. 1983. ELIZA — a Computer Program for the Study of Natural Language Communication between Man and Machine. *Commun. ACM* 26, 1 (Jan. 1983), 23–28. https://doi.org/10.1145/357980.357991
[20] Herry Derajad Wijaya, Wawan Gunawan, Reza Avrizal, and Sutan M Arif. 2020. Designing chatbot for college information management. *IJISCS (International Journal of Information System and Computer Science)* 4, 1 (2020), 8–13. http://www.ojs.stmikpringsewu.ac.id/index.php/ijiscs/article/view/826

A SUPPLEMENTARY INFORMATION

A.1 Chat Session Example



A.2 Chat Session Flow Chart



A.3 Chatbot Training Results

Training size	Training Time (minutes:seconds)	Testing Accuracy (%)	Hidden Layer Size
200 Questions	16:29	91.2	8
400 Questions	20:36	87.1	10
600 Questions	24:43	86.8	12
800 Questions	28:50	83.4	14

A.4 Suggested Chat Structure

Next Page...

CHAT BOT SUGGESTIONS

Example: “careers in major_name” -> “careers in computer science”.

CAREERS

ASSOCIATED MAJORS

Careers in **major_name.**

SALARY RANGE

Salary range for **major_name.**

major_name salary range.

DESCRIPTION

major_name description.

ALL

What is **major_name.**

COURSES

COURSE CODE

Course name of **course_code.**

What is the course code for **course_name**?

NQF CREDITS

How many credits does **course_name/course_code** have?

course_name/course_code NQF credits.

NQF credits for **course_name/course_code**

NQF LEVEL

What is the NQF level of **course_name/ course_code.**

course_name/course_code NQF level.

NQF level for **course_name/course_code.**

GLOSSARY

Describe **word**

MAJORS

What are the majors offered at UCT?

First year courses of **major_name/major_code**

PROGRAMMES

Length of **programme_code**

Minimum credits for **programme_code.**

STUDENT ADVISORS

Contact details for **name surname.**

name surname contact details.

name surname email address.

I need a student advisor.