Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Engineering

Personalisation in Chatbots

Ciara Byrne

Supervisor: Prof. Vincent Wade

April 17, 2023

A dissertation submitted in partial fulfilment
of the requirements for the degree of
MAI (Computer Engineering)

Declaration

I hereby declare that this dissertation is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

I consent / do not consent to the examiner retaining a copy of the thesis beyond the examining period, should they so wish (EU GDPR May 2018).

I agree that this thesis will not be publicly available, but will be available to TCD staff and students in the University's open access institutional repository on the Trinity domain only, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement. **Please consult with your supervisor on this last item before agreeing, and delete if you do not consent**

Signed: ______________________ Date: ______________________

Abstract

Chatbots are interactive computer programs which use natural language dialogue to help a user carry out a task or have a chat, but this varies depending on the type of chatbot. Chatbots are increasing in popularity and widely available with the rise of voice-based chatbots such as Siri and Alexa or text-based chatbots such as ChatGPT. They are used in many areas, one of which is education. This study described an educational chatbot which tutored students in natural language as a human teaching assistant would, guiding a student through exercises, answering questions and adapting the pace of learning based on individual students' needs. The chatbot designed in this study created an effective and engaging learning environment using personalisation and adaptation. Students were given structured Structured Query Language (SQL) exercises based on their ability and personalised feedback on their answers. Rasa Open Source framework was used to build and test the system and all training and test data was created for the purpose of this study. The functionalities of two chatbot models were tested, one chatbot model trained with a Rasa recommended pipeline and the other trained with a Spacy pre-trained model. It was shown that both models performed similarly in accuracy, F1-score, precision, and response selection. However, the SQL chatbot trained with the Rasa recommended pipeline performed better in intent classification with a p-value of 0.01171.

Lay Abstract

Chatbots are computer programs that engage in conversation with a user. They are used in everyday life, for example Siri or Alexa. Chatbots are applied in many areas such as education. This study described the design and development of a chatbot which teaches students a programming language called SQL through conversation, letting them know if they were wrong and where they went wrong. Sample conversations were used to train the chatbot how to understand what the user was saying. Two different chatbot models were trained and found to be largely similar, however, one was shown to better understand the user.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Professor Vincent Wade for all his support, guidance and motivation during this project. Professor Wade was a great mentor and supervisor over the last year. I would also like to thank my amazing friends and family who have pushed me to reach my full academic potential. I would not be where I am today without them.

Contents

List of Figures

List of Tables

1 Introduction

Chatbots, also known as conversational agents or dialogue systems, are computer programs that engage in and mimic natural human conversation [4]. Chatbots have come a long way since some of the first chatbots ELIZA [5] and PARRY [6]. With advancements in natural language processing, machine learning, and artificial intelligence, chatbots are becoming more and more sophisticated. Current chatbots are not just simple rule-based programs but are capable of understanding context, recognising emotions, learning from past interactions, generating responses and have a human-like natural conversation. They are becoming more personalised, and accurate, enabling them to offer a more engaging and easy experience to users. Furthermore, the increasing availability of Application Programming Interfaces (APIs) and pre-built frameworks is making it easier than ever for developers to build and deploy chatbots. ChatGPT [7] is an example of a new and popular chatbot that has caught the attention of many this year. This is because it is able to have clean conversation with a human and can generate natural responses.

Chatbots are currently used and have been researched in many sectors such as healthcare [8], customer service [9], hospitality and tourism [10] and education [11]. Chatbots are also used in everyday life such as voice controlled conversational agents like Siri, Alexa or OkGoogle. According to a study done in 2017 [12], chatbots are becoming increasingly popular due to their ease of use, and the speed and convenience at which information can be obtained from them. Not only can users ask them nonsensical questions without fear of judgement, but they also serve as a source of information and can even sometimes alleviating feelings of loneliness. In addition to their practicality, the novelty of chatbots appeals to users, making them a popular choice. The study highlights productivity as a key factor of chatbot use, with users seeking quick and reliable feedback. To keep users engaged, chatbot design should incorporate personalisation and social features while maintaining quick and reliable interactions and responses.

An area where chatbots are especially useful is education. The potential that chatbots have to enhance a student's learning is huge [13] and this potential needs to be taken advantage of. The aim of implementing an educational chatbot is to emulate a teacher or teaching assistant, both in conversation, personality and educating. An educational chatbot has to have the characteristics of a teacher such as encouraging, supportive, giving feedback, and progressively advancing their knowledge of a subject. The challenge of designing personalised chatbots is that there are a plethora of features that can make the experience more personalised. In this research, many features are implemented into an educational chatbot to enhance the personalisation and therefore improving the students' learning and experience. A student's engagement, satisfaction and understanding are enhanced through the use of a chatbot that can adapt to the student and support them as they tackle practical problems.

Chatbots can help solve many problems in education and there are many reasons why they should be implemented in classrooms or at home. Chatbots can help make education more accessible by providing learning resources and assistance to students with disabilities or those who are unable to attend traditional classrooms due to physical, geographical, or economic constraints. They can help engage students by providing interactive learning experiences that are more interesting and engaging than traditional lectures or textbooks. This can help student's more effectively retain information and improve their overall learning experience. When there is a lack of teaching assistance or resources chatbots can provide additional support to teachers and students by answering frequent questions, providing educational resources, and offering advice. This can help reduce teacher workload and improve student performance.

1.1 Objectives

This research will investigate the effect of personalisation in a chatbot. This project will be broken into three main objectives:

1. Perform a literature review on the current state-of-the-art in personalised chatbots, educational chatbots and task based chatbots. This objective aims to report the highlights of chatbot research including their methodologies, technologies, models, corpus and evaluation. Thus, understanding chatbots and their technologies.

2. Take a case study approach to design, build and implement a personalised chatbot that emulates a teaching assistant teaching the coding language SQL, that understands context and has a pedagogical model.

3. Design an evaluation process for the chatbot system and present results and conclusions about the performance and feasibility of the system. Evaluate the functionality, usability, effectiveness and user experience of the system.

1.2 Methodology

This dissertation takes a case study approach [14]. A case study approach involves taking and developing a real-life example to gain a deeper understanding of a research topic. The study developed a prototype of an SQL tutor chatbot to gain an insight into personalised chatbots in the education domain. All training data was designed and developed for the purpose of this project. It consisted of conversations and paths a tutor might take when instructing a student. This included conversational text, SQL questions and answers. A questionnaire was designed for the evaluation process which was given to participants detailed in Chapter 5. Test data in the form of sample conversations with hard coded user input, action, intent and entity classifications was designed and generated for the sake of this study.

1.3 Overview of Thesis

The outline of this report is as follows. Chapter 2 reviews the current state-of-the-art and literature in chatbots, task-based chatbots, personalised chatbots and chatbots in education. It identifies types of chatbots currently on the market or in research, their strengths and weaknesses, and the key technology or concepts used in this study. Chapter 3 reviews the design, architecture and data for the chatbot system presented in this study. This includes details of the application, overall architecture of the system, design and technology choices. Chapter 4 discusses the implementation of the design which includes explaining details of how the developed software works and how it was implemented. In Chapter 5 this study evaluates the system that was designed and implemented including evaluation objectives, setup, results and discussion of results. Finally, in Chapter 6, conclusions are drawn and future work is discussed.

2 State of The Art

2.1 Overview

2.1.1 History

Chatbots have been around for a long time. One of the first being a rule-based chatbot Eliza [5], which was developed in 1966 to replace a psychotherapist. ELIZA was one of the first dialogue systems to pass the Turing Test [15] which was a test proposed by Alan Turing in 1950 to judge how human-like a machine was. Although ELIZA passed this test, it lacked context understanding due to it being a rule-based keyword matching system. Similar rule-based systems followed such as PARRY [6] in 1972 which responded to a user as a person with Schizophrenia.

ALICE (Artificial Linguistic Internet Computer Entity) [16] was released online in 1995 and was inspired by Eliza. It was a natural language processing bot and although impressive at the time, it could not pass the Turing test. In 1997 a chatbot named Jabberwacky, designed by Rollo Carpenter, made use of Artificial Intelligence. This chatbot was designed so that it could be entertaining, funny and have a natural conversation. Jabberwacky was designed to have conversations and act as a social bot [17]. Smarterchild [18], on the other hand, was a chatbot or an early virtual assistant released in 2001 and was used in instant messengers such as MSN to complete daily tasks and retrieve information easily such as movie times, weather, sports scores etc. Smarterchild performed tasks for the user, making it a task-based chatbot

Since the early to mid 2000's, chatbots have been improved and tailored to a lot of sectors and markets.

2.1.2 Current Public Chatbots

Large Language Models (LLMs) are used in state-of-the-art chatbots today. LLMs generate and understand natural language using deep learning models. LLMs produce great results due to their complex neural network architecture. They have many parameters and can understand the relationships between inputs due to the transformer which they are built on. They also perform well due to the way they are trained. LLMs are first trained on large amounts of unlabelled data, they are then trained on a smaller labelled dataset. This provides high accuracy with minimum human effort [19](when it comes to annotating.)

ChatGPT is a state-of-the-art chatbot that was released by OpenAI in 2022. It has good language understanding and generation due to the use of the powerful models GPT 3.5 and GPT 4. ChatGPT was fine-tuned using a mix of reinforcement learning and supervised learning. According to an article by Y.Shen et al. [20] there are advantages and disadvantages of the LLM based chatbots, including ChatGPT. ChatGPT can engage in natural conversation by understanding user intent and generating natural responses. According to the manuscript, it can help in medical imaging diagnostics and summarise medical reports. Despite its natural language capabilities ChatGPT has drawbacks. The authors say ChatGPT tends to "hallucinate", where it provides incorrect responses, such as made-up information or terms. It is not connected to the internet so it will confidently answer a user's question with information, such as references, that it fabricated as it was not in its training set. ChatGPT depends on the user to control the conversation. This can lead to errors as it will not ask follow up questions, instead it will make assumptions, which are sometimes inaccurate. The authors talk about other downsides to the chatbot such as report falsification, plagiarism, and unethical image fabrication [20].

2.2 Task-Based Chatbots

Task-based, or task-oriented, chatbots help users complete tasks using natural conversation [21]. They are programmed to understand user intent and provide relevant responses or actions. These chatbots are used in many sectors such as customer service, online shopping, food ordering and more. Research has shown that productivity is a key factor for chatbot use, with users seeking quick and reliable feedback [12].

Task-oriented chatbots are built and trained in a variety of different ways and for different applications. A paper written in 2020 [3] described studies on task-oriented systems, which were typically implemented using either pipeline or end-to-end methods. Generally, pipeline systems are divided into several modules, including Natural Language Understanding (NLU), dialogue state tracking, dialogue policy, and Natural Language Generation (NLG). End-to-end methods build the system using a single model. Pipeline systems are more stable and most real-world systems use them, while end-to-end methods are easier to build but are more of an uncontrollable black box.

Pipeline systems normally include NLU, dialogue state tracking, dialogue policy, and natural language generation. The NLU component takes user input and tries to understand the intention of the user using intent and slot structure. Intent detection can be thought of as understanding a users intention, and slot value recognition can be thought of as a sequence labelled problem. Dialogue state tracking estimates the user's goals at each time step using previous turns and the dialogue policy generates the action that the system will next take. Natural language generation uses the dialogue act to generate a response in natural language. For a good user experience, responses should be correct, specific, informative and in natural language. Most end-to-end methods use sequence-to-sequence models and require large amounts of training data.

Data collection comes in many forms, including human-to-machine (HM), machine-to-machine (MM), and human-to-human (HH). Human-to-machine involves a user interacting with the system to collect data, machine-to-machine is where a machine plays a user and the system, human-to-human involves conversation data between two humans where one user plays the system and one plays the user. Collecting data is difficult, especially domain-specific data as annotating data is expensive and time-consuming. Examples of a table of corpora and how they collected data from [3] can be seen in table 2.1.

Name	Application	Method	Size
WOZ [22]	Restaurant Booking	HH	1.2K
SimD [23]	Restaurant/Movie Bookings	MM	3K
MultiWOZ [24]	Multi-domain	HH	10K
CoSQL [25]	Multi-domain	HH	3K
SGD [26]	Multi-domain	MM	23K

Table 2.1: Example From Corpora Overview Table in [3]

An example of an end-to-end task-oriented chatbot was described in a paper [27]. It proposed using a generative adversarial network (GAN) to generate sentences for a task-oriented chatbot to diversify conversation and improve the quality of responses. The GAN model consisted of a generator which produced diverse sentences and a discriminator that judged the generated sentences versus the ground truth sentences. Sentence information was extracted using the generator by combining the attention model with a sequence-to-sequence model using long short-term memory (LSTM). The discriminator used a reward mechanism that assigned low rewards for repeated sentences and high rewards for diverse ones. A Chinese corpus of restaurant booking conversations was used to train the model. The research found that the model outperformed existing models as it generated more diverse and contextually appropriate conversations. Human testing showed that the model had better response quality, diversity and satisfaction.

A paper by Yan et al. [28] described the pipeline design and development of a task-oriented chatbot for online shopping that helped customers complete purchasing tasks like searching for products and answering questions in natural language. The authors used existing resources along with crowd sourcing to build training data as it was hard to gather large amount of labelled training data in the domain of interest. Various data sources were used in the system, including a product knowledge base, logs of searches, user clicks and community sites where users shared their intents about products. Product categories were detected using a Convolutional-Neural-Network (CNN) approach. The authors also developed their own algorithms for product attribute extraction. Dialogue state tracking was done to ensure a good user experience in multi-turn conversations and dialogue management was used so that the bot could use the dialogue state tracker to select the next action. The chatbot performed well in utterance type classification but there was no user feedback recorded. It was session/context-aware which was important to avoid user frustration.

Another example of an application of a task-based chatbot was presented in a paper by Z. Wei et al. [29]. This bot diagnosed medical conditions by collecting information about patient symptoms. A dataset was built for the system using patient self-diagnosis and conversations between patients and doctors. The system chatted with patients to collect additional information about symptoms that they may have excluded from their reports. Results showed that including these extra symptoms improved the accuracy of the system. Template-based models were used for NLU and NLG in this chatbot. The system used Markov Decision Process (MDP) and learned dialogue policy through reinforcement learning. Performance of the system was compared with baselines using Support Vector Machines (SVM)[1], random chatbot, and rule-based chatbot. A random chatbot chose an action randomly in response to a user's action and a rule-based bot chose the next action based on strict rules. The results of evaluation showed that this system outperformed all baselines or the random, rule-based and SVM-based chatbots.

Task-based chatbots can vary from basic Questions and Answer (QA) chatbots to personalised chatbot assistants. Personalisation has been defined as "a process that changes the functionality, interface, information access and content, or distinctiveness of a system to increase its personal relevance to an individual or a category of individuals" [30]. Chatbot personalisation can be generalised into two categories, chatbot persona-based personalisation (which aims at matching user personality and chatbot personality) and personalisation through dynamically adapting the system to an individual user based on some other element (behaviours, needs, expectations and abilities [31]) to improve their experience.

Personalisation with a persona-based chatbot was identified in a study [32], which investigated the effects of making a chatbot personalised when it comes to consumers. It found out if chatbots can have or emulate a personality. The effect on customer engagement when matching chatbot personality with customer personality was investigated, as well as how personalised chatbots affected the company financially. The authors used two

[1] SVM is a supervised machine learning algorithm used for classification or regression

personality types to do this and therefore designed two chatbots with these personalities, introvert and extrovert. Machine learning techniques were used to predict the personality of the consumer based on responses and previous interactions. The personality of the chatbot was assigned based on this prediction. The results showed that using a chatbot with a matching personality to the customer increased customer engagement, with some customers spending around twice as long in a session with a chatbot with a matching personality compared to a chatbot with a non-matching personality. Matching customer and chatbot personality also increased the likelihood of purchasing and therefore, benefited the company financially. To keep users engaged, chatbot design should incorporate personalisation and social features while maintaining quick and reliable interactions and responses [12].

Another example of a personality-aware chatbot was mentioned in a study done in 2022 [33] where the chatbot realised a user's personality and made a job recommendation based on this. There were two chatbots designed using the Microsoft Azure Bot Framework, one extroverted and one introverted. A personality report was filled out by the user so that their personality could be realised. Once a personality was discovered, one of the two chatbots was deployed and at the end of a conversation a job recommendation was made to the user. When tested both chatbots were given to participants, and the user assessed the usability of both bots including indicating their preference. It was found that users trusted the chatbot that matched their personality more and they had more satisfying and appropriate conversations with the matched personality in comparison to the non-matched chatbot.

As previously mentioned, another type of personalised chatbot was one that did not focus on personality of the chatbot but rather adapting the system to the user. This type of personalisation was observed in the chatbot Reflection Companion [34]. This system had a chatbot in a mobile app which used fitness tracking data to offer personalised adaptive graphs and reflection dialogues. The goal was to get user to engage by asking them to analyse their fitness trends and reflect on them. Language Understanding Intelligent Service (LUIS) NLU API was used to understand user responses and a web server contained dialogue management, user profile data and graph generation. Users found the reflection aspect led them to be more aware, motivated, and mindful. The personalised adaptive conversation (chatbot) and visual (graphs) aspect of this system made users more engaged and therefore made them more committed to their fitness goals. One challenge with long-term user engagement noted by the authors was to further diversify dialogue, as users often encounter repeated dialogue with regular use.

Name	Year	Purpose	Technology	Remarks
Hsueh and Chou [27]	2022	Restaurant reservation	GAN, LSTM	Better response quality, diversity, satisfaction
Yan et al. [28]	2017	Online shopping	Pipeline, CNN	Good utterance type classification
Wei et al. [29]	2018	Medical Diagnosis	Markov Decision Process, reinforcement learning	Outperformed baselines
Shumanov and Johnson [32]	2021	Online shopping	Machine learning	Person based engages customer
Fernau et al. [33]	2022	Job Recommendation	Microsoft Azure Bot Framework	Persona based, users more trusting satisfied
Reflection Companion [34]	2018	Fitness App	Language Understanding Intelligent Service(LUIS)	Adaptive, engaging, motivating

Table 2.2: Task-Based Chatbot State of the Art Summary

2.3 Chatbots in Education

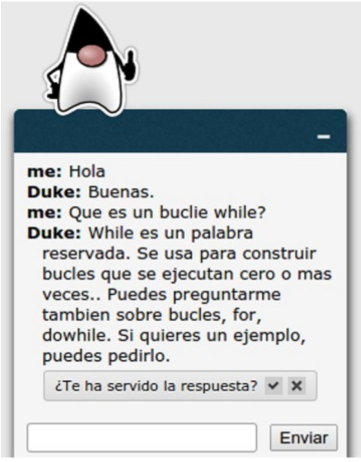
Chatbots can benefit both teacher and student [35]. They offer students an engaging and personal learning experience which is accessible to students when and where they want. Educational chatbots make the course material easily obtainable in a quick and user-friendly way. Teachers benefit from these systems too, they take pressure off teaching staff when students require assistance and are a tool to aid teachers in helping students understand a topic.

Educational chatbots provide many benefits to student and teacher but in some cases this may not be true. For example, in a study done by L.K. Fryer et al. [36], students studying a foreign language in university were asked to rate their interest in a subject using a human partner compared to a chatbot partner. The study verified that students find topics more interesting when they have a human partner. There was a novelty effect where some students initially found the chatbot more interesting but as time went on in the study this faded, and the human was favourable.

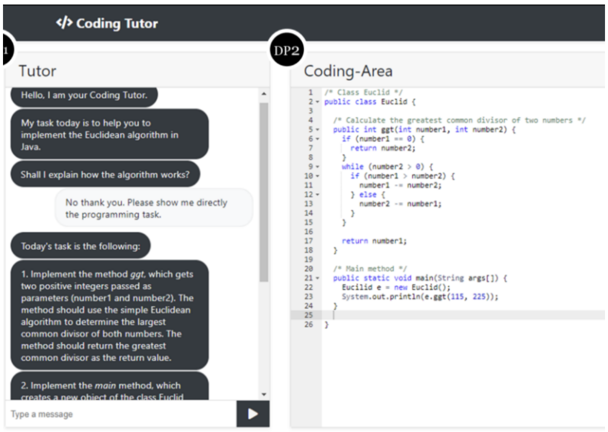
The authors of Coding Tutor [37] had a goal to design a chatbot system that assumed the role of a teaching assistant. Coding Tutor guided, supported and answered questions that beginner programmers had while completing a practical Java programming problem. A teaching assistant traditionally guides students through work without giving them the complete answer, the goal of this chatbot was to mimic that. They created a coding tutoring system which interacted and taught students code using natural language. This system could also be adapted for other programming languages. NLP.js was used for user intent recognition. A database was used to store answers to students' questions where the answer was found by intent recognition, if an answer cannot be found, chit-chat took over. The pedagogy in this paper was thoroughly researched, Bloom's Taxonomy [38, 39] was used to define learning objectives so a student could learn effectively. Lecturers and e-learning experts were interviewed and adaptive learning paths for students were designed based on these interviews. Most students found the Coding Tutor useful and easy to use. They emphasised the benefit of a step-by-step guidance and also pointing out errors in code. Some issues they faced were the responsiveness of the bot.

Another chatbot that was developed to teach Java to students called Duke, was outlined in the paper [40]. The paper described the development of a personal chatbot agent to assist students learning Java, which used ChatScript to build the interface. The agent had manually created learning paths which changed depending on student questions. Duke had a knowledge base which was scrapped from a Java course online and mapped to a domain specific ontology (ie. Java). This tutor helped students with their questions about Java using domain knowledge, stages in the learning path and historical student queries. To incorporate natural language a social dialogue database was constructed for the study. Two versions of the system were given to the users, one which was purely information retrieval and one that used dialogue and natural language. Students appreciated natural language interaction and this encouraged them to interact with the bot. Users prefer using a conversation with Duke to find answers over key word searches.

A side by side comparison of the two mentioned Java tutors can be seen in figure 2.1. Duke (although in Spanish) can be seen2.1a to be a much more simple tutor. It was just a conversational interface where a student asked questions about a Java topic. In the Coding Tutor dialogue example2.1b it appears to be more practical. It gave a student questions and allowed them to try it out, answering any questions a student might have as they complete the question.



(a) Duke Dialogue Example From [40]



(b) Coding Tutor Dialogue Example From [37]

Figure 2.1: Two Java Tutor Chatbots Compared

SQL Tutor on the Web (SQLT-Web) [41] and SQL-Tutor (SQL-Tutor) [42], programmed in LISP, is old but interesting example. This SQL tutor understood, learned and adapted depending on the students' abilities. It tutored a student in the syntax of SQL. The model was stored on a local machine which was not ideal as the student was tied to the one machine if they wanted to continue their progress. However, if a student was using the same machine they could be recognised as a returning user by entering their name. The system incorporated personalisation by using a level of expertise to keep track of a student's progress. Constraint-Based Modelling (CBM) [43] was used to correct and give feedback on student answers where the student answer was matched to ideal solutions and to constraints. SQL can have equivalent queries where there is more than one correct solution to a problem. This system contained only one correct solution to each problem, however it could use constraints to check equivalence between the student answer and the ideal solution where it was able to tell if a solution was correct. As this was a tutoring system there was a pedagogical module that provided feedback about a solution a student entered, handled additional explanations and could infer what areas the student had problems in so that the next question could be given accordingly. A possible flaw in the system was that there was no curriculum structure, only constraints relating to basic SQL. Despite this the researchers found that most of the students liked learning at their own pace and using practical problems to do this. An interesting finding from this research was that many of the students "found learning with SQLT-Web to be more personal than lectures". Even though it was reported that SQLT-Web was more personal than lectures it still lacked natural language capabilities.

Oscar [44] was another SQL tutor. Oscar was a personalised educational chatbot designed to tutor undergraduate Science and Engineering students SQL. A pattern matching approach was used to develop a conversational agent that could recognise and dynamically adapt to a student's learning style. The chatbot offered online SQL revision tutorials and personalised the learning material by offering it to a student sequentially and in the style to best suit their needs (verbal or visual) while also offering them intelligent feedback. Oscar used a knowledge base of tutor material and conversation scripts to provide SQL tutoring in natural language. Results showed that the chatbot was well-received, understandable, and helpful, and students' test scores improved after using Oscar. The interface was natural and easy to understand. It was also found that 90% of the group said they would use Oscar to support class tutoring.

Python-Bot [45] was a chatbot that taught Python to a student and implemented AI techniques to do it. This chatbot helped novice programmers to understand basic syntactic structures and semantics in the Python programming language. They used a chatbot API called snatchbot. The snatchbot API provided the Natural Language Processing (NLP)[2] capabilities, a knowledge unit and a message bank of predefined answers. When using this chatbot it was found that it did not have natural language understanding. It simply gave a series of prompts and one of the predefined questions was chosen. There was no conversation, interaction and no personalisation. The bot gave an exercise but did not allow the user to input (it told the user to use a pen and paper). Therefore, there was no feedback and no way of learning from mistakes. There was an option for support, but it made an appointment with a tutor or lecturer, defeating the pedagogical aspect of the chatbot. This chatbot did not remember students and what they last tried, and it did not keep track in conversations.

Another chatbot which helped students learn Python was Sara [47]. Sara appeared during an online video lecture on a side bar so that students could practice their python knowledge as they watched lectures. A scaffolding-based approach was used, for example, asking the user to explain a concept in Python and breaking down incorrect answers asking the user to explain individual components. Sara interrupted the lecture and asked practical problems, classifying the user's response as correct, wrong, or unsure. Pattern matching was applied and the NLP.js framework was used for intent classification. The studie's pre-trianed NLP model was applied to a user's answer and the dialogue adapted based on the result of this. Evaluations from this study showed that scaffolding-based conversational agents may create better interactions with learners, deepen a user's knowledge on a subject and tailor to individual learning gaps. Some learners were impressed by how much Sara could understand however others felt she did not fully understand their questions which lead to frustration among users. This limitation of Sara was due to the quality of the training data.

Another programming tutor that was of interest was Edubot [48]. Edubot was a chatbot

[2]NLP is a field of study that investigates ways in which computers can analyse and process natural language [46]

designed to teach MATLAB in introductory programming courses. The study aimed to determine how students in first-year programming courses interacted with a chatbot and their opinions on using a chatbot in their courses. Some of the main challenges in developing chatbots in all domains was creating a natural dialogue and developing the knowledge database to do so. A knowledge database for Edubot was populated with MATLAB functions that related to the course. An interface of Edubot was provided through a plugin to the web-based learning management system (LMS), Canvas. Microsoft QnA Maker was used for AI processing in this chatbot. Students were introduced to EduBot at the start of term and asked to interact with it throughout the semester when they had any questions about MATLAB. When students asked a question, the answer was returned if it was found in the database, if not the professor was notified to fill in the answer. The study relied on student interaction with the system to develop the corpus, which failed due to a lack of student involvement with the chatbot. Due to this, when the chatbot was evaluated it failed to answer a lot of student questions. Students also said that it was quicker and easier to use Google for answering their questions. The majority of students expressed that Edubot would be more beneficial in addressing enquiries related to the practical application of programming concepts, presenting course-relevant examples, and guiding students towards course-specific resources.

A paper by Clarizia et al. [49] described a model system for E-Learning chatbots and a prototype. They implemented this chatbot for two courses, fundamentals of Computer Science and Computer Networks. The Latent Dirichlet Allocation (LDA) algorithm was used to extract key words and infer user intention. This algorithm performed basic NL steps such as removing stop words, stemming and extracting keywords. A knowledge base consisting of content, practice questions, test questions was stored on a database server. The chatbot was given to students to try, and was evaluated using a questionnaire where students were asked about the usability and accuracy of the chatbot. It performed terribly in accuracy with 83% of students saying it provided incorrect responses. The researchers suggest the bad results were due to the lack of technical language understanding using C++ for example. Despite this, students found the chatbot easy to use.

An earlier and thoroughly researched educational chatbot AutoTutor [50] was developed for teaching Newtonian qualitative physics and computer literacy. It was designed to guide students through problems using dialogue, giving hints, correcting misconceptions, and providing feedback. The system would draw answers from a student by asking questions and giving prompts so that the answer to a question was built with every conversation turn. AutoTutor had a curriculum script that contained questions, problems, expectations, and most relevant subject matter content. This curriculum script was a repository of content associated with a question or problem, and subject matter. Experts created this content using an authoring tool called AutoTutor Script Authoring Tool. AutoTutor presented questions and answers drawn from the curriculum script. It used Latent Semantic Analysis (LSA) to match student input to expected answers. LSA is a high-dimensional statistical technique that measures the conceptual similarity of any two pieces of text, such as words, sentences, paragraphs, or lengthier documents. The system set a threshold match in a certain range depending on the problem. As the threshold increased, a learner needed to be more precise in order for their answer to be marked as correct. AutoTutor had a dialogue management component that adaptively selected hints and prompts to achieve expectation (answer) completion. Augmented finite state transition network and algorithms such as fuzzy production rules were applied to select dialogue move categories (give feedback, corrects a student when wrong, answers, summarises the entire answer, etc.). An example of personalisation in this system was when the system used student ability to decide how much help a student could get. Experts rated it effective at promoting learning gains at deep levels of comprehension. AutoTutor has been shown to be more effective than reading textbooks alone. The system was also evaluated using a Bystander Turing Test. In this it was found that the bystander could not discriminate between computer and human dialogue turns, indicating that AutoTutor provided a good simulation of a human tutor.

Online classes require some sort of tool to keep students engaged, a chatbot was one tool that could be used to do this according to [51]. This study claimed the future of engaging online learning was through the use of personalisation in chatbots. They proposed a chatbot that could support a student in an online class. An engaging tool such as a chatbot would have benefits such as providing individualised feedback, motivation and support which could help a student be academically successful. This research aimed to achieve this in a web-based chatbot system which was tested on an online graduate level technical course. The tutor designed by this research asked the student a reflective question about the topics covered that week and the bot gave general follow up questions so that the student elaborated on their previous answer. It did not have specific domain knowledge or the ability to aid students in queries. It prompted a student to evaluate what they learned and what they struggled at. The chatbot was rule-based and its behaviour was predefined. The researchers improved the system by working with experts, students and teachers and found that learning reflection can be effective in online learning.

Name	Year	Subject	Technology	Remarks
Coding Tutor [37]	2019	Java	NLP.js	Useful, easy to use, good feedback
Duke [40]	2018	Java	ChatScript	Good natural language capabilities
SQLT-Web [41, 42]	1999,2003	SQL	LISP, Constraint-Based Modelling	More personal than lectures, feedback
Oscar [44]	2010	SQL	Pattern matching	Helpful, test scores improved, natural
Python-bot [45]	2020	Python	Snatchbot API	Bad natural language and personalisation
Sara [47]	2020	Python	Pattern matching, NLP.js	Structured learning
Edubot [48]	2018	MATLAB	Microsoft QnA Maker	Not effective for learning
Clarizia et al. [49]	2018	Computer Science Computer Networks	Latent Dirichlet Allocation (LDA) algorithm	Performed terribly in correctness
AutoTutor [50]	2004	Newtonian physics, Computer literacy	Latent Semantic Analysis(LSA), Augmented finite state transition network	Personalisation, passed the Bystander Turing Test
Song et al. [51]	2017	Online courses	Rule-based, Node.js, Angular, MongoDB, Typescript	Learning reflection can be effective in online learning

Table 2.3: Educational Chatbot State of the Art Summary

The table 2.3 shows a summary of all the educational chatbots reviewed in this section. The name of the bot, year of paper, subject that it taught, technology it used and remarks (good or bad) about the chatbot are all seen in the heading row.

2.4 Chatbot Development and Concepts

As mentioned in a recent overview of chatbot technology [52], there are a few key concepts related to conversational agent technology. These will be explained in the section.

Some of the earliest chatbots such as Eliza used an approach called *pattern matching*. To put it simply, pattern matching algorithms analyse a user input and compare it to information in the chatbot knowledge base. Some chatbots used these algorithms to classify text so that they could provide a response using a template that was paired with the pattern recognised. Eliza used simpler pattern matching where the chatbots that succeeded Eliza used more complex pattern matching techniques [53].

The chatbot ALICE used a more advanced pattern matching method called *Artificial Intelligence Markup Language (AIML)*. AIML was developed between 1995-2000 and ALICE was the first chatbot to use it [54]. It was developed based on pattern matching and pattern recognition techniques. AIML is an XML based language that used tags which contains a pattern (user input) and a template (agent responses). *Latent Semantic Analysis (LSA)* is typically used alongside AIML to find similarities between vectorised words. LSA was used in the previously mentioned AutoTutor [50].

Chatscript was a software tool used in a study by M. Coronado et al. [40] that integrated both a NLP engine and a dialog management system, with the aim of enabling personalised conversations. A text file of scripts that containing rules is stored and used. Chatscript also has the ability to integrate machine learning tools so that conversation flows.

Domain Knowledge is a key concept in chatbots. Domain knowledge contains information about the domain in which the chatbot is in. These are essential for domain specific chatbots. Domain knowledge is used in training the chatbot and when selecting responses for the agent.

Artificial Neural Network Models can be used to build a chatbot. These employ machine learning algorithms inspired by the structure and function of the human brain. Networks are composed of interconnected nodes, or "neurons," that process information given as an input and produce an output. They can be used in chatbots to improve language understanding and generation.

2.5 Rasa

Rasa is an open-source framework for developing speech and text based chatbots and was used in this project to develop a chatbot. This framework was chosen over others, such as dialogue flow [55]. Rasa has some advantages such as:

- Flexibility to manage context in complicated conversations.
- Rasa has the ability to produce better responses in natural language compared to other platforms.
- Rasa offers great resources such as documentation and YouTube tutorials.
- Good deployment options, such as text or speech chatbots to websites or apps such as Facebook messenger.
- Customisable modules such as NLU, deployment and more. Dialogue flow, for example, does not offer such freedoms.

2.5.1 Rasa Architecture

Figure 2.2 is an overview of the Rasa architecture. A Rasa **Agent** is at the core of this architecture, this is the conversational agent or chatbot. The bot uses the **NLU Pipeline** to perform intent classification, entity extraction, and response retrieval. The **Dialogue Polices** help the system understand context so that the next action can be picked. The Rasa **Agent** accesses the **Action Server** to execute custom actions. A **Tracker Store** keeps track of the conversation. The **Lock Store** displayed in the diagram is used to ensure conversations are processed in the correct order in the event that there are multiple Rasa servers. Finally, models and training data are stored in the computers **File System**.

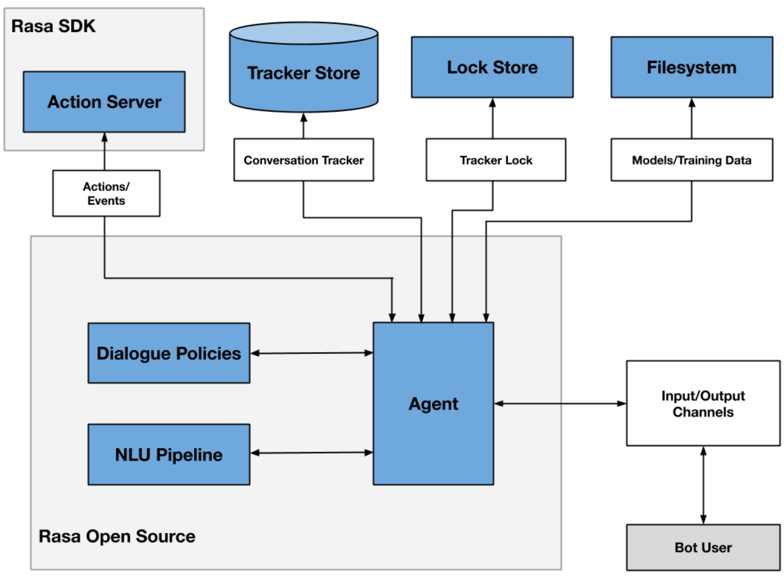


Figure 2.2: Rasa Architecture [1]

2.5.2 Training

Rasa uses Yet Another Markup Language (YAML) format to write training data. The training data is split into many YAML files and components. The training data is structured by a combination of NLU, stories and rules files.

NLU

In Rasa this file contains possible user input so that the system can be trained on what utterances to expect. Each possible input is listed with examples of similar text and grouped together with an intent. An intent tends be a word or phrase that describes the intention of a user input. Entities are information that is contained in an input when intents are recognised. An entity represents a specific piece of information that the user mentions in their message. Entities are normally extracted from user utterances so that they can be used as values to fill slots.

Stories

The dialogue management is trained using Rasa stories. Dialogue management helps the system understand context in a conversation. Stories help generalise conversation between a user and a system. The system learns different conversation paths by training stories. In Rasa, stories are written similar to a conversation between a human and the chatbot, only the user inputs are swapped out for intents and entities. The conversational agents replies are represented as actions.

Rules

Another way to train the dialogue management model in a Rasa chatbot is rules. Rules are segments of conversation that can happen at any point in the interaction. They always follow the same series of instructions starting with an intent that triggers the rule, followed by a bot action. For example, a user can quit the session at any point in the conversation when intent 'quit' is recognised.

2.5.3 Domain

The domain links everything together as it defines intents, entities, slots, responses, forms, and actions that are used in the system. All intents that are used in the NLU file and training data (stories and rules) are included in the domain. All entities to be extracted by the entity extractor from the NLU are documented here too. Slots in Rasa are the chatbots memory. In the domain, they are entered under the slots heading and given a name along with mapping information and type of slot. During a session, the chatbot uses slots to map entity values to slots so that they can be used in bot response/action or stored in a database. Chatbot responses are also defined in the domain. These are simple actions that send a reply to a user without the use of custom actions. They can contain entities, which the bot will fill with slot values during a session. For example, if the slot "entity:name, value: Declan" is filled at the start of a session when a user tells the bot her name is Declan, then at the end of the session the bot can utter the response "Bye {name}" and the user will see "Bye Declan".

2.5.4 Actions

Actions are tasks a chatbot performs. They could include querying a database, replying to a user, querying a database. Custom actions are a type of action that allows the bot to execute custom code. Both actions and custom actions are listed in domain.

2.5.5 Pipeline

The NLU pipeline processes input and handles intent classification, response selection and entity extraction. Pre-trained models can be loaded into the pipeline, such as SpacyNLP, different language models are used for different applications. A pipeline will include tokenizers which segment text into tokens. There is a range of available tokenizers in Rasa, depending on how text should be split. Featurizers are responsible for producing features that can be used by the system. Also included in the pipeline are intent classifiers and entity extractors. Intent classifiers allocate an intent (from domain) to user utterance and entity extractors collect entities from a user's message. Selectors are used to pick a reply from many possible bot responses.

3 Design

This chapter explains the design and overall architecture of the system. Design choices and technology are described in this chapter, as well as the data design and collection.

3.1 Case Study Outline

As mentioned previously, this dissertation takes a case study approach, which is a method used to gain insights into a complex issue in its real-life context [14]. Using a case study approach enables critical events, interventions, policy developments and program enhancements to be thoroughly investigated in a real-life context. In the context of this study, a personalised educational chatbot is developed and studied to gain an insight into personalisation in chatbots.

The second objective of this study is to develop a personalised conversational agent that can guide and tutor students through SQL questions, that behaves, acts and speaks like a teaching assistant. To achieve this, a chatbot is trained using specific training data. The training data is generated by the study, which includes structured SQL questions, common student queries about SQL, responses to these queries, and test data.

3.2 Design and Technology Choices

Due to the complexity and design of this case study a number of different technologies were used. The table 3.1 shows an overview of the technologies used for this study.

Rasa Open Source is a framework used to build, develop, train and test chatbot systems locally. Other frameworks were investigated such as Google's Dialogue flow, however Rasa was chosen for this study as it has flexibility to manage context in complicated conversations as well as better responses in natural language, good deployment options for chatbots, better customisation, and Rasa is open source and free.

In order to develop custom actions and database access code Python was required. This was all done using Pycharm IDE which has a nice user interface and also has a feature to

Purpose	Technology/Software
Chatbot Development Platform	Rasa Open Source
Cutsom Actions & Database Access	Python
Python IDE	PyCharm
Database	MySQL
Virtual Machine	Google Cloud Platform
Webserver	Nginx
Online SQL Query Console	Fiddle
User Interface	Botfront/Rasa Webchat

Table 3.1: Technology Choices

connect to a MySQL server and edit MySQL databases within PyCharm. To make design and development efficient and easy all development was done in the same interface, PyCharm.

MySQL is a widely used open-source relational database management system (RDBMS) that offers many advantages for storing and managing data. MySQL is compatible with Rasa and Python and it is usable within PyCharm. MySQL can also handle substantial amounts of data and high traffic volumes, it offers robust security features, and it is open-source and free to use i.e. it is reliable, scalable, flexible and secure [56]. For this study, a MySQL server was downloaded locally and connected to using a username and password.

Google Cloud Platform (GPC) was used for a virtual machine which runs a Nginx webserver which hosted the website for the Rasa chatbot. GPC was chosen due to the fact that it has a free trial of $300 in credits and was also easy to use with the ability to access the virtual machine in the browser. Once a google account was logged in, an Ubuntu 20 virtual machine was set up with 10Gb of storage.

Nginx was selected as the preferred web server due to its superior loading speed and performance compared to many other web servers [57]. Additionally, it is known for its efficient use of hardware resources, making it a practical and cost-effective choice for hosting websites or applications. It was installed in the GPC virtual machine using the command line.

SQL Fiddle was used as an online SQL query console so that users (participants) could test queries out before giving answers to the chatbot. SQL Fiddle allows a database to be created and queried. The link to a database and query window could be shared and a user could click the link to access the database and query console.

Botfront or Rasa webchat was implemented as a user interface in a browser. This was used both in local mode and on the website. A user interface is important as it makes the user experience of the chatbot more enjoyable. If a user interface is not used then the user would have to talk to the chatbot on the command line or terminal. This can be more intimidating for those who are not familiar with using the command line. A conversation feels more natural in a user interface with the use of an icon and typing graphics.

Just to note that GPC, Nginx and Rasa webchat were needed due to the removal of free Rasa X in summer 2022.

3.3 Overall Architecture

An overview of the system architecture is captured in the diagram 3.1. The user has two windows open on their device, one being the online Fiddle SQL Query console with the practice database loaded into it. The second window a user has open is the chatbot user interface. This is displayed on the website which is running the rasa agent and server simultaneously. The rasa agent is using a rasa action server to access and execute custom actions, where the database.py file is accessed, which can query the systems database. This is all explained in detail in chapter 4.

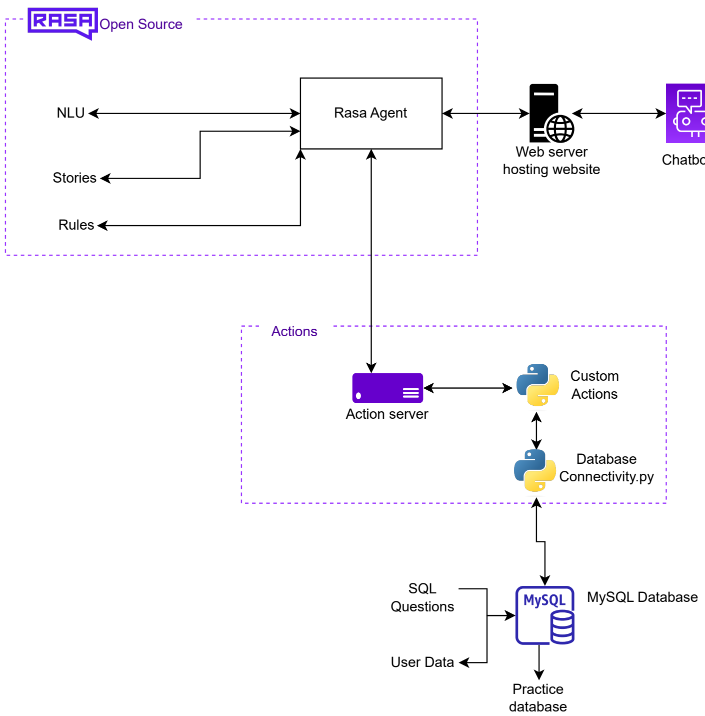


Figure 3.1: System Architecture Overview

3.4 Data Design and Collection

For this study training data is generated both manually and through the use of Rasa interactive training. There is a total 22 intent classes, 11 entity values, 11 slots, 303 user examples, 31 chatbot responses, 9 custom actions, 8 MySQL database tables, 10 python database functions, and 1 form.

3.4.1 NLU Data

NLU data is generated to represent possible user input so that the chatbot can recognise what the user is saying (classifying intent) and if it needs to extract any values (entities) from the input. Listing 1 shows an example of NLU training data in nlu.yml that classifies the intent "general_help". Each intent contains example user input. In this study, most intents and examples were manually entered, some are entered through the use of Rasa interactive training. Listing 1 shows examples of how a user asks for help. The bot was trained on examples like these, so that it can understand a wide range of phrases that indicate a user needs help. The likelihood of correctly classifying the intent increases with the number of examples in the NLU.

```
- intent: general_help
  examples: |
    - how do I start?
    - can I have a hint?
```

Listing 1: Example Intent Data

3.4.2 Responses

Responses refer to the messages that a chatbot sends back to the user. Typically, responses consist of text, although it is possible for them to contain additional content, such as images or buttons. Potential responses for the chatbot are created in domain.yml with some responses containing response variations. This involves writing multiple responses for a bot to chose from when replying. Thus, adding variety to the conversation and making the bot more interesting. Responses are designed to make the conversation seem more personal and fluid.

3.4.3 Stories

Training data called stories are used to train a chatbots dialogue management model so that conversation paths can be generalised. This allows the model to learn how to handle a wide range of user inputs. Stories can be created manually or by using Rasas interactive training functionality. Stories can become complex as they involve labeling intents, actions, responses, forms, entities and slots. Due to the complicated nature of stories, interactive training is used to generate stories data. Rasa's interactive training involves having a conversation with the chatbot while also manually classifying intents, actions, responses and entities so that Rasa can format everything into training data such as stories. There are two main paths a user could take, that is, if a user is a first time user or if they are a returning user. This can be seen in section 3.7.

3.4.4 Rules

Rules are another type of training data (as well as stories) that train a chatbot's dialogue management model. They are much shorter than stories and always follow the same path. Rules cannot generalise the conversation path so in order to maintain natural conversation it is not recommended to overuse them.

3.5 Database Design

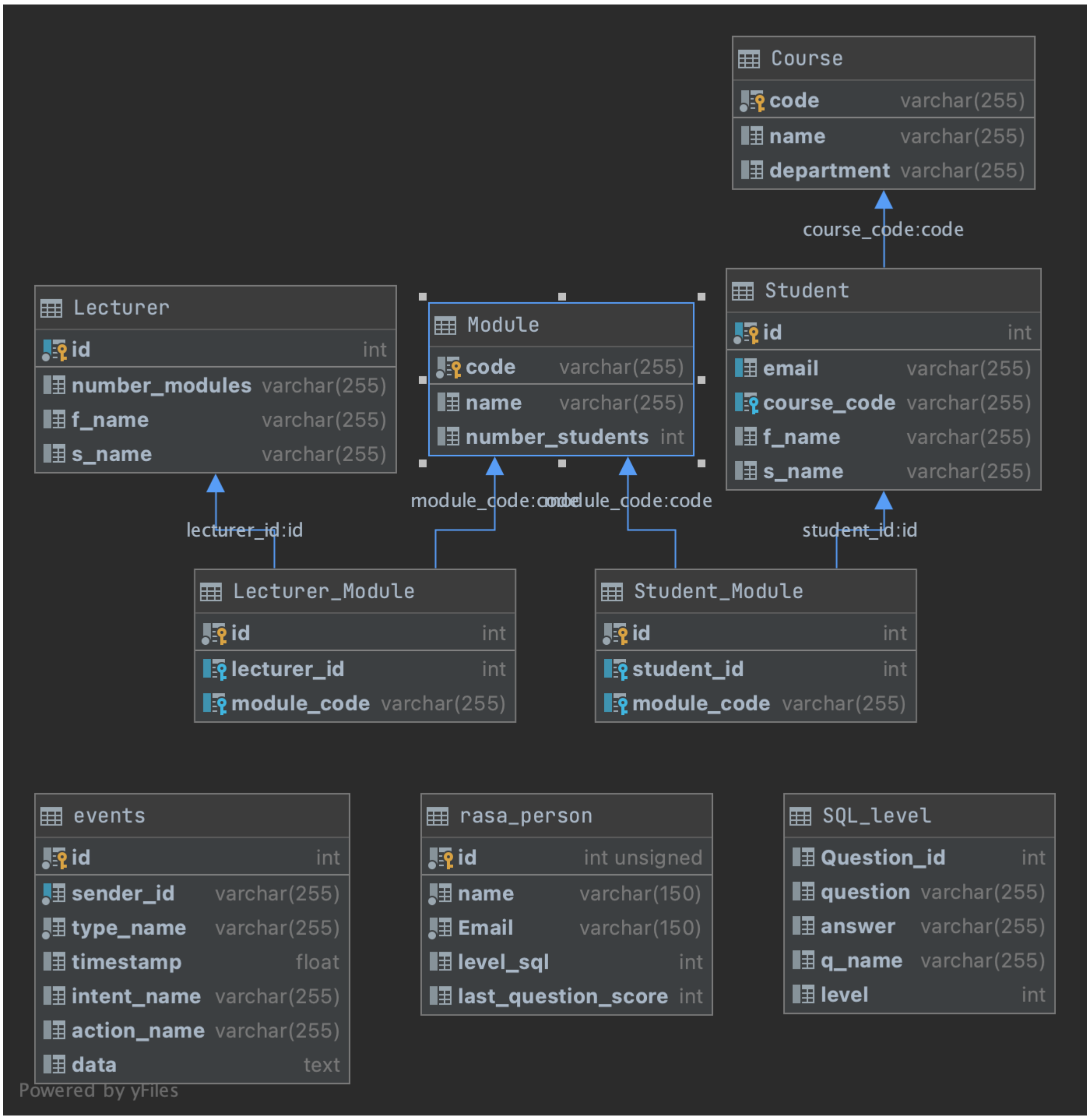


Figure 3.2: Database Diagram

Figure 3.2 shows MySQL tables, entities and relationships. Just to note that this is the developers view of the database, a user is given a copy of the mock (practice) database which includes tables "Lecturer, Module, Lecturer_Module, Student, Student_Module" and "Course". The remaining tables "rasa_person, SQL_level" and "events" are for researchers only. The reason the user is given a copy of the mock database (in Fiddle) and not access to the actual database server is due to the fact that user data could be accessed if security is bypassed and that users could unintentionally (or intentionally) alter tables.

The practice database consists of a real-life example, college modules, lecturers and students. The database is designed so that users can easily understand it as an over complicated database could confused users with lower abilities. While the database is not over complicated it allows for the construction of challenging questions due to the relationships and foreign keys present, depicted in 3.2 by the blue keys and arrows to other tables. The data in tables is filled with made up names, courses and emails etc. It is purely to give example context to the user so that they can grasp SQL concepts.

The "rasa_person" table contains information about users including their id, name, email, SQL ability and the number of questions that they got right on the topic they were last on. This table is designed so that each id given to a user is unique and cannot be null. Level of SQL corresponds to the "SQL_level" table, the reason it is not shown as a foreign key is due to the fact that SQL level is not unique in the "SQL_level" table. The choice to store sql_level over the Question_id in "rasa_person" is on purpose, when using sql_level a user can browse the questions freely and come back and answer similar questions of the same level instead of with Question_id, moving strictly through questions. It gives more flexibility and freedom to a user and also gives defined bounds of the level of difficulty they are at.

Table "SQL_level" stores all SQL questions that a user could be asked. The table consists of 5 columns, Questions_id is a unique question number, question is the text of the question the chatbot gives to the user, answer is the answer, q_name is a template for the topic of the question and level is the SQL difficulty of the question. Details on how these questions are designed are in section 3.6.1.

Finally, "events" is the Rasa tracker store, where details about interactions and Rasa decisions are stored for each conversation.

3.6 SQL Question Design

3.6.1 SQL Structured Exercises

In this section the design of the SQL questions is explained. Structured SQL exercises are crucial in creating context awareness for the bot and helping the student learn effectively. The SQL exercises are structured sequentially from easy to hard, following the W3Schools [58] website course structure. Some research has found that structure in teaching improves students' engagement [59, 60]. Therefore, designing and giving students structured questions in increasing difficulty would be more effective than giving them random questions.

Questions are modelled for 3 domains[1] in SQL where the chatbot's abilities can easily be expanded to. Table 3.2 gives an example of how the questions are designed and structured for domain one. Domain one contains questions about one table only using SELECT. This means no complicated JOIN statements are included but gets the student to perfect basic types of queries first. Even within domain one the questions get increasingly more difficult. The table shows questions from levels 1-6 in increasing difficulty only however a full table and template of questions can be seen in the appendix A1.

Level	Question	Example answer
1	Select the module code and name from the Module table	`SELECT code, name FROM Module`
2	Select all module codes in the Module table	`SELECT * FROM Module`
3	Select all distinct module codes in the Module table	`SELECT DISTINCT code FROM Module`
4	Select the module names where the module code is 'c2'	`SELECT name FROM Module WHERE code = 'c2';`
5	Select the name of the course that has the code 'EE' and department 'Engineering'	`SELECT name FROM Course WHERE code = 'EE' AND department = 'Engineering';`
6	Select the all values of the courses that has the department 'Engineering' that has code 'EE' or 'ME'	`SELECT * FROM Course WHERE department = 'Engineering' AND ( code = 'EE' OR code = 'ME');`

Table 3.2: Domain 1 Example Question Design

This study only uses questions from domain one, however more domains are modelled for potential expansion of the chatbot. Table 3.3 show example questions that would be found when a student moves to the second domain of questions. Domain two contains questions that involve the SELECT statement and two tables. As in domain one, the questions increase in difficulty, following a similar pattern.

[1] A domain in the context of this chatbot knowledge base is the common topic or theme (within SQL) that questions are based on. Domain 1: SQL SELECT with one table, Domain 2: SQL SELECT with two tables...

Level	Question	Example answer
1	Select the module code and name from the Module table and students first name from Student	`SELECT Module.code, Student.f_name FROM Module, Student;`
2	Select all values in Lecturer and Module	`SELECT * FROM Lecturer, Module;`
3	Select distinct values of department in Course and first names of students (make sure there are no duplicates)	`SELECT DISTINCT Course.department, Student.f_name FROM Course, Student`
4	Select student first name and the Course they are doing	`SELECT Student.f_name,Course.name FROM Student JOIN Course ON Student.course_code = Course.code`

Table 3.3: Domain 2 Example Question Design

Table 3.4 shows a template for questions that could be asked in a domain where three tables are involved using the SELECT and JOIN. As stated previously, this study only incorporates questions from domain one, however domains could be added easily. Other domains could include using SQL INSERT, table views, triggers, or how to add security permissions.

Level	Description	Outline
1	JOIN	`SELECT table1.col1, table2.col2 FROM ((table3 INNER JOIN table1 ON table1.col3 = table3.col4) INNER JOIN table2 ON table2.col5 = table3.col6);`

Table 3.4: Domain 3 Template Question Design

3.6.2 SQL Questions

One feature of the designed system is the ability to ask the chatbot questions during a session. In a real-life teaching scenario, students have questions about the topic they are on as they are doing a problem. Sometimes students have random unrelated questions too. This system aims to create a personalised human-like tutoring experience by understanding context and answering a user's question based on the context of the question. It also does this by answering non-relevant SQL questions which a user may have out of interest. Hence, questions were split into 2 categories general(contextual) or ChitChat(non-contextual). From table 3.5 examples of user input that needs context to answer and that does not.

Rasa has a special response action for 'FAQ' chatbots. It is a fast and effective way to provide responses to non-contextual questions, detail on this in subsection 4.1.6. In the case of questions where the user is specific or a question is asked about a topic that the user is not currently on, context is not needed for the chatbot to respond and the chatbot can answer easily. Whereas, when a user asks a general question such as "I need help" the bot needs to perform steps, such as executing custom actions, in order to find out what topic the user is on. For each topic, non-contextual questions and answers are gathered from websites [61–63]. Variations of the same question are written into nlu.yml and answers are written to domain.yml.

Example Text	Question Type
How do I write a WHERE clause?	Non Contextual
can I have help?	Contextual
what is the syntax?	Contextual
what does SELECT do?	Non Contextual

Table 3.5: SQL Questions

3.7 Conversation Flow

The figure 3.3 shows a general template for 2 basic chat paths. The two main paths are a new user and a returning user. The conversation is designed so that it feels personal and natural. To do this, returning users are remembered, names are used and the bot keeps track of where they last left off. This gives a personalised learning experience.

When a new user starts a session the chatbot introduces itself and prompts them for their name and email, responding with a unique ID which they must remember. The ID is used for the next time they want to interact the system. The chatbot then gives them the first exercise to work through and once they get that correct (or skip it) they move onto the main basic chat path.

A returning user enters their ID and the chatbot gives them a summary about the last time they interacted. The user moves onto the main basic path.

The main basic path starts by asking the user if they want to stay on the current topic or move onto a more difficult one. This feature gives the user control of their learning and makes the experience feel more personalised. When the user decides what they want to do, the chatbot gives them a question based on their answer. Once the user gets the question correct or skips it, the chat returns to ask the user if they want to move on or not.

This is just a template of a basic conversation flow, the system is designed to handle questions and help the student through the question that they are doing.

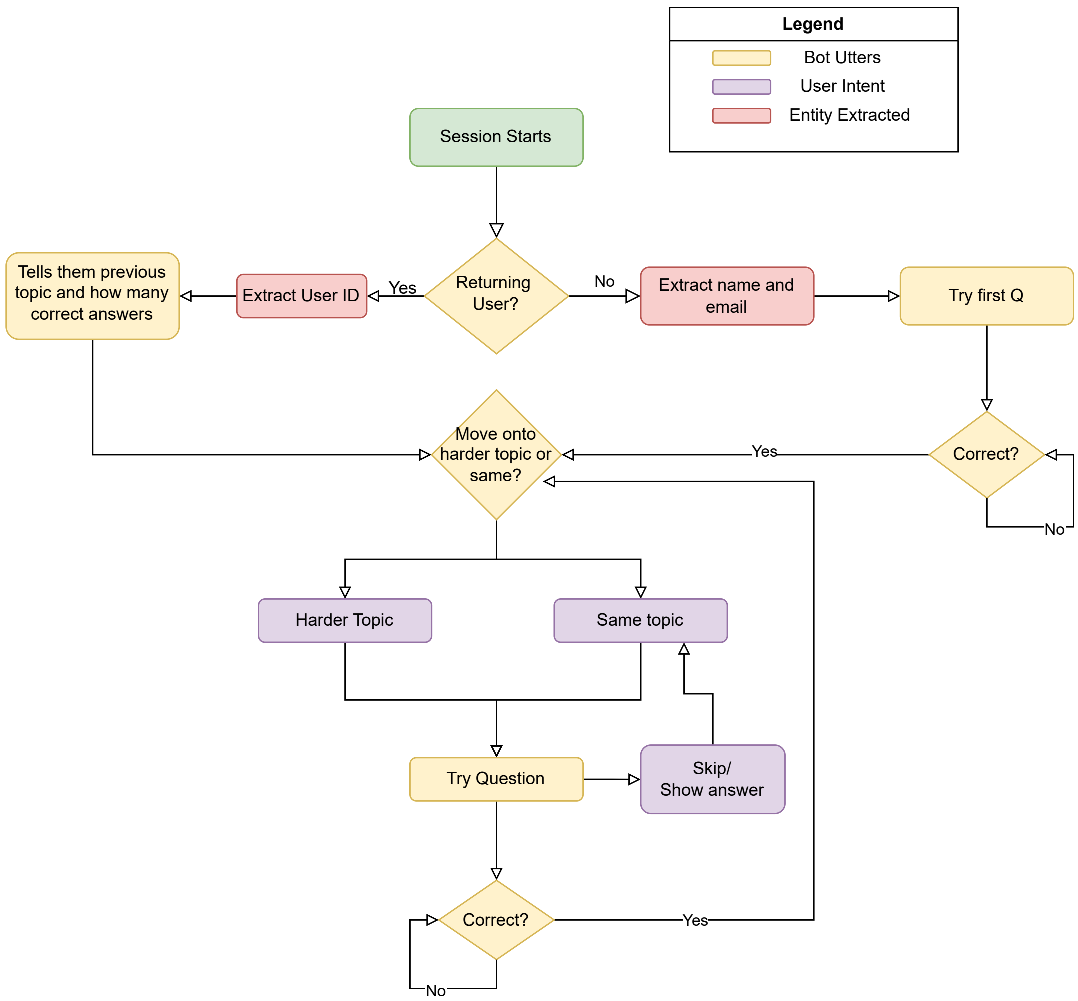


Figure 3.3: Overview - Template Chat Flow

3.8 Feedback to User

One indicator of good learning is learners getting feedback so that they can view their progress and learn from their mistakes [64]. Hence, in this chatbot system it is important to have feedback. The system in this study provides feedback to a student when answering SQL questions. When a user gets a question wrong, the system fires the user's answer at the mock database to see if there is any error messages, the chatbot replies by telling them the answer is wrong and prints the error message returned from the database. The user is then encouraged to try the question again. This feedback can be useful as it can tell the user if syntax errors or operational errors were made.

Feedback is also given after every answer that produces the correct results set from the mock database but was not the answer the chatbot wanted. The chatbot tells them the query is correct but not what it was looking for and then gives them the answer for the correct query.

Feedback is given about the number of questions a user has gotten correct on a topic. This is useful for a student to gauge how well they are doing. Finally, feedback is given when a user quits a session. The chatbots gives a summary of the lesson and the topics that were covered by the student.

4 Implementation

In this section, the implementation of the chatbot and other components are explained. Code examples and conversation examples are presented in this section too.

4.1 Chatbot Agent Implementation

As mentioned in previous sections, Rasa Open Source was used to develop the chatbot. Rasa uses YAML format to manage training data such as NLU, rules and stories. Python was used to manage custom actions.

4.1.1 Intents

In Rasa, an intent represents the intention behind a user's message. An intent helps the chatbot understand and classify the user's purpose for sending the message. Intents are found in nlu.yml, they are generalised by one word and contain examples of user input that would classify as an intent.

For example, in listing 2 the user can send any message in the examples and the intent of the message would be "repeat_question". If a user's message was "Can you repeat the question?", the intent of the message would be recognised as "repeat_question". Additionally, the chatbot does not need the exact example text that is in the NLU to classify an intent. The chatbot is trained to learn examples and can classify user input as an intent based on partial matches with NLU examples. For instance, "repeat the problem" would classify as intent "repeat_question" even though the exact text is not in the example.

By recognising the intent of a user's input, the bot can take the appropriate action and provide a relevant response. Intent recognition is a key component of natural language understanding, which is critical for creating effective conversational AI.

4.1.2 Entities and Slots

Sometimes more information is contained in an input when intents are recognised in Rasa. This extra information can be defined as words called entities. An entity represents a specific

```
- intent: repeat_question
  examples: |
    - What was the question again?
    - I cant remember the question?
    - Can you repeat the question?
    - can you tell me the exercise again?
    - I forgot what problem we are doing
    - can you repeat the exercise.
```

Listing 2: Example Intent

piece of information that the user mentions in their message. Entities are used to extract relevant information from the user's message, which is then used to understand the user's intent and take the appropriate action or give the correct response.

Listing 3 shows an example of an intent containing entities. The words in the square brackets are entity values and the round brackets define what entity those values belongs to. In this case, [John] is the entity value and (name) is the entity. This is used when a new user is entering their details for the first time.

```
- intent: supply_contact_info
  examples: |
    - My name is [John](name). email's [john@email.com](email)
    - name: [David](name) email: [david@email.com](email)
    - Yeah sure. I'm [Barbara](name). My email is [barbara@email.com](email)
    - [Susan](name), [susan@email.com](email)
```

Listing 3: Example Entity

Slots are the chatbot's memory. They store and keep track of information or entities and their values, during a session. Values can be retrieved from slots too. Listing 6 is an example of how slots with type "text" in listing 4 or type "categorical" in listing 5 are written in domain.yml. These show how entities are extracted from intents and assigned an entity name, the value is then stored and can be extracted when needed. Most slots in this system are of type "text". The second example listing 5 shows slots of type categorical. Using categorical slots in this example involves classifying a student's answer as being either right or wrong so that the conversation can move on (when correct) or so that it stays listening for a right answer or user intent to skip. The parameter "influence_conversation: true" is set so that a correct or incorrect answer can influence a conversation.

```yaml
slots:
  name:
    type: text
    mappings:
    - type: from_entity
      entity: name
  email:
    type: text
    mappings:
    - type: from_entity
      entity: email
```

Listing 4: Example Slots - Type Text

```yaml
reply:
  type: categorical
  influence_conversation: true
  values:
  - Correct
  - Incorrect
  mappings:
  - type: from_entity
    entity: reply
```

Listing 5: Example Slots - Categorical Text

Listing 6: Slot Examples

4.1.3 Responses

A response is a message that the chatbot sends back to the user in response to input or to ask for more information. Responses are defined in the domain file using the "responses" key. Responses do not involve running any custom code or actions.

In listing 7 there are examples of responses that the chatbot can send the user. The first response "utter_greet1" from lines 2-5 is the response the bot gives a user to start the conversation. This is a simple response with no entities. The "|-" in the text is to allow new lines to be represented. In the second response, "utter_trythis" from lines 6-10, the chatbot is telling the user to try the new question that it has retrieved for the user (using a custom action which will be talked about in section 4.1.4). The slot is filled with the entity "new_exercise" using curly brackets {} and displayed for the user. In the last example "utter_show_answer" multiple responses are seen on each line. This is done in many responses for this system to add variety so the user does not get bored reading the same repeated text.

```yaml
1 responses:
2   utter_greet1:
3   - text: |-
4         Hey my name is Seequill SQL Tutor! Have you talked to me before?
5              If yes write your id, if no type 'n'
6   utter_trythis:
7   - text: |-
8       Okay try this:
9        {new_exercise}
```

```
      I will be here if you have any questions, type them :)
  utter_show_answer:
  - text: Remember, you can request the solution if you are fed up!
  - text: Ask to see the answer if you have had enough.
  - text: I can show you the solution to this question if you are fed up.

```

Listing 7: Example Responses

4.1.4 Actions

Custom actions in Rasa are Python classes that define the behaviour of the bot in response to user input or system events. These are events that cannot be handled through regular dialogue flow. Custom actions allow custom code to be written to perform complex actions such as accessing external APIs, querying databases, and more.

Custom actions can be triggered by user input or system events that match a certain condition defined in the story or rule. In this study, custom actions were mainly triggered to access the user or SQL questions tables in the database.

Listing 8 shows a template of what most of the action code looks like in this system. Each action has a class name taking "Action" as argument as it inherits from the Action class in Rasa. Each custom action has a method "name" which returns the unique name of the action, in the example the name of the action is "action_example_action". Next, a method "run" is also required with every custom action and takes "dispatcher", "tracker", "domain" as arguments. The "run" method contains the code that the custom action performs when it is triggered.

```python
from database_con import dataBaseFunction
class ActionExampleAction(Action):
    def name(self):
        return "action_example_action"

    def run(self, dispatcher, tracker, domain):
        x, y, z = dataBaseFunction(tracker.get_slot("input"), dispatcher)
        return [SlotSet("x_slot", x), SlotSet("y_slot", y), SlotSet("z_slot", z)]
```

Listing 8: Template Action Code

The "dataBaseFunction" in listing 8 is a function in database_con.py which allows the chatbot to interact with the MySQL database (The details of the database functions will be explained in 4.2.). The function passes two arguments, the value in the slot "input" and the dispatcher. The example function seen in the template can collect current slots using "tracker.get_slot("input")". Values are returned to the custom action using the database function. These values fill slots using return "SlotSet("x_slot", x)" at the end of the "run" method.

Previous Session Information

Using the template above, the remainder of basic custom actions can be easily explained by pointing out what differs.

```python
def run(self, dispatcher, tracker, domain):
    exercise, level, n_correct_qs, name, exercise_id =
     ↪  dataGetPrevQ(tracker.get_slot("id"), dispatcher)
```

Listing 9: Action Code - Previous Session

Once a returning user enters their ID, the chatbot finds out who they are and where they last left off. In order for the chatbot to do this a custom action named "action_lastquestion" is triggered. In listing 9 the run method accesses the dataGetPrevQ. All details of database functions will be explained in 4.2. The only information that the chatbot has so far in the session is the users ID, hence this slot value is extracted and used as argument in the database function. Values for "exercise", "level", "n_correct_qs", "name", "exercise_id" are returned using the database function to query the database. These values fill the slots using "SlotSet()". Exercise is the last exercise tried by the user with "exercise_id" being its id , level is users SQL ability and "n_correct_qs" is the number of correct answers the user has.

Getting New Questions

Once a user decides on whether they want to move onto a harder topic or try a similar question the action "action_newquestion" is triggered. In listing 10 line 2 the action is collecting the last intent using "tracker.latest_message['intent'].get('name')" which is either intent "move_on" or "similar". Once this is gathered the database function is called. This action fills the slots for "new_exercise", "exercise_id", "level", "n_correct_qs". "new_exercise" is the new question that the user will be asked with "exercise_id" being its ID.

```python
1 def run(self, dispatcher, tracker, domain):
2     x = tracker.latest_message['intent'].get('name')
3     new_exercise, exercise_id, level, n_correct_qs = dataGetNewQ(x,
     ↪  tracker.get_slot("id"), ...)
```

Listing 10: Action Code - New Question

Checking Answers

The action called "action_checkanswer" is triggered when a user attempts to answer an exercise. Line 2 in listing 11 shows the custom action storing the most recent user input in the variable "message". This actions runs the database function dataCheckAnswer() which returns values for slots "reply", "n_correct_qs". "Reply" is a categorical slot with only two possible values: "Correct" or "Incorrect".

```
1 def run(self, dispatcher, tracker, domain):
2     message = tracker.latest_message.get('text')
3     reply, n_correct_qs = dataCheckAnswer(message,...)
```

Listing 11: Action Code - Checking answer

General Questions

The action "action_general_help" is triggered when a user is looking for help on a the question they are answering. Again the tracker on line 2 of listing 12 is able to extract the most recent user message and input it as argument into the database function "dataGeneralQuestion". This action fills the slot for reply to null so that the tracker does not get lost and can stay on the current exercise until the user answers and gets the exercise correct or not.

```
1 def run(self, dispatcher, tracker, domain):
2     message = tracker.latest_message.get('text')
3     reply = dataGeneralQuestion(message, tracker.get_slot("level"),
      ↪ dispatcher)
```

Listing 12: Action Code - General Question

Show Answer & Skipping Questions

A rule triggers the actions "action_show_answer" and "action_skip_question" one after the other. This is triggered when a user wants to skip the current question or simply see the solution of the current problem they are on. As seen in listing 13 the simple action triggers a database function.

```
def run(self, dispatcher, tracker, domain):
    dataShowAnswer(dispatcher, tracker.get_slot("exercise_id"))
```

Listing 13: Action Code - Show Answer

The next action "action_skip_question" is triggered in listing 14, after the answer to the current question is displayed with "action_show_answer" above . This is done by setting the intent to "similar" and calling the database function that gets another question. This, as in 10, sets the slots "new_exercise", "exercise_id", "level", "n_correct_qs".

```
def run(self, dispatcher, tracker, domain):
    x = 'similar'
    new_exercise, exercise_id, level, n_correct_qs  = dataGetNewQ(x,...)
```

Listing 14: Action Code - Skip Question

Quitting Session

When a user opts to quit a session with the chatbot the action "action_quit" is triggered. This runs the database function "dataUpdateOnQuit" so that all database values can be updated and so the user can receive a summary of the session.

```
def run(self, dispatcher, tracker, domain):
    dataUpdateOnQuit(tracker.get_slot("id"), dispatcher,...)
```

4.1.5 Forms

In Rasa, forms are a way to capture structured information from users. Forms allow a set of required slots that need to be filled by the user to be defined while guiding the conversation until all the required information has been collected. For example, in this study, a form is used to fill the slots 'email' and 'name' from a first-time user. Without these slots being filled the database would have null values and there would be errors. They help ensure that the bot collects all the required information in a structured way, reducing the risk of misunderstandings or errors in the conversation.

The form used in this project is called "person_form" which was the unique identifier of the form. A method called "required_slots" returns the required slots of the form, in the case of this study that was "name" and "email". Another method called "slot_mappings" maps the slots to an extracted entity and an intent. Finally, a "submit" method defines what the form has to do after the required slots are filled.

4.1.6 ChitChat

Rasa has a special response action for 'FAQ' chatbots. It is a fast and effective way to provide responses to non-contextual questions. In the case of questions where the user is specific or a question is asked about a topic that the user is not currently on, context is not needed for the chatbot to respond. All input classified as 'chit_chat' followed by a word to identify the question and answer as seen in listing 16. Here, user questions are classified with intent "chitchat/ask_SELECT_syntax", the bot response is seen as utter_chitchat/ask_SELECT_syntax.

```yaml
- intent: chitchat/ask_SELECT_syntax
  examples: |
    - how do I write a select statement?
    - what is the syntax for select?
    - how to I write select?
```

Listing 15: User - nlu.yml

```yaml
responses:
  utter_chitchat/ask_SELECT_syntax:
  - text: |-
      SELECT column1, column2
       FROM table_name;
```

Listing 16: Bot - domain.yml

Listing 17: Chitchat Examples

4.1.7 Pipelines and Policies

A pipeline in Rasa is a series of data processing components that are executed in sequence to extract relevant information from user input and understand them. The pipeline can include various NLU (Natural Language Understanding) and dialogue management components, such as tokenization, entity extraction, intent classification and dialogue management policies. Tokenizers are used in NLP so that sentences and text can be split into smaller pieces, called tokens, in order for them to be processed and understood. In this study, the white space tokenizer and Spacy tokenizer were used. Featurizers are responsible for producing numerical features that can be used by machine learning models. The featurizers used in this study were RegexFeaturizer, LexicalSyntacticFeaturizer, CountVectorsFeaturizer and SpacyFeaturizer. Intent classifiers are used when features have been made for all tokens and can classify intents. DIET classifier was used in this project. DIET was a good choice as it can handle intent classification as well as entity extraction. Other things such as FallbackClassifier can be defined in a Rasa pipeline. In this project, this was used to define the threshold and ambiguity in confidence for a repair statement to be triggered. A ResponseSelector for 'chitchat' was also defined in this systems pipeline. This was done so that chitchat responses could be used. These pipelines are found in config.yml.

In a conversation, a Rasa chatbot employs policies to determine the appropriate action to take at each stage. These policies can work together in combination and include both rule-based and machine-learning based policies. For each turn in conversation a next action is predicted with a confidence level based on the policies chosen in config.yml. In this project, MemoizationPolicy, RulePolicy and TEDPolicy were used.

4.2 Database Interactions

In this section the python functions that interact with the MySQL database are explained. To start, a template of how to execute SQL in python is shown in listing 18. The database is connected using a MySQL connector passing the arguments "host", "user", "passwd" and "database". These must all be filled when connecting to the database (for the purpose of security the values are replaced with asterisk in this example). A database cursor is instantiated and given a name. Next the SQL statement to be executed is initialised. A try-catch statement is used in the case where the SQL query fails. The cursor executes the statement and stores the results in a variable. This variable is parsed through in case there is more than one result returned.

```python
def examplePyQuery():
    mydb = mysql.connector.connect(host="****", user="****",
                                   passwd="***", database="****")
    mycursor = mydb.cursor()
    sql = "SELECT...." # some SQL query
    try:
        # Execute the SQL Query
        mycursor.execute(sql)
        results = mycursor.fetchall()
        for row in results:
            some_result = row[0]
            second_results = row[1]
    except:
        dispatcher.utter_message("Error : Unable to fetch data.")
```

Listing 18: Executing SQL Querys in Python

Previous Session Information

A function named "dataGetPrevQ" is called when the custom action "action_lastquestion" is triggered. The database function "dataGetPrevQ" takes arguments "id" and "dispatcher". In Rasa, the "dispatcher" is an object that is responsible for sending messages to the user. Passing the dispatcher as an argument to a custom action allows the system to perform more sophisticated actions and responses. Using the dispatcher is also the only way to send text responses to the user from Python helper functions. Dispatcher can act as the Python "print" function in this scenario.

This function works by using the only information available to the chatbot, the user ID. It's purpose is to tell the user what topic they last covered and how many questions they got correct on that topic. First, the "rasa_person" table or user table, is queried to retrieve the name, level of SQL of the user and the score that they last got. This result is parsed and results are stored in variables. Another SQL query is constructed to return the topic and question id of the previous session from the table "SQL_level". This is parsed using another for loop and stored in variables. The topic, level of SQL, last session score, user name and question id are all returned to the custom actions to fill slots. See listing 19 for reference, any "..." is code that was previously mentioned or not that important.

```
def dataGetPrevQ(id, dispatcher):
    ...
    sql = "SELECT level_sql,last_question_score,name FROM rasa_person WHERE id='%s'" % (id)
    try:
        ...
        for row in results:
            level_sql = row[0]
            last_question_score = row[1]
            name = row[2]
            sql2 = "SELECT q_name,Question_id FROM SQL_level WHERE level='%s'" % (level_sql)
            # execute nested SQL query
            mycursor.execute(sql2)
            results2 = mycursor.fetchall()
            for rows in results2:
                q_name = rows[0]
                qid = rows[1]
                return q_name, level_sql, last_question_score, name, qid
```

Listing 19: Python Code - Previous Session

Getting New Questions

When a user is asked if they want to move to a harder topic or want to stay on the current one their reply classifies as either intent "move_on" or "similar". Once this is recognised, a custom action "action_newquestion" is triggered where the database function "dataGetNewQ" is ran. A new question is retrieved from the database by using the intent. It takes 6 arguments one of which is "name" which is the user intent.

To explain in further detail, the function checks what the value of "name" or the intent is. Depending on this value the function increases the exercise id or the level of SQL by one. Increasing the exercise means the topic stays the same and the next question on that topic is picked. This in combination with the level of SQL is used to query the database and return values for a new question and question id. Increasing the level of SQL by one means that the next (more difficult) topic is picked. Using the new level calculated by the function, a new question and question id is selected from the database.

See listing 20 for reference, any "..." is code that was previously mentioned or not that important.

```python
def dataGetNewQ(name, id, dispatcher, level_sql, exercise_id, n_correct_qs):
    ...
    if name == 'similar':
        exercise_id = int(exercise_id) + 1
        sql2 = "SELECT question,Question_id FROM SQL_level WHERE level='%s' AND Question_id ='%s'" % ( level_sql, exercise_id)
        try:
            ...
    if name == 'move_on':
        n = 0 # number correct Qs reset
        level_sql = level_sql + 1
        sql2 = "SELECT question,Question_id FROM SQL_level WHERE level='%s'" % (level_sql)
        try:
            ...
```

Listing 20: Python Code - New Question

Checking Answers

Checking a user's answer to a question proved to be a complicated task. Sometimes in SQL there is more than one 'correct' answer to a query. Some queries are different but can produce the same results sets. This happens for example using "SELECT *" and just "SELECT col_names". For this study, a user can be one of three things: be completely correct, have the correct results set but not the intended answer (nearly correct) and they can also be wrong. For a user to be completely correct their answer must match the answer in the MySQL database. For a user to be nearly correct, the results set of their answer must match the results set of the actual answer (in the MySQL database). To be incorrect a user's answer does not match the database answer, their results set does not match the answer results set or an error was thrown from their executing their query.

To check if a student's answer is correct the system queries the database for the "answer" to the exercise that the student attempted by using "exercise_id" to look it up in the "SQL_level" table. Once this is matched to the student's answer the student is said to be correct. Thus, the number of correct questions on the topic is increased by one and the reply "Correct" is returned with the new number of correct answers so that the slots can be filled by these values in the custom action.

As seen in listing 21 a user gets a question incorrect when the query they entered throws an error when fired at the mock database. This is done by executing the user query and using try-except to catch the error and print it out as feedback to the user. This is an effective form of feedback as the user can see why their query did not work.

```
# in dataCheckAnswer
else:
    sql2 = user_answer
    try:
        mycursor.execute(sql2)
    except mysql.connector.Error as err:
        dispatcher.utter_message("Hmm looks like something went wrong,
        ↪  heres some feedback:\n")
        dispatcher.utter_message(f"Message: {err.msg} \n")
        dispatcher.utter_message(response="utter_show_answer")
        reply2 = "Incorrect"
        return reply2, n_correct_qs
```

Listing 21: Incorrect Query and Feedback

A user technically gets a question correct when the user's answer provides the same results set as the correct answer's results set. The system executes the user's answer and collects the results in a variable. The system also executes the correct query that was queried from the system database. If the length of the results set is 1 and if the users result is equal to the correct result then it can be said that the user collected the correct results, however, used a different query. The user is then shown the correct query and awarded a correct answer.

The process is slightly different when the results set is more than one result because the two sets could be the same values in different orders. The function parses through each list of results and compares them element-wise. If they are all the same, the user receives a message that the results are the same but that the answer is not quite right. The answer is then displayed so that the user can learn this for the next question. If some elements do not match in either results set then the user is told they are incorrect and to try again.

General Questions

When a user asks a general question to the chatbot without giving context the custom action "action_general_help" is triggered. This runs the function "dataGeneralQuestion" with arguments for the most recent user input (the question the user has,) the level of the user and he dispatcher. The function "dataGeneralQuestion", first puts the question in lower case. Then there are 3 conditional statements, if 'syntax' is in the question, if 'theory' is in the question and if there are no key words in the question. If any condition is true, then the function checks the SQL level of the user (telling the system what topic the user is on.)

Once the level is known the system can give answers about the syntax, theory or general help on that topic. Syntax and theory are self-explanatory but general help entails a combination of the two. The chatbot gives an introduction to the topic and a template of the syntax that can be used in that type of question.

```
def dataGeneralQuestion(message, level, dispatcher):
    msg = message.lower()
    if 'syntax' in msg:
        if level == 1:
            dispatcher.utter_message(
                f" Remember you learned about the SELECT statement? \n
                ↪  Try this syntax in your exercise: \n SELECT column1,
                ↪  column2  \n FROM table_name;")
```

Listing 22: General Question Code Example

Skipping Questions

When a user wants to either skip a question or see the answer, two custom actions are triggered. The first one "action_show_answer" runs the database function "dataShowAnswer". All this function does is collect the answer for the current question using an SQL query and prints it on the screen for the user. The next action that is triggered is "action_skip_question" which sets the user intent to "similar" and calls the function "dataGetNewQ", which was explained previously. This selects an SQL exercise that is on the same topic that the user is currently on. The new exercise is given to the user and the session continues on.

Quitting Session

A user can ask the chatbot to end the session. This triggers the custom action "action_quit" which runs the database function "dataUpdateOnQuit". This function takes "id", "level", "dispatcher", "name", "n_correct_qs" as input. The purpose of this action and function is to update the user table in the database so that it reflects a user's progress in a session. Another reason is to provide the user with a summary of the lesson that tells them what they have covered. To create a lesson summary, the level of SQL that the user had at the start of the session is found by querying the user table. This is then compared with the current slot value of the level of SQL to find out how many topics were covered. To display a summary, the question topics are queried using a for loop from the level of SQL at the start of the lesson to the level of SQL at the end of the session as seen in listing 23.

```
# in dataUpdateOnQuit
for row in results2:
```

```
    level_s = row[0] # starting level
dispatcher.utter_message("------ LESSON SUMMARY --------\n")
dispatcher.utter_message("Types of questions attempted:\n")
for i in range(level_s, level + 1):
    sql3 = "SELECT DISTINCT q_name FROM SQL_level WHERE level='%s' LIMIT
    ↪  1" % (i)
    try:
        mycursor.execute(sql3)
        results3 = mycursor.fetchall()
        for rows in results3:
            q_name = rows[0]
            dispatcher.utter_message(f">> {q_name}")
```

Listing 23: Lesson Summary Code

To update the user table, a simple SQL query is used to update the user table. The query updates the "rasa_person" table and the columns "level_sql" and "last_question_score" using a users ID, as seen in listing 24.

```
# in dataUpdateOnQuit
sql = "UPDATE rasa_person SET level_sql='%s',last_question_score='%s'
↪  WHERE id='%s'" % (level, last_question_score, id)
mycursor.execute(sql)
mydb.commit()
dispatcher.utter_message(f" Bye {name}! I hope to see you again soon")
```

Listing 24: Updating User Data

4.3 Web Deployment

To use this system easily and give it to users to test, the chatbot has to be deployed on the web. As mentioned previously, free support for Rasa X was removed in summer 2022. This means that a Rasa chatbot developed in Rasa Open Source could not be deployed to the web easily or for free, in the past deployment to the web only required one command in the terminal. This section will explain the work-around for deploying the chatbot without Rasa X.

Google Cloud Platform (GCP) is required as a virtual machine. To sign up, all that is required is a Google account which entitles a new user to $300 free credits in GCP. A new virtual machine was created with Ubuntu 20.0 and 10GB of storage. Rasa, Nginx and MySQL were installed on this machine using the command line. The Rasa open source project was cloned from a github repository so that it could be run on the virtual machine.

Once the Rasa chatbot was set up from the github repository, the Nginx web server was configured so that it could use socket.io channels in order to communicate with the chatbot and the webchat interface. A MySQL database was configured and set up as it was on the local machine, all tables were created and populated in the exact same way. When everything was installed and configured, a Rasa server and Rasa action server were started and the chatbot could be accessed through a URL link on the internet.

4.4 GDPR Considerations

For a real-life implementation of the chatbot described in this project General Data Protection Regulation (GDPR) would have to be considered. GDPR is European law that protects the public's personal data from malicious use. As this chatbot stores information about the user (name, email, SQL ability) and their interactions, GDPR has to be considered. A real system would abide by article 5.1-2 [65] as personal data is being processed. Article 5.1-2 states that personal data should be processed transparently, collected for a specific reason, collected necessary data only (adequate), accurate and up to date, stored for a certain period only, stored securely. Transparency about data processing and use would be handled in a real system by outlining to the user how their data is processed, why it is being processed/stored and how long their data is stored. The system already collects necessary data (name, email, SQL ability) for a specific reason (to personalise a user's interaction so that they have a more engaging and effective experience with the chatbot). A real system would also need to implement appropriate high security infrastructure and protocols. Security measures could include encryption of user data and database access control policies which can prevent malicious attacks such as spoofing, denial of service, elevation of privileges to name a few [66]. Using MySQL encryption functions for storing data in the database protects user's data from being viewed by anyone with access to the database. MySQL provides a range of encryption algorithms such as AES, SHA1, MD5 and more [67]. Database access control policies can be implemented to control which database users (developers) have certain permissions. Again, MySQL offers access control and authentication features which could be incorporated into a real chatbot system.

4.5 Example Chatbot Conversation

Figure 4.1 shows an example conversation between the chatbot system described in this dissertation and a user. The user is new to the system so they provide their name and email to the chatbot. The entities "name" and "email" are extracted from the intent recognised by the system. Through custom actions and database functions described in section 4.1.4 and 4.2 the user is added to the user database. This is depicted in figure 4.2 where the user's name and email are added, as well as initialising their SQL ability and scores.

Figure 4.1: New User in Conversation

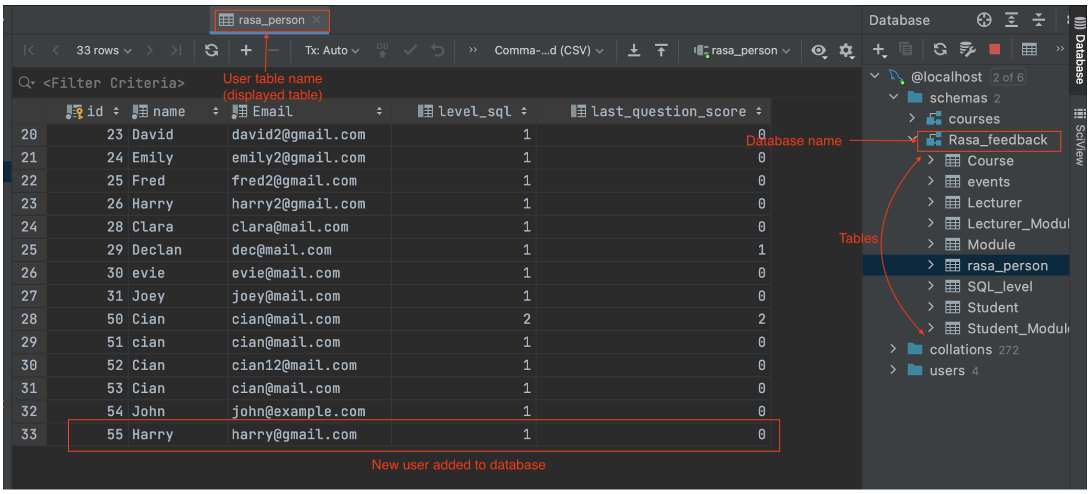


Figure 4.2: New User in Database

Figure 4.3 shows the online query console that students use to practice SQL queries before giving answers to the chatbot. For example, the user is given help on syntax needed for the question, in the conversation from figure 4.1. Using the query console the student can see if results sets are being returned that they expect and can test answers before asking for help, just as a student would in a tutoring session. Once the user sees that they have some sort of results from the SQL query console as in figure 4.3 they enter them to the chatbot so that they can see if they are correct.

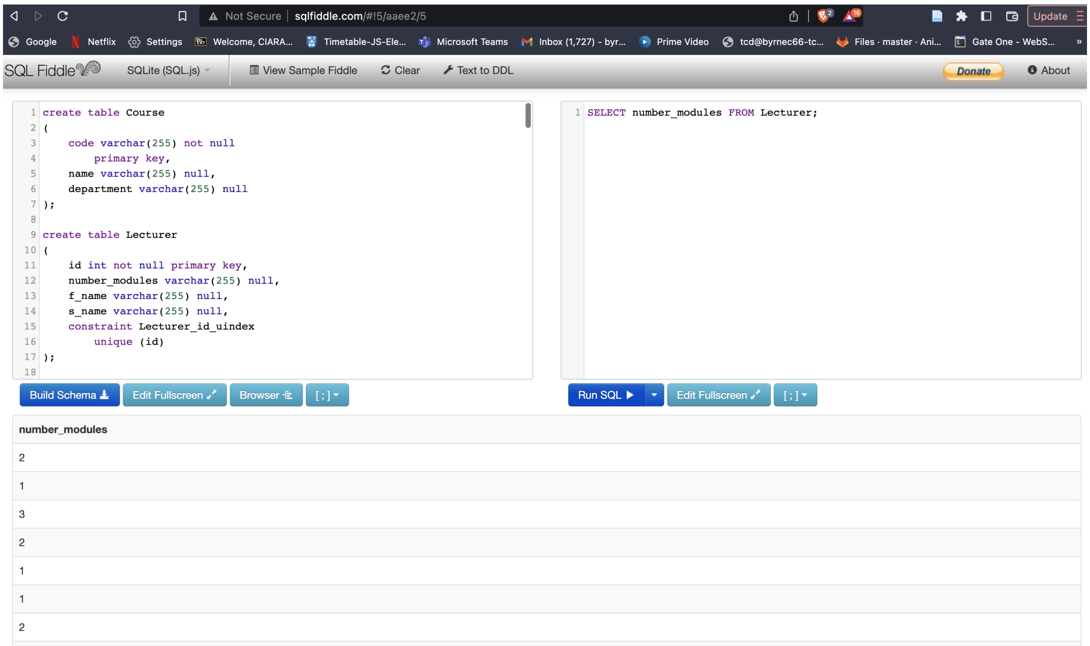


Figure 4.3: User Testing Online Query Console

Full conversation examples and system features are conveyed in figures in the appendix A2.

5 Evaluation

This section outlines the evaluation process for this dissertation. Details about evaluation objectives, the set up of the evaluations, results from evaluations and a discussion of the results are presented in this section.

5.1 Evaluation Objectives

Several evaluation objectives were defined before carrying out the evaluation on the system. According to Cahn [68], chatbots should be evaluated from an information retrieval point of view with accuracy, precision and F1-score but also from other perspectives such as user experience. Objectives were set so that every aspect of the chatbot system could be examined.

- *Functionality* of the system was measured by finding out the accuracy, precision, f1-score and performance.
- The *usability* of the system of the system had to be investigated to see how easy or hard to system was to navigate or use. This was evaluated using a customised version of the System Usability Scale (SUS) [69].
- *User experience* was an important aspect of evaluation. If the user was not happy with the experience then the system was not worth using to them. A good user experience entailed gauging how natural the conversation was or how human the assistant seemed.
- *Effectiveness* of the system was another objective of evaluation. This involved finding out how much the user learned or how effective it was as a tutor. Effectiveness was evaluated by measuring how much a student learned from using the system.

5.2 Evaluation Setup

The evaluation was split into two parts in order to complete the objectives listed in the previous section. In this section, the setup of system testing and user testing are explained. The design and process of carrying out system testing is explained in this section. The process, relevant questionnaire design and experiment management for user testing are presented but due to delays in Trinity College Dublin with ethical clearance, the process of user testing could not be carried out.

5.2.1 System Testing

The second part of testing was system testing where the accuracy and performance of the system was measured. For this, test data was manually generated in the form of Rasa test stories. These test stories are sample conversations with hard coded user input, action, intent and entity classifications. Rasa has built in testing programs where it uses the trained model and test data to give metrics such as accuracy, precision and f1-score and also provides nice graphs such as confusion matrices and histograms.

Accuracy in a chatbot system defines how well the chatbot made correct predictions in comparison to every prediction (correct and incorrect) this was outlined in equation 5.1 where True Positive (TP) and True Negative (TN) were divided over all predictions. A TP was when the chatbot predicted a label and it was correct (correctly predicted an intent), TN was when the chatbot did not make a false prediction, a False Positive (FP) was when the system made an incorrect prediction (classified the user input with the wrong intent), and False Negative (FN) can be described as an intent that the bot should have known due to its training but got incorrect in testing (e.g. confidence in prediction was not high enough to classify). Precision was the proportion of positive predictions that were correct (the number of correctly labelled intents over all labelled intents). F1-score was the combined measure of precision and recall. Recall in this scenario was $\frac{TP}{TP+FN}$ which was the number of correct predictions compared to the total number of predictions that should have been made. Accuracy, precision and F1-score were chosen as metrics as they were easily available in Rasa and they are commonly used to evaluate chatbots such as in [70–72] but to name a few.

The chatbot system was also evaluated in terms of intent classification using Rasa test programs which run the unseen test data with the chatbot and produce confusion matrices and histograms. Confusion matrices are graphs that depict a chatbot's performance in testing. A sign of perfect prediction in a confusion matrix can be seen in its shape. A diagonal pattern from top left-hand corner to bottom right-hand corner depicts perfect predictions (all predicted labels are correct). Rasa also provides a way to test using cross validation. Both normal Rasa test programs and cross validation were performed on two pipelines to compare and see which one performed better.

Some of the features of both pipelines were the following (see those highlighted in red were what differed):

- Rasa recommended pipeline: WhitespaceTokenizer, RegexFeaturizer, LexicalSyntacticFeaturizer, CountVectorsFeaturizer, DIETClassifier, EntitySynonymMapper, ResponseSelector, FallbackClassifier
- Spacy pipeline: "SpacyNLP"[1](model: "en_core_web_md",) SpacyTokenizer, SpacyFeaturizer, RegexFeaturizer, LexicalSyntacticFeaturizer, ountVectorsFeaturizer, DIETClassifier, EntitySynonymMapper, ResponseSelector, FallbackClassifier

5.2.2 User Testing

Evaluation of usability, user experience and effectiveness could only be evaluated using real users. In order for users to privately access the chatbot and use it easily the bot had to be deployed online. Unfortunately, as mentioned in previous sections, free support to deploy the bot to the web using Rasa-X was removed. This meant a website with the deployed bot had to be created. A google cloud platform VM was used to run a Nginx webserver which hosted the website which ran and displayed the Rasa chatbot. Rasa webchat was used for UI and the system stored everything via a MySQL server in the VM. To run the bot, the Rasa server had to be started on the VM as well as a Rasa action server. Once this was set up testing could begin. A link to the SQL tutor chatbot and a link to an online SQL query console was sent to a participant. The user interacted with the chatbot while also testing their answers in the query console before submitting them. Afterwards, participants were required to complete a questionnaire. The user interactions were also stored so that they could be analysed and used to improve the chatbot.

A questionnaire was designed for this evaluation such that users could provide relevant information about usability, user experience, effectiveness and the accuracy of responses. Many chatbot systems and papers use questionnaires as a means to evaluate usability as was obvious in a paper [72] which gave many examples of systems and papers that use questionnaires. System Usability Scale (SUS) [69] was used to evaluate the usability of the system. SUS contains 10 questions with a scale from 1-5 of how much or how little the user disagrees with a statement. Statements were added to the questionnaire to evaluate effectiveness, usability and accuracy of responses. These statements were ranked by the participant using the same scale as in SUS. The user was asked a question such as "This SQL chatbot helped me learn and understand SQL concepts" to evaluate effectiveness of the chatbot, "The system spoke like a human/seemed human-like" to learn about user experience and "The chatbot always responded with what I wanted" to find out about response accuracy of the chatbot.

[1] "SpacyNLP" is a pre-trained model that uses pre-trained word vectors.

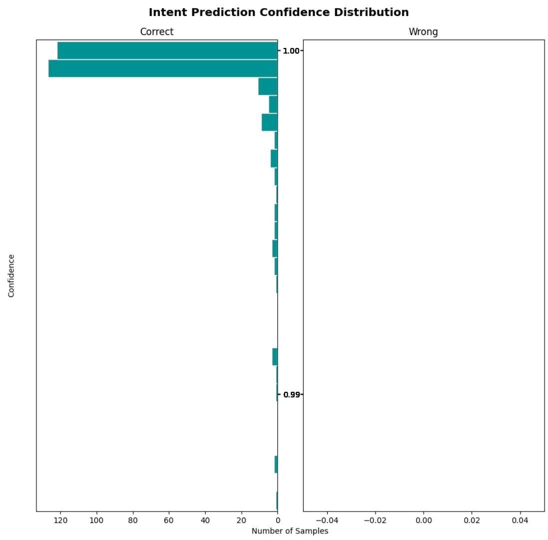
5.3 Evaluation Results

This section provides experiment results using both Rasa and Spacy pipelines

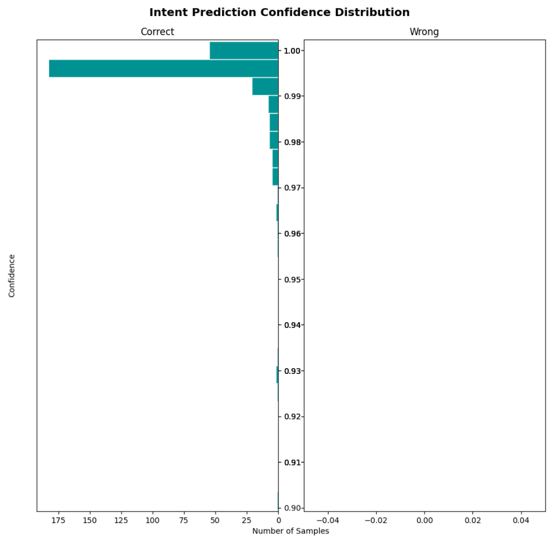
5.3.1 Normal Test

Seen in figure 5.2 is a confusion matrix for intent classification. This was obtained using Rasa's test programs, which involved typing "rasa test" in the command line once the test stories were written. This was done for both Rasa and Spacy pipelines however they produced the same confusion matrices for intent classification. The x-axis shows the label predicted by the system and the y-axis shows the actual label. The darker the shade of blue in the squares, the more examples there were in that box.

Figure 5.1 shows intent prediction confidence distributions for both Rasa in fig5.1a and Spacy in fig5.1b pipelines. The confidence is shown on the y-axis and the number of samples is shown on the x-axis.



(a) Rasa Pipeline Intent Histogram



(b) Spacy Intent Histogram

Figure 5.1: Intent Histograms

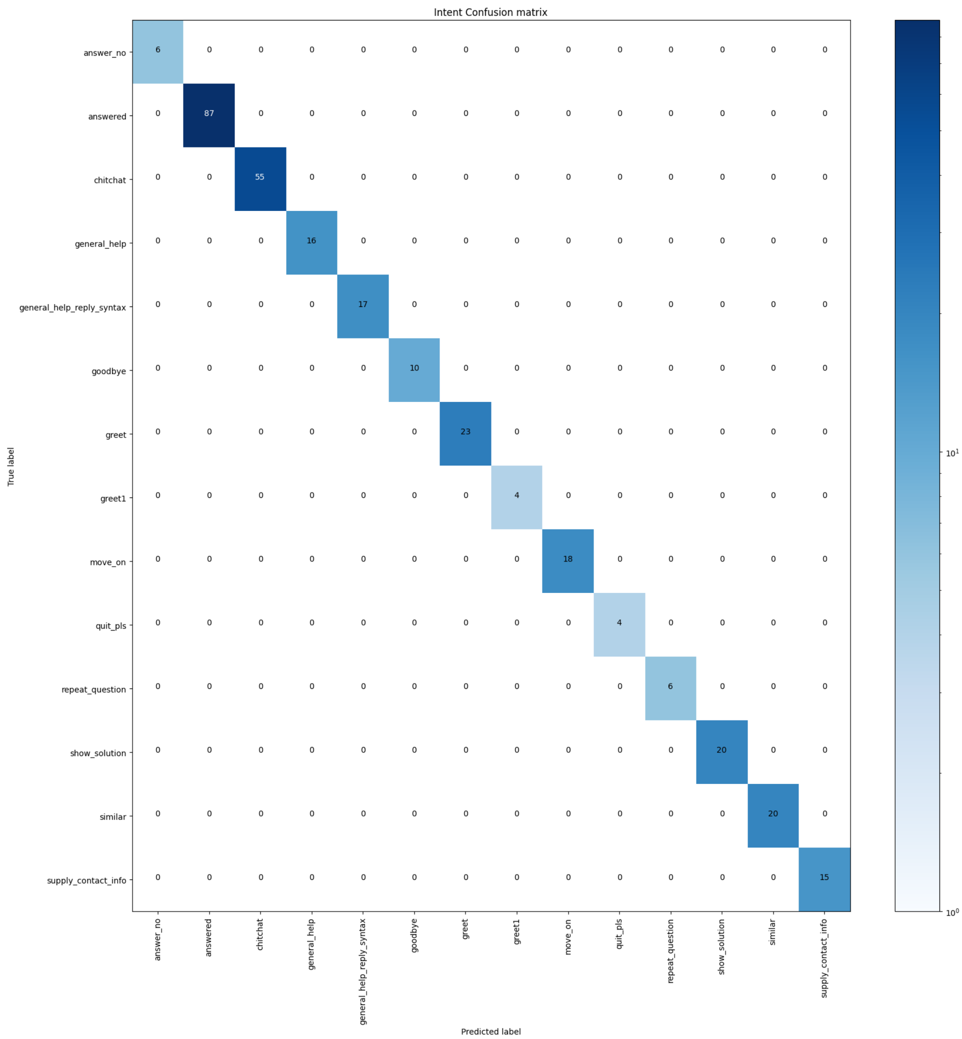


Figure 5.2: Rasa and Spacy Pipeline Confusion Matrix

5.3.2 5-Fold Cross Validation

K-fold cross validation is a common way to compare models in machine learning and classification problems. Cross validation sections the data in K folds and then split each fold into training and test data. The model was trained on the training sample and tested on the test data with scores being calculated at each fold. Rasa offers cross validation on test data using "rasa test nlu –nlu data/nlu –cross-validation –folds 5" on the command line where K=5. Rasa then calculated the accuracy, F1-score and precision for intent classification, entity evaluation and response evaluation at each fold and averaged these scores at the end. The accuracy can be described as the probability that a prediction was correct and can be calculated using equation 5.1. The F1-score can be used to judge how good a model performs and can be calculated using equation 5.2. Precision was the probability that a positive prediction was actually positive and can be calculated as in equation 5.3. Table 5.1 shows these metrics, with the best scores highlighted in red.

	Accuracy		F1 Score		Precision	
	Rasa	Spacy	Rasa	Spacy	Rasa	Spacy
Intent Classification						
Train	1.0	0.999	1.0	0.999	1.0	0.999
Test	0.939	0.908	0.933	0.900	0.942	0.903
Entity Evaluation						
Train	1.0	0.999	1.0	0.979	1.0	0.980
Test	0.991	0.993	0.744	0.834	0.805	0.914
Response Selection Evaluation						
Train	1.0	1.0	1.0	1.0	1.0	1.0
Test	0.764	0.673	0.759	0.650	0.800	0.703

Table 5.1: 5-fold Cross Validation Results

In order to better understand the results, averages were calculated for all test scores and variances. In table 5.2 the average accuracy, F1-score and precision was calculated across all test evaluations (intent classification, entity evaluation and response selection evaluation) using equation to find the mean: $\frac{1}{n}\sum_{i=i}^{n} x_i$. This was done to see which pipeline performed best in each metric.

	Rasa (variance)	Spacy (variance)
Accuracy	0.898 (0.0657)	0.858 (0.056)
F1	0.812 (0.077)	0.795 (0.09)
Precision	0.849 (0.0993)	0.840 (0.0943)

Table 5.2: Cross Validation Average Test Scores

The average for each evaluation was calculated for both pipelines in table 5.3. Again, the best scores can be seen highlighted in red. This was calculated to see which pipeline performed best in each evaluation.

	Rasa	Spacy
Intent Classification	0.938	0.904
Entity Evaluation	0.846	0.914
Response Selection	0.774	0.675

Table 5.3: Cross Validation Average Test Scores At Each Evaluation

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

$$F1 - score = \frac{2 * TP}{2 * TP + FP + FN} \tag{5.2}$$

$$Precision = \frac{TP}{TP + FP} \tag{5.3}$$

Where TP = True Positive, TN = True Negative, FP = False Positive, FN = False Negative.

5.3.3 Statistical Significance Tests

Performing statistical significance tests was essential to understand whether the difference in performances of the two models' was significant or whether there was any difference in the models at all. Observations can be made but without knowing the significance of results, conclusions can not be made about which model performed best. Two statistical significance tests were performed, testing the difference of proportions of the model's performances and using the McNemar test to compare the models. These were taken from a paper by Raschka [2] which described details about these statistical tests.

Testing the Difference of Proportions

This involved testing the difference of proportions (z-score test) of the two models' performance which was accuracy, F1-score and precision. A null hypotheses was stated that the two models have the same population proportions. The null could be rejected if the confidence intervals of the metrics being tested from both models did not overlap. Raschka stated in the paper that this method is one of the most straight forward ones, but not the most robust [2].

The null hypothesis was stated: *H0 = Two models have the same population proportions* with a significance threshold of 5% (which is standard). Next the Z-statistic was calculated:

$$z = \frac{ACC_1 - ACC_2}{\sqrt{\sigma_1^2 + \sigma_2^2}} \tag{5.4}$$

Where ACC_1 and ACC_2 represent the average performance metrics being compared from the two models. Average variances were shown as σ_1^2 and σ_2^2.

$$z_{Accuacy} = \frac{89.8 - 85.8}{\sqrt{6.57 + 5.6}} = 1.147 \tag{5.5}$$

$$z_{F1} = \frac{81.2 - 79.5}{\sqrt{7.7 + 9.0}} = 0.416 \tag{5.6}$$

$$z_{Precision} = \frac{84.9 - 84.0}{\sqrt{9.93 + 9.43}} = 0.205 \tag{5.7}$$

Values for equations 5.5, 5.6 and 5.7 were taken from table 5.2 which shows the average of all scores and variances across all evaluations in 5-fold cross validation. Raschka remarked that the null hypothesis can be rejected if $|z| > 1.96$ (confidence level of 0.05). From results of $z_{Accuacy}$, z_{F1} and $z_{Precision}$ it was evident that none were greater than 1.96. Thus, failing to reject the null hypothesis with this test.

Comparing Two Models with the McNemar Test

As mentioned the difference of proportions method can be a bad choice for testing statistical significance between models. Raschka explained McNemar test for comparing models which can also be called "within-subjects chi-squared test" [2]. The layout of models to compute values using this test was depicted in figure.

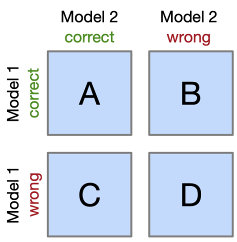


Figure 5.3: Confusion matrix layout in context of McNemar's test taken from [2]

The null hypothesis for this test was *H0 = probabilities of B and C are the same*. Which means that the models performed the same, one did not perform better than the other. Using equation 5.8, the "chi-squared" value as obtained. Using this and one degree of freedom a software package [73] to look up the p-value. The null hypothesis could be rejected if the p-value was smaller than confidence threshold (0.05). Thus, the difference in the models would be statistically significant.

$$X^2 = \frac{(B - C)^2}{B + C} \tag{5.8}$$

The same layout confusion matrix was recreated using intent classification confusion matrices and responses selection confusion matrices from 5-fold cross validation as in figure 5.3. The original matrices for intent classification during cross validation in appendix figures A3.1 and A3.2 A3 were used to find the confusion matrices in tables 5.4 and 5.4 which were formatted as in figure 5.3 of McNemar's layout. This was done by observing the confusion matrices to see where both models were correct and incorrect in their predictions versus actual values.

		Spacy Correct	Spacy Wrong
Rasa	Correct	265	10
Rasa	Wrong	1	17

Table 5.4: Cross Validation Intent Confusion Matrix Summary

To compare the two models using intent classification in cross fold validation, the values for B and C were substituted into equation 5.8.

$$X^2_{Intent} = \frac{(10-1)^2}{10+1} = 7.364 \tag{5.9}$$

An online calculator was used to find to find out the p-value [73] using 7.364 equal to chi-squared, 1 degree of freedom and 0.05 significance level. This gave a p-value of .006654, which is less than 0.05. Meaning that the Rasa model and Spacy model were different enough to reject the null hypothesis.

		Spacy Correct	Spacy Wrong
Rasa	Correct	35	7
Rasa	Wrong	2	11

Table 5.5: Cross Validation Response Selection Confusion Matrix Summary

To compare the two models using response selection in cross fold validation, the values for B and C were substituted into equation 5.8.

$$X^2_{Response} = \frac{(7-2)^2}{7+2} = 2.778 \tag{5.10}$$

Again, an online calculator was used with $X^2_{Response}$ to find the p-value = 0.095567. This p-value was less than 1.96 and therefore the null hypothesis was not rejected. Thus, there was no difference in the models when it came to response selection.

Exact p-Values via the Binomial Test

Another test Raschka used in the paper was calculating exact p-values for testing the statistical significance of models [2]. The paper claims that while McNemar's test is sound, a more computationally expensive test to compute p-values when B and C are relatively small (B+C < 25). The values for B and C in the Spacy pipeline model and Rasa pipeline model are small, so binomial tests were performed to further understand the difference (if any) in the two models. Again, the null hypothesis was *H0 = both models perform the same* and this was done testing the intent classification and response selection of both models.

$$p = 2\sum_{i=B}^{n}\binom{n}{i}(0.5)^{i}(1-0.5)^{n-i} \tag{5.11}$$

An equation was presented to give the p-value using values found in the McNemar formatted confusion matrices (in tables 5.4 and 5.5) [2]. The value n in equation 5.11 was calculated $n = B + C$. Where B was the value where Spacy was wrong and Rasa was right, C was the value where Spacy was correct and Rasa was wrong.

$$P_{Intent} = 2\sum_{i=10}^{11}\binom{n}{i}(0.5)^{i}(1-0.5)^{n-i} = 0.01171 \tag{5.12}$$

$$P_{Response} = 2\sum_{i=7}^{9}\binom{n}{i}(0.5)^{i}(1-0.5)^{n-i} = 0.1797 \tag{5.13}$$

The p-value for intent recognition in cross validation using exact p-values was $P_{Intent} = 0.01171$ from equation 5.12. Thus, $P_{Intent} < 0.05$ which means that both models did not perform the same with respect to intent classification and the null hypothesis was rejected (at a confidence level of 0.05). However, $P_{Response} = 0.1797$ and therefore $P_{Response} > 0.05$ so the null hypothesis is not rejected meaning both models performed the same in terms of response selection.

5.4 Discussion of Results

In this section, results are interpreted and discussed so that conclusions about the performance of the chatbot system can be drawn. The only thing in these models to change was the pipeline, all other training and testing data was the same.

From the histograms of the intent prediction confidence distributions in figure 5.1 a few things can be seen. There were more samples at a higher confidence for the Rasa pipeline with 120 samples at confidence of 1.0 (figure 5.1a) compared to the Spacy pipeline with around 60 samples at a confidence of 1.0 (figure 5.1b). As well as this, the range of intent confidence was bigger in the Spacy pipeline indicating lower confidence scores were hit in Spacy[2]. Spacy confidence range was from 1.0-0.90 and Rasa pipeline ranged in confidence from 1.0-0.99. This shows that Rasa pipeline produced higher confidence for more samples

[2]The model trained with the Spacy pipeline may be referred to as Spacy and the model trained with the Rasa pipeline may be referred to as Rasa

than the Spacy pipeline. However the difference was not significant as both models performed perfect according to the confusion matrix. The confusion matrix in figure 5.2 demonstrated that both pipelines produced models which classify intents accurately. As mentioned previously, both models produced the same confusion matrix. The confusion matrix shows a diagonal distribution of boxes. This indicates that all predicted labels were also true labels, which indicates there were no wrong classifications in these tests. The lack of wrong intent classification distribution on the right side of histograms in figure 5.1 also indicates the same thing.

In subsection 5.3.2 cross validation results are presented in tables 5.1, 5.2, 5.3. The first table 5.1 contains all results from both models in 5-fold cross validation. As can be seen highlighted in red, the Rasa pipeline model scored higher in 6 out the 9 results from the test set. The reason test set metrics were focused on is due to the fact that test set metrics are better indications of how the models did on unseen data. The next table 5.2 displays how each model performed in each metric, accuracy, F1-score and precision. Rasa pipeline model's accuracy differed from Spacy by 0.04, which was the largest observed metric difference between the two. The average scores in each evaluations are displayed in table 5.3. Rasa had a positive difference in 2 out of the 3 evaluation categories, intent classification and response selection evaluation. However, the Spacy model had a higher average in entity evaluation with a score of 0.914 compared to Rasa's score of 0.846. Overall, these tables indicate that the model trained with a Rasa recommended pipeline have higher scores than the Spacy pipeline model. However, without knowing the significance of these differences a conclusion about the better performing model could not be drawn.

Thus, statistical significance tests were conducted to find out if the models differ. Three significance tests were carried out to fully understand the results presented in subsection 5.3.2. When testing the difference in proportions it was found that all mean metrics provided in table 5.2 failed to reject the null hypothesis that the models were the same. Hence, the difference in average accuracy, F1-scores and precision scores were not great enough to state that one model performs better than the other (in terms of those metrics in table 5.2). However, when conducting the McNemar test it was determined that the Rasa and Spacy models were different enough to reject the null hypothesis. Meaning there was sufficient evidence to prove that the Rasa pipeline model performed better with intent classification than the Spacy pipeline model. This fact was also verified with the use of binomial tests to find the exact p-values. Where an exact p-value of 0.01171 confirmed that the Rasa and Spacy models differed. This was not the case with response selection and the two models. Both McNemar and binomial tests failed to reject the null hypothesis. Thus, there was no enough evidence to state that the models were different in terms of response selection.

Although user testing could not go ahead, anecdotal evidence was collected to confirm that the system works for users and there were no complaints about it.

6 Conclusion

In this section, conclusions are made about the design, implementation and results from this study. Future work is also discussed.

6.1 Conclusion

In Chapter 1 objectives for this dissertation were outlined. Objective one was to perform a literature review of the current state-of-the-art in personalised, task-based and educational chatbots. Chapter 2 reviewed and analysed the history and current state-of-the-art in chatbot research and industry chatbots. Types of chatbots, applications of chatbots and chatbot training corpora were highlighted in this chapter. Hence, chatbots and their technology were understood and objective one completed.

The second objective was to take a case study approach to design and implement a personalised educational chatbot that assumed the role of a teaching assistant teaching the coding language SQL. As seen in Chapters 3 and 4, a chatbot system was designed and implemented using Rasa Open Source and manually created training data. This system implemented features that created a personalised learning environment. Such features included remembering students and reminding them of their progress, guiding students through SQL exercises, allowing students to learn at their own pace, giving students the opportunity to ask any SQL related questions they had, helping students when they were struggling and giving them effective feedback. The chatbot understood context and had a pedagogical model which comprised of structured SQL exercises described in section 3.6. Thus, objective two was achieved.

The third and final objective was to design an evaluation process and evaluate the chatbot system. The aim was to investigate the functionality, usability, user experience, effectiveness, performance and accuracy of the chatbot. Chapter 5 outlined the evaluation process for the system. The system was due to be evaluated in two ways, user testing and system testing. User testing was supposed to find out about the usability, user experience, effectiveness and functionality. As mentioned previously, user testing could not go ahead as ethical approval was not obtained by time of submission of this dissertation. This was a major limitation of the system because the chatbot could not be fully evaluated and therefore could not be improved. However, system testing was completed which involved testing the system using manually created test data which produced accuracy, F1-score and precision metrics for two models trained with two different pipelines. This testing also provided information about intent classification, entity extraction and response selection of the chatbot. The third objective was not fully achieved because details of the system's usability, user experience, effectiveness and functionality could not be investigated.

Information about the chatbot's performance was obtained from the evaluation of the system described in Chapter 5. Overall performance of the system was very promising as seen in the confusion matrix 5.2, intent distribution histograms 5.1 and tables 5.1, 5.2 and 5.3. The confusion matrix showed a diagonal alignment meaning all predicted intents were actual intents. There were also no wrong intent predictions in the histograms. Both models were said to be accurate due to there being no wrong intent predictions and no significant difference was observed between the models during normal testing. The results from 5-fold cross validation were investigated with difference of proportions, McNemar's test and binomial test and they showed that both models performed the same in response selection and performance metrics such as accuracy, F1-score and precision. However, it was found that the Rasa pipeline model performed better in intent classification.

6.2 Future Work

A personalised educational chatbot was created for the purpose of this study. This chatbot tutored students specifically on SQL exercises and questions concerning SQL SELECT statement with one table. As modelled in section 3.6, this can be expanded for more domains of SQL such as SQL SELECT with two tables or exercises not concerning the SELECT statement but rather UPDATE (which updates values in tables or databases) or using a TRIGGER statements to name a few possibilities. Future work could expand the knowledge and capabilities of the current chatbot by incorporating more domains of SQL such as the ones mentioned. This would allow for a wider range in student ability as students go into a course with varying abilities from beginner to expert, the chatbot should cater to all of these in order to fully resemble a human tutor.

Future work could investigate the effect of using this chatbot system as a tool for students to use in their SQL course. The goal of this system was never to replace the tutor but act as a resource for when there is no tutor available. The prospect of this system being used as an additional pedagogical resource to help students is fascinating. Students could be given the chatbot at the start of a semester and asked to interact with it as part of tutorials. The students would be given a questionnaire at the end of the course and their interactions with the bot could also be analysed. A control group, who do not have the chatbot, could be used to gauge how effective the chatbot is in helping students learn. This evaluation would be useful to see how truly useful and effective this system is.

Bibliography

[1] Rasa Architecture Overview. [Online]. Available: https://rasa.com/docs/rasa/arch-overview/

[2] S. Raschka, "Model evaluation, model selection, and algorithm selection in machine learning," *arXiv preprint arXiv:1811.12808*, 2018.

[3] Z. Zhang, R. Takanobu, Q. Zhu, M. Huang, and X. Zhu, "Recent advances and challenges in task-oriented dialog systems," *Science China Technological Sciences*, vol. 63, no. 10, pp. 2011–2027, 2020.

[4] G. Caldarini, S. Jaf, and K. McGarry, "A literature survey of recent advances in chatbots," *Information*, vol. 13, no. 1, p. 41, 2022.

[5] J. Weizenbaum, "Eliza—a computer program for the study of natural language communication between man and machine," *Communications of the ACM*, vol. 9, no. 1, pp. 36–45, 1966.

[6] K. M. Colby, S. Weber, and F. D. Hilf, "Artificial paranoia," *Artificial intelligence*, vol. 2, no. 1, pp. 1–25, 1971.

[7] "Introducing ChatGPT." [Online]. Available: https://openai.com/blog/chatgpt

[8] N. Bhirud, S. Tataale, S. Randive, and S. Nahar, "A literature review on chatbots in healthcare domain," *International journal of scientific & technology research*, vol. 8, no. 7, pp. 225–231, 2019.

[9] M. Adam, M. Wessel, and A. Benlian, "Ai-based chatbots in customer service and their effects on user compliance," *Electronic Markets*, vol. 31, no. 2, pp. 427–445, 2021.

[10] R. Pillai and B. Sivathanu, "Adoption of ai-based chatbots for hospitality and tourism," *International Journal of Contemporary Hospitality Management*, vol. 32, no. 10, pp. 3199–3226, 2020.

[11] J. Q. Pérez, T. Daradoumis, and J. M. M. Puig, "Rediscovering the use of chatbots in education: A systematic literature review," *Computer Applications in Engineering Education*, vol. 28, no. 6, pp. 1549–1565, 2020.

[12] P. B. Brandtzaeg and A. Følstad, "Why people use chatbots," in *Internet Science: 4th International Conference, INSCI 2017, Thessaloniki, Greece, November 22-24, 2017, Proceedings 4*. Springer, 2017, pp. 377–392.

[13] R. Winkler and M. Söllner, "Unleashing the potential of chatbots in education: A state-of-the-art analysis," in *Academy of Management Annual Meeting (AOM)*, 2018.

[14] S. Crowe, K. Cresswell, A. Robertson, G. Huby, A. Avery, and A. Sheikh, "The case study approach," *BMC medical research methodology*, vol. 11, no. 1, pp. 1–9, 2011.

[15] A. M. TURING, "I.—COMPUTING MACHINERY AND INTELLIGENCE," *Mind*, vol. LIX, no. 236, pp. 433–460, Oct. 1950.

[16] R. S. Wallace, "The Anatomy of A.L.I.C.E." in *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer*, R. Epstein, G. Roberts, and G. Beber, Eds. Dordrecht: Springer Netherlands, 2009, pp. 181–210.

[17] H. Shah and K. Warwick, "Machine humour: examples from turing test experiments," *AI & SOCIETY*, vol. 32, pp. 553–561, 2017.

[18] T. Kay and R. Hoffer, "Method and system for interactively responding to instant messaging requests," Patent US6 430 602B1, Aug., 2002.

[19] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heinz, and D. Roth, "Recent advances in natural language processing via large pre-trained language models: A survey," *arXiv preprint arXiv:2111.01243*, 2021.

[20] Y. Shen, L. Heacock, J. Elias, K. D. Hentel, B. Reig, G. Shih, and L. Moy, "ChatGPT and other large language models are double-edged swords," *Radiology*, vol. 307, no. 2, p. e230163, 2023.

[21] D. Jurafsky and J. H. Martin, "Speech and language processing," 2023, draft. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/

[22] N. Mrkšić, D. O. Séaghdha, T.-H. Wen, B. Thomson, and S. Young, "Neural belief tracker: Data-driven dialogue state tracking," *arXiv preprint arXiv:1606.03777*, 2016.

[23] P. Shah, D. Hakkani-Tür, G. Tür, A. Rastogi, A. Bapna, N. Nayak, and L. Heck, "Building a conversational agent overnight with dialogue self-play," *arXiv preprint arXiv:1801.04871*, 2018.

[24] P. Budzianowski, T.-H. Wen, B.-H. Tseng, I. Casanueva, S. Ultes, O. Ramadan, and M. Gašić, "Multiwoz–a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling," *arXiv preprint arXiv:1810.00278*, 2018.

[25] T. Yu, R. Zhang, H. Y. Er, S. Li, E. Xue, B. Pang, X. V. Lin, Y. C. Tan, T. Shi, Z. Li *et al.*, "Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases," *arXiv preprint arXiv:1909.05378*, 2019.

[26] A. Rastogi, X. Zang, S. Sunkara, R. Gupta, and P. Khaitan, "Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 8689–8696.

[27] Y.-L. Hsueh and T.-L. Chou, "A task-oriented chatbot based on lstm and reinforcement learning," *ACM Transactions on Asian and Low-Resource Language Information Processing*, vol. 22, no. 1, pp. 1–27, 2022.

[28] Z. Yan, N. Duan, P. Chen, M. Zhou, J. Zhou, and Z. Li, "Building task-oriented dialogue systems for online shopping," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[29] Z. Wei, Q. Liu, B. Peng, H. Tou, T. Chen, X.-J. Huang, K.-F. Wong, and X. Dai, "Task-oriented dialogue system for automatic diagnosis," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2018, pp. 201–207.

[30] H. Fan and M. S. Poole, "What is personalization? perspectives on the design and implementation of personalization in information systems," *Journal of Organizational Computing and Electronic Commerce*, vol. 16, no. 3-4, pp. 179–202, 2006.

[31] M. Bahja, R. Hammad, and M. Hassouna, "Talk2learn: a framework for chatbot learning," in *Transforming Learning with Meaningful Technologies: 14th European Conference on Technology Enhanced Learning, EC-TEL 2019, Delft, The Netherlands, September 16–19, 2019, Proceedings 14*. Springer, 2019, pp. 582–586.

[32] M. Shumanov and L. Johnson, "Making conversations with chatbots more personalized," *Computers in Human Behavior*, vol. 117, p. 106627, 2021.

[33] D. Fernau, S. Hillmann, N. Feldhus, T. Polzehl, and S. Möller, "Towards personality-aware chatbots," in *Proceedings of the 23rd Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2022, pp. 135–145.

[34] R. Kocielnik, L. Xiao, D. Avrahami, and G. Hsieh, "Reflection companion: a conversational system for engaging users in reflection on physical activity," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 2, pp. 1–26, 2018.

[35] C. W. Okonkwo and A. Ade-Ibijola, "Chatbots applications in education: A systematic review," *Computers and Education: Artificial Intelligence*, vol. 2, p. 100033, Jan. 2021.

[36] L. K. Fryer, M. Ainley, A. Thompson, A. Gibson, and Z. Sherlock, "Stimulating and sustaining interest in a language course: An experimental comparison of Chatbot and Human task partners," *Computers in Human Behavior*, vol. 75, pp. 461–468, Oct. 2017.

[37] S. Hobert, "Say Hello to 'Coding Tutor'! Design and Evaluation of a Chatbot-based Learning System Supporting Students to Learn to Program," *ICIS 2019 Proceedings*, Nov. 2019.

[38] L. W. Anderson and D. R. Krathwohl, *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Longman,, 2001.

[39] B. S. Bloom and D. R. Krathwohl, *Taxonomy of educational objectives: The classification of educational goals. Book 1, Cognitive domain*. longman, 2020.

[40] M. Coronado, C. A. Iglesias, Á. Carrera, and A. Mardomingo, "A cognitive assistant for learning java featuring social dialogue," *International Journal of Human-Computer Studies*, vol. 117, pp. 55–67, 2018.

[41] A. Mitrovic, "An Intelligent SQL Tutor on the Web," *International Journal of Artificial Intelligence in Education*, vol. 13, no. 2-4, pp. 173–197, 2003.

[42] A. Mitrovic and S. Ohlsson, "Evaluation of a constraint-based tutor for a database language," *International Journal of Artificial Intelligence in Education*, vol. 10, no. 3-4, pp. 238–256, 1999.

[43] S. Ohlsson, "Constraint-based student modeling," in *Student modelling: the key to individualized knowledge-based instruction*. Springer, 1994, pp. 167–189.

[44] A. M. Latham, K. A. Crockett, D. A. McLean, B. Edmonds, and K. O'shea, "Oscar: An intelligent conversational agent tutor to estimate learning styles," in *International conference on fuzzy systems*. IEEE, 2010, pp. 1–8.

[45] C. W. Okonkwo and A. Ade-Ibijola, "Python-Bot: A Chatbot for Teaching Python Programming," *Engineering Letters*, vol. 29, no. 1, pp. 25–34, 2020.

[46] K. R. Chowdhary, "Natural Language Processing," in *Fundamentals of Artificial Intelligence*. New Delhi: Springer India, 2020, pp. 603–649.

[47] R. Winkler, S. Hobert, A. Salovaara, M. Söllner, and J. M. Leimeister, "Sara, the lecturer: Improving learning in online education with a scaffolding-based conversational agent," in *Proceedings of the 2020 CHI conference on human factors in computing systems*, 2020, pp. 1–14.

[48] M. Verleger and J. Pembridge, "A pilot study integrating an ai-driven chatbot in an introductory programming course," in *2018 IEEE frontiers in education conference (FIE)*. IEEE, 2018, pp. 1–4.

[49] F. Clarizia, F. Colace, M. Lombardi, F. Pascale, and D. Santaniello, "Chatbot: An Education Support System for Student," in *Cyberspace Safety and Security*, A. Castiglione, F. Pop, M. Ficco, and F. Palmieri, Eds. Cham: Springer International Publishing, 2018, pp. 291–302.

[50] A. C. Graesser, S. Lu, G. T. Jackson, H. H. Mitchell, M. Ventura, A. Olney, and M. M. Louwerse, "Autotutor: A tutor with dialogue in natural language," *Behavior Research Methods, Instruments, & Computers*, vol. 36, pp. 180–192, 2004.

[51] D. Song, E. Y. Oh, and M. Rice, "Interacting with a conversational agent system for educational purposes in online courses," in *2017 10th International Conference on Human System Interactions (HSI)*, 2017, pp. 78–82.

[52] E. Adamopoulou and L. Moussiades, "An Overview of Chatbot Technology," in *Artificial Intelligence Applications and Innovations*, I. Maglogiannis, L. Iliadis, and E. Pimenidis, Eds. Cham: Springer International Publishing, 2020, pp. 373–383.

[53] S. Hussain, O. Ameri Sianaki, and N. Ababneh, "A Survey on Conversational Agents/Chatbots Classification and Design Techniques," in *Web, Artificial Intelligence and Network Applications*, L. Barolli, M. Takizawa, F. Xhafa, and T. Enokido, Eds. Cham: Springer International Publishing, 2019, pp. 946–956.

[54] M. d. G. B. Marietto, R. V. de Aguiar, G. d. O. Barbosa, W. T. Botelho, E. Pimentel, R. d. S. França, and V. L. da Silva, "Artificial intelligence markup language: a brief tutorial," *arXiv preprint arXiv:1307.3091*, 2013.

[55] "Dialogflow Documentation." [Online]. Available: https://cloud.google.com/dialogflow/docs

[56] "What is MySQL?" [Online]. Available: https://www.oracle.com/mysql/what-is-mysql/

[57] "What is NGINX?" [Online]. Available: https://www.nginx.com/resources/glossary/nginx/

[58] "SQL Tutorial." [Online]. Available: https://www.w3schools.com/sql/default.asp

[59] H. Jang, J. Reeve, and E. L. Deci, "Engaging students in learning activities: It is not autonomy support or structure but autonomy support and structure." *Journal of educational psychology*, vol. 102, no. 3, pp. 588–600, 2010.

[60] C. E. Hmelo-Silver, "Problem-based learning: What and how do students learn?" *Educational psychology review*, vol. 16, pp. 235–266, 2004.

[61] 50 SQL Interview Questions and Answers for 2023. [Online]. Available: https://www.guru99.com/sql-interview-questions-answers.html

[62] Top 115 SQL Interview Questions You Must Prepare In 2023. [Online]. Available: https://www.edureka.co/blog/interview-questions/sql-interview-questions

[63] SQL Interview Questions. [Online]. Available: https://www.interviewbit.com/sql-interview-questions/

[64] T. L. Good and J. E. Brophy, *Educational psychology: A realistic approach*. Longman/Addison Wesley Longman, 1990.

[65] "Art. 5 GDPR - Principles relating to processing of personal data," Nov. 2018, section: Uncategorized. [Online]. Available: https://gdpr.eu/article-5-how-to-process-personal-data/

[66] M. Hasal, J. Nowaková, K. Ahmed Saghair, H. Abdulla, V. Snášel, and L. Ogiela, "Chatbots: Security, privacy, data protection, and social aspects," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 19, 2021.

[67] "MySQL :: MySQL 8.0 Reference Manual :: 12.14 Encryption and Compression Functions." [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/encryption-functions.html

[68] J. Cahn, "Chatbot: Architecture, design, & development," *University of Pennsylvania School of Engineering and Applied Science Department of Computer and Information Science*, 2017.

[69] "System Usability Scale (SUS)," 2013, publisher: Department of Health and Human Services. [Online]. Available: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

[70] A. Jiao, "An Intelligent Chatbot System Based on Entity Extraction Using RASA NLU and Neural Network," *Journal of Physics: Conference Series*, vol. 1487, no. 1, p. 012014, 2020.

[71] F. Almusharraf, J. Rose, and P. Selby, "Engaging unmotivated smokers to move toward quitting: Design of motivational interviewing–based chatbot through iterative interactions," *Journal of Medical Internet Research*, vol. 22, no. 11, p. e20251, 2020.

[72] R. Ren, J. W. Castro, S. T. Acuña, and J. de Lara, "Evaluation techniques for chatbot usability: A systematic mapping study," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 11n12, pp. 1673–1702, 2019.

[73] "Quick P Value from Chi-Square Score Calculator." [Online]. Available: https://www.socscistatistics.com/pvalues/chidistribution.aspx

A1 Appendix

A1.1 SQL Question Design

In chapter 3, tables of modelled domains and SQL questions are presented, with example questions and answers based on the topic. Here, tables A1.1, A1.2 and A1.3 show each topic in an SQL domain with a description of each topic (level) and an outline of the SQL syntax. Table A1.1 shows topics in domain one, which involve SQL SELECT queries with one table. Table A1.2 shows domain two which involves problems with two tables, and table A1.3 in domain three which concern exercises with three tables.

Domain One:

Level	Description	Outline
1	SELECT statement (basic)	`SELECT col1,col2... FROM table1;`
2	SELECT all using *	`SELECT * FROM table1 ;`
3	SELECT Distinct	`SELECT DISTINCT col1,col2... FROM table1;`
4	WHERE using simple operations (=, <, >)	`SELECT col1,col2...` `FROM table1 WHERE condition;`
5	WHERE w multiple conditions using one of (AND, OR, NOT)	`SELECT * FROM table1` `WHERE col1 = var1 AND col2 = var2;`

6	WHERE w multiple conditions using multiple of (AND, OR, NOT)	`SELECT * FROM table1`
		`WHERE col1 = var1`
		`AND (col2 = var2 OR col2 = var3) ;`

7	Order by ascending or descending	`SELECT col1,col2...` `FROM table1 ORDER BY ASC/DESC;`

8	SELECT Top or set Limit	`SELECT col1,col2... FROM table1`
		`WHERE condition`
		`LIMIT num;`

9	Get minimum and maximum of columns	`SELECT MAX(col1) FROM table1` `WHERE condition;`
10	Count	`SELECT COUNT(col1) FROM table1` `WHERE condition ;`
11	Average	`SELECT AVG(col1) FROM table1` `WHERE condition ;`
12	Sum	`SELECT SUM(col1) FROM table1` `WHERE condition ;`
13	Group by	`SELECT COUNT(col1),col2 FROM table1` `GROUP BY col2;`
14	'IN' operator	`SELECT * FROM table1` `WHERE col2 IN (var1,var2);`

15	Having clause. WHERE cannot be used with aggregate functions	`SELECT COUNT(col1),col2 FROM table1`
		`GROUP BY col2`
		`HAVING COUNT(col1) < 2;`

16	Using aliases	`SELECT col1 AS my_col_name FROM table1;`

Table A1.1: SQL levels and questions for domain 1 with description of topics and template SQL examples

Domain Two:

Level	Description	Outline
1	SELECT	`SELECT table1.col1, table2.col2` `FROM table1,table2 ;`
2	SELECT *	`SELECT * FROM table1,table2 ;`
3	SELECT DISTINCT	`SELECT DISTINCT table1.col1, table2.col2` `FROM table1, table2;`

4	JOIN	`SELECT table1.col1,table2.col2 FROM table1`
		`JOIN table2`
		`ON table1.col2 = table2.col3;`

5	LEFT JOIN	`SELECT table1.col1,table2.col2 FROM table1`
		`LEFT JOIN table2`
		`ON table1.col2 = table2.col3;`

6	RIGHT JOIN	`SELECT table1.col1,table2.col2 FROM table1`
		`RIGHT JOIN table2`
		`ON table1.col2 = table2.col3;`

7	UNION	`SELECT col1, FROM table1`
		`UNION`
		`SELECT col2, FROM table2;`

8	UNION ALL	`SELECT col1, FROM table1`
		`UNION ALL`
		`SELECT col2, FROM table2;`

9	UNION with WHERE	`SELECT col1 FROM table1 WHERE condition`
		`UNION ALL SELECT col2 FROM table2`
		`WHERE condition;`

10	GROUP BY with JOIN	`SELECT table1.col1, COUNT(table2.col2)`
		`FROM table2`
		`JOIN table1`
		`ON table1.col3 = table2.col2`
		`GROUP BY col1;`

11	WHERE EXISTS	`SELECT col1 FROM table1`
		`WHERE EXISTS`
		`(SELECT col2 FROM table2`
		`WHERE condition);`

12	'IN' operator second SELECT	`SELECT * FROM table1` `WHERE col2 IN (SELECT col2 FROM table2);`

13	ANY syntax	`SELECT col1 FROM table1`
		`WHERE col3 = ANY`
		`(SELECT col2 FROM table2`
		`WHERE condition);`

14	ALL syntax challenging	`SELECT col1 FROM table1`
		`WHERE col3 = ALL`
		`(SELECT col2 FROM table2`
		`WHERE condition);`

16	COUNT, SUM, AVERAGE	`SELECT table1.col1,SUM(table2.col2) FROM table1`
		`JOIN table2`
		`ON table1.col2 = table2.col3`
		`GROUP BY table1.col1;`

Table A1.2: SQL levels and questions for domain 2 with description of topics and template SQL examples

Domain Three:

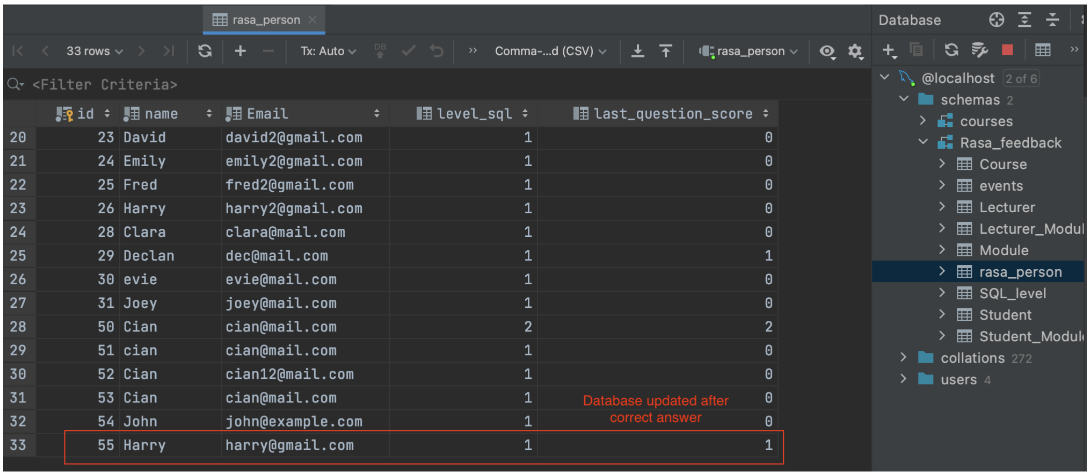
<table>
<tr><th>Level</th><th>Description</th><th>Outline</th></tr>
<tr><td>1</td><td>JOIN</td><td><code>SELECT table1.col1, table2.col2
FROM ((table3
INNER JOIN table1 ON table1.col3 = table3.col4)
INNER JOIN table2 ON table2.col5 = table3.col6);</code></td></tr>
<tr><td>2</td><td>JOIN with WHERE</td><td><code>SELECT table1.col1, table2.col2
FROM ((table3
INNER JOIN table1 ON table1.col3 = table3.col4)
INNER JOIN table2 ON table2.col5 = table3.col6)
WHERE condition ;</code></td></tr>
<tr><td>3</td><td>COUNT, SUM, AVERAGE</td><td><code>SELECT table1.col1, SUM(table2.col2)
FROM ((table3
INNER JOIN table1 ON table1.col3 = table3.col4)
INNER JOIN table2 ON table2.col5 = table3.col6)
GROUP BY table1.col1;</code></td></tr>
<tr><td>4</td><td>CASE Statements</td><td><code>CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;</code></td></tr>
</table>

Table A1.3: SQL levels and questions for domain 3 with description of topics and template SQL examples

A2 Appendix

A2.1 Conversation Examples

(a) Conversation with first time user and chatbot



(b) Database updated after conversation

Figure A2.1: Annotated conversation with a first time user including database figure

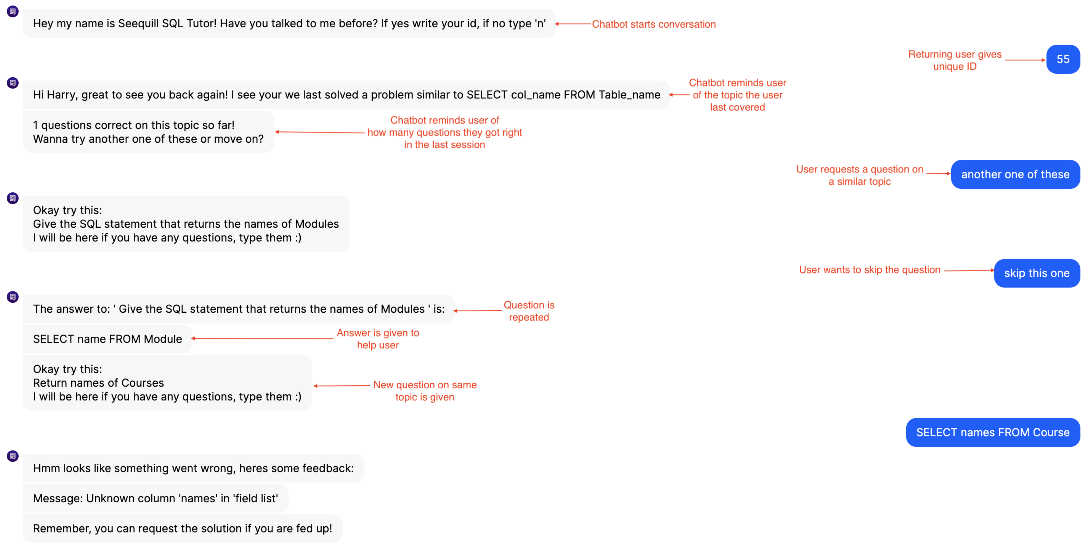


Figure A2.2: Annotated conversation with a returning time user showing a user skipping a question and receiving feedback on a wrong answer

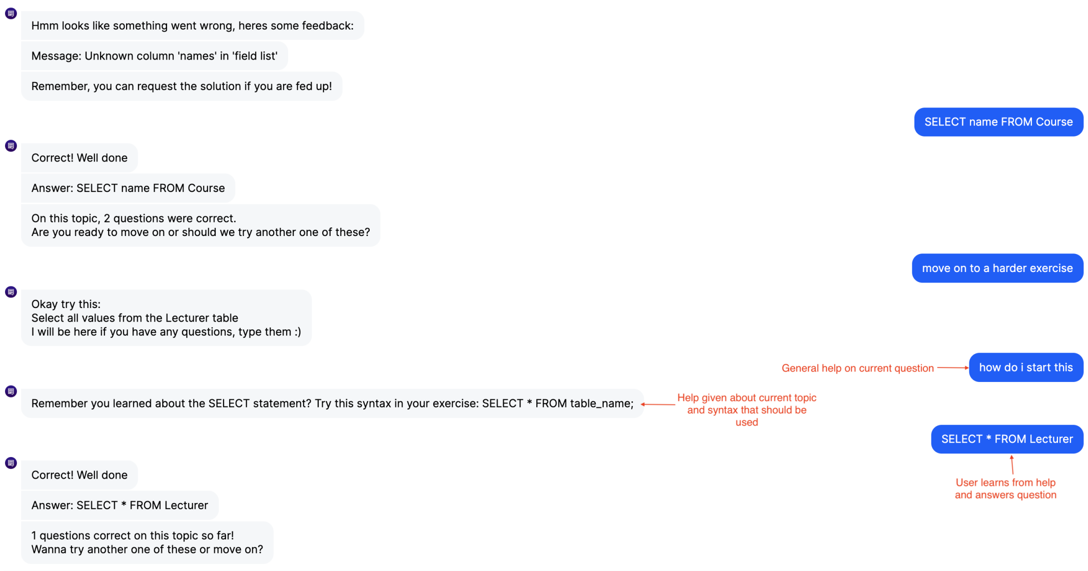


Figure A2.3: Annotated conversation with a returning time user showing a user asking the chatbot for help on the current question

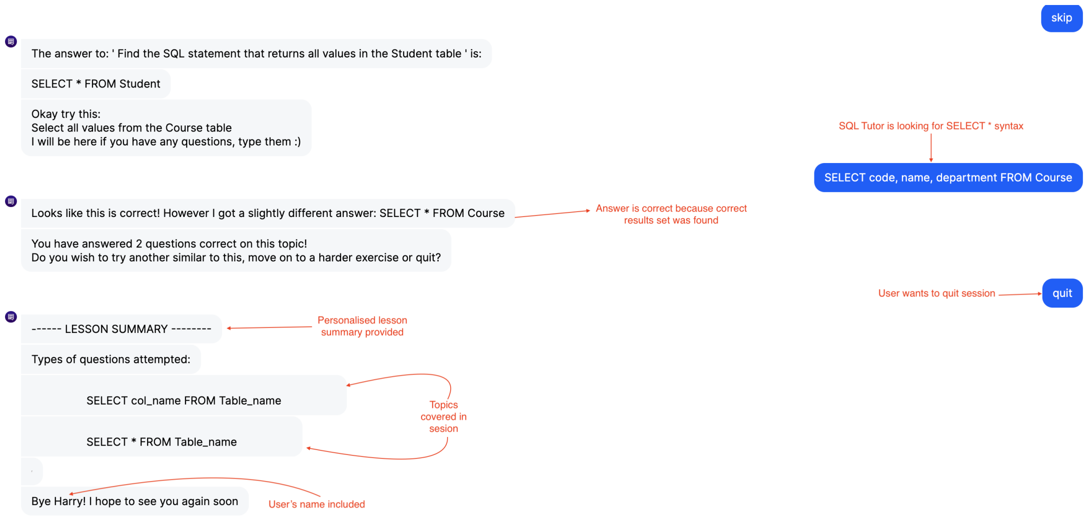


Figure A2.4: Annotated conversation with a returning time user showing a user asking the chatbot for help on the current question

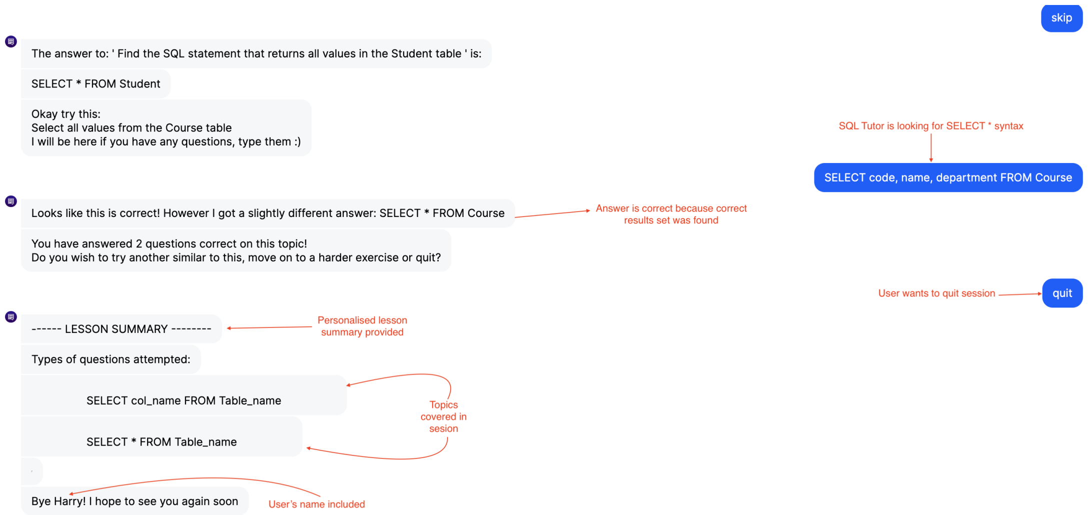


Figure A2.5: Conversation with a returning user showing chatbot reply when answer is nearly correct and showing lesson summary

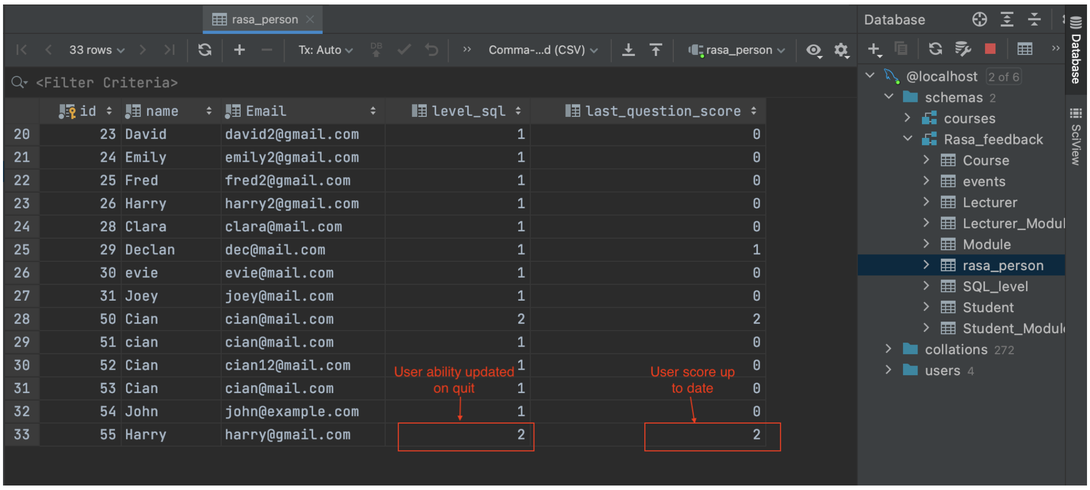


Figure A2.6: Database updated after conversation to show student's progress

A3 Appendix

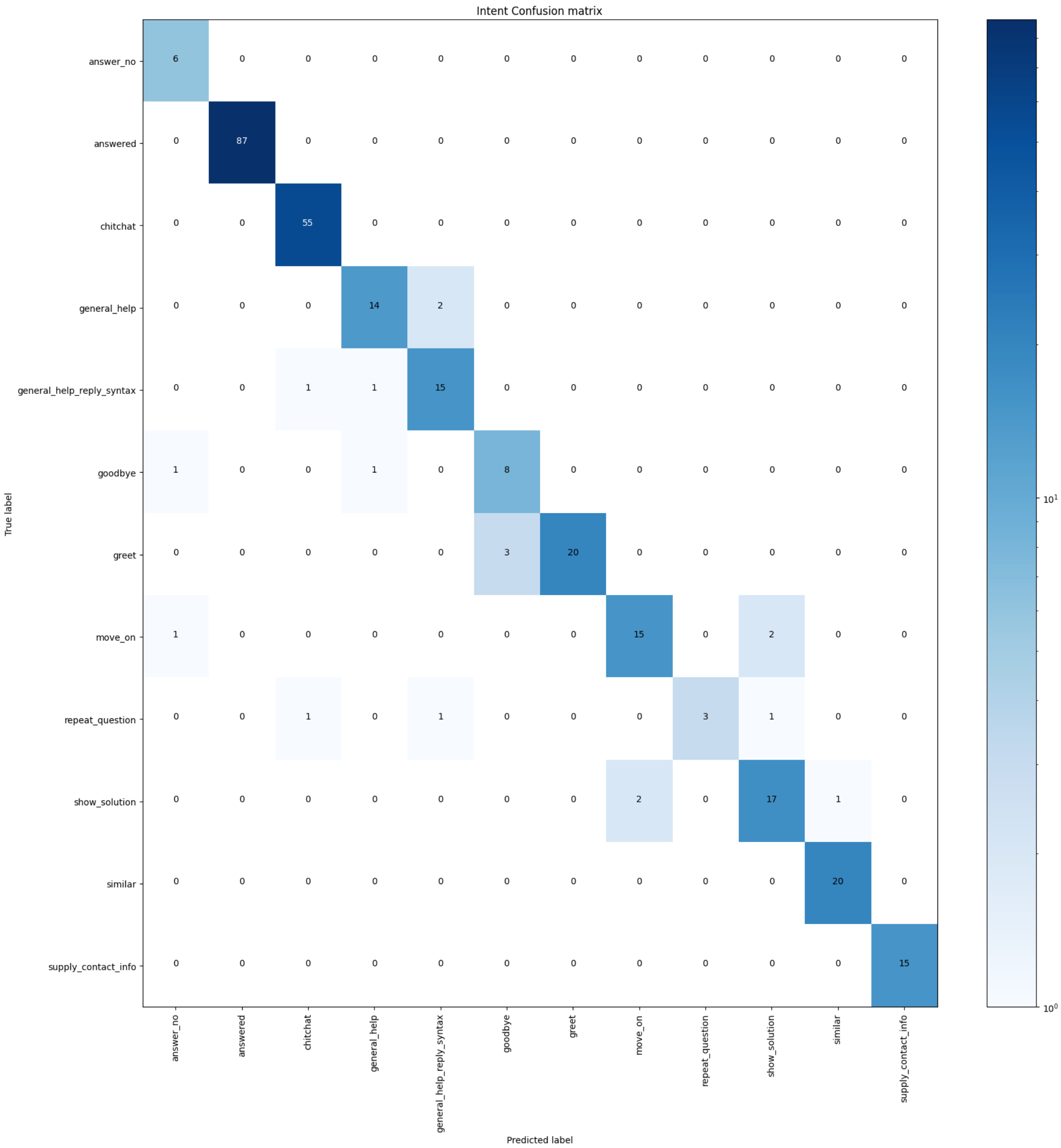


Figure A3.1: Confusion matrix for intent classification during 5-fold cross validation for the Rasa pipeline model

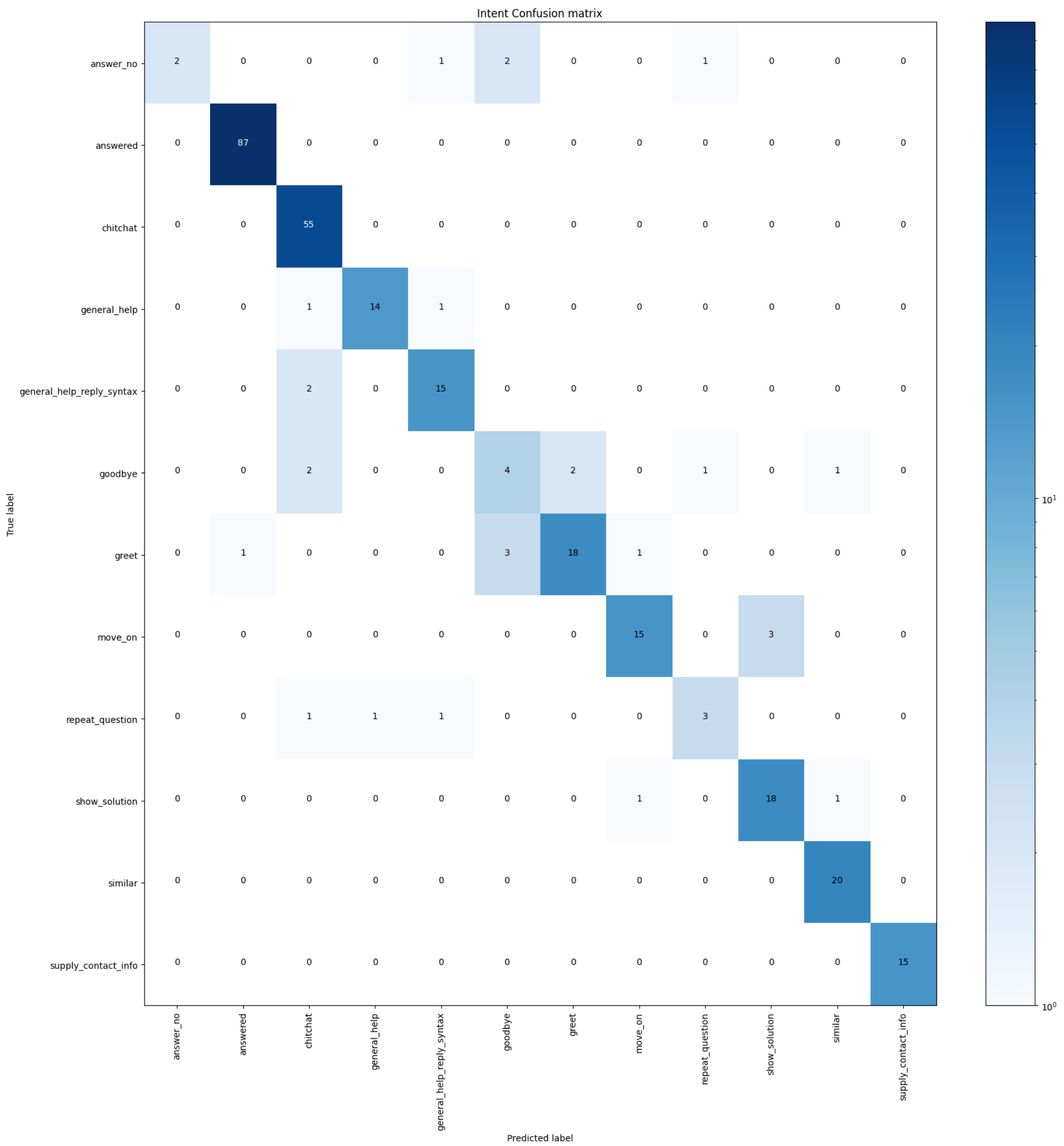


Figure A3.2: Confusion matrix for intent classification during 5-fold cross validation for the Spacy pipeline model

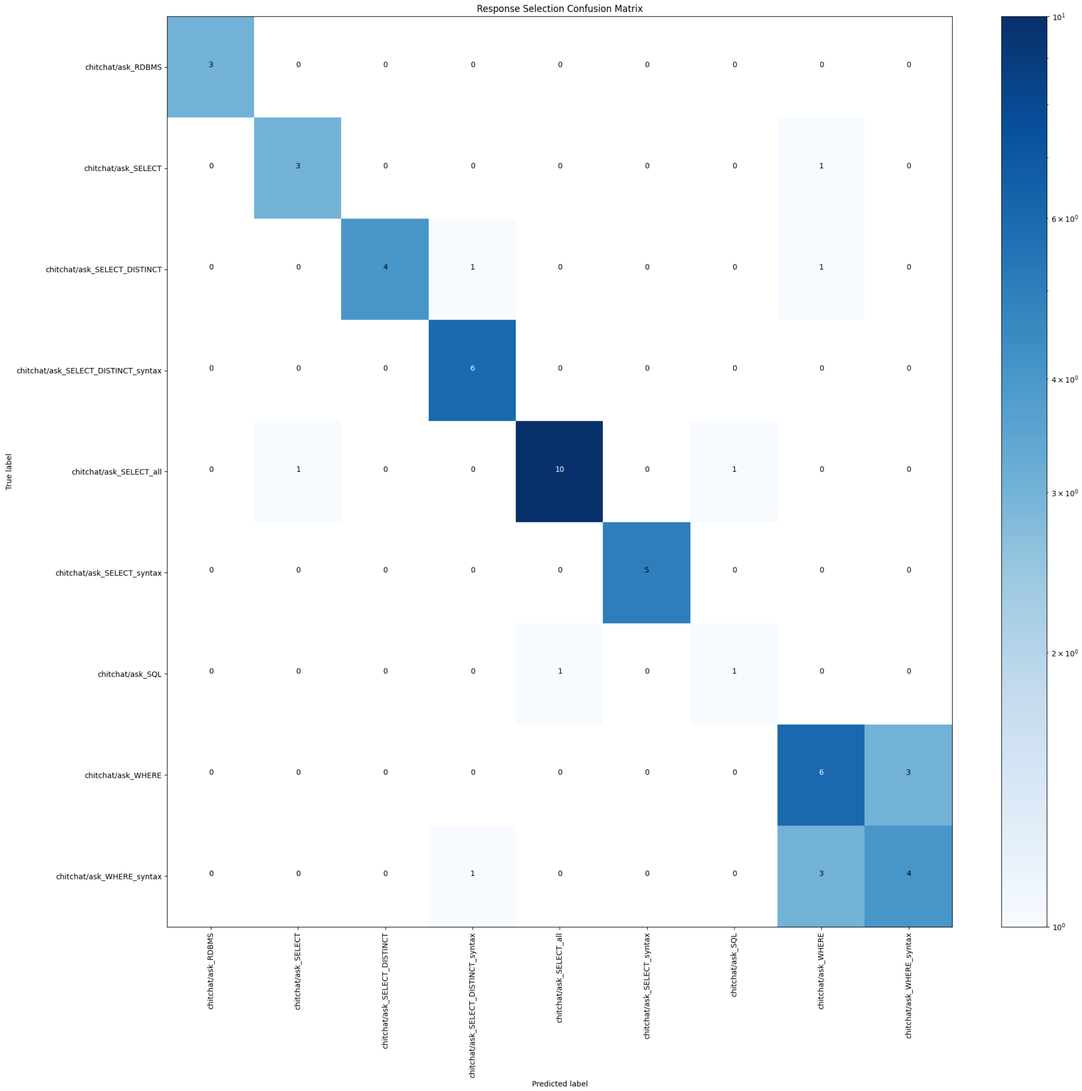


Figure A3.3: Confusion matrix for response selection during 5-fold cross validation for the Rasa pipeline model

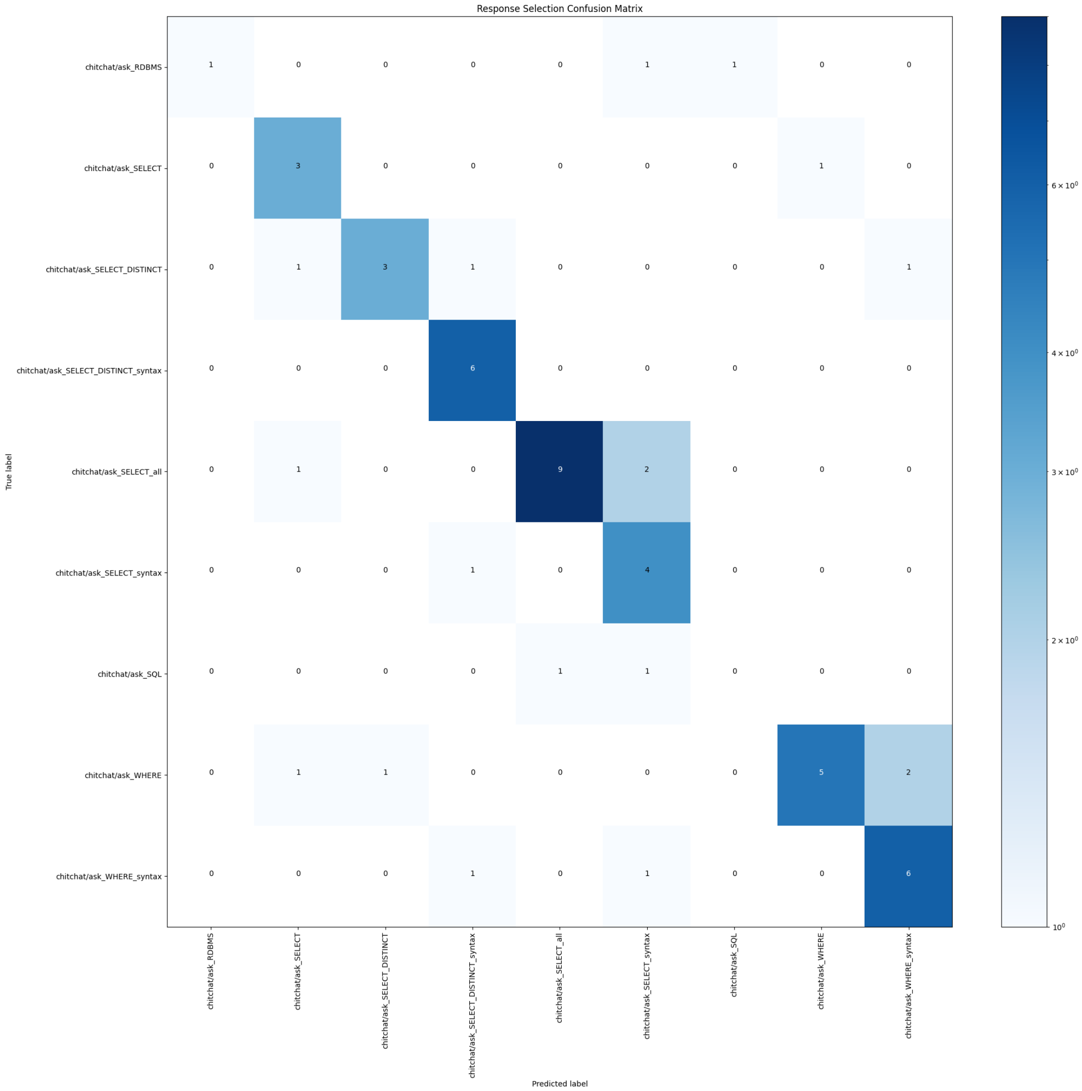


Figure A3.4: Confusion matrix for response selection during 5-fold cross validation for the Spacy pipeline model