Software Engineering for Chatbots/Conversational Agents

Subtitle: Building a Testing System for Chatbots

Master's Thesis submitted to the

Constructor Institute

in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Software Engineering

presented by

Mohammad Nadeem Ghafoori

under the supervision of

Prof. Manuel Oriol

June 2023

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Mohammad Nadeem Ghafoori
Schaffhausen, 26 June 2023

To my beloved parents

“Our greatest weakness lies in giving up.
The most certain way to succeed is
always to try just one more time.”

Thomas A. Edison

Abstract

This master's thesis presents an innovative methodology for evaluating the performance of chatbots, focusing primarily on the accuracy and response time of the bot when presented with both original and paraphrased questions. The basis of our research is the development of a robust testing system. We have introduced a unique twist to the testing process by employing paraphrased questions alongside the original ones. This distinctive approach allows us to assess the chatbot's ability to comprehend and respond accurately to questions that convey the same intent but are structured differently. The system works by feeding a set of original and paraphrased questions to the chatbot and subsequently capturing and analyzing the responses. Through this research, we aim to contribute to the broader discussion on AI behavior, particularly as it pertains to the nuances of language comprehension and the implications of these systems in diverse real-world scenarios. We believe this work lays a solid foundation for further exploration in the field of chatbot evaluation and performance optimization.

Acknowledgements

First of all, I am thankful to God, who gave me the strength and the ability to complete my thesis, without his divine help I would have never been able to complete it.

I would like to express my sincere appreciation and thanks to my advisor, Prof. Manuel Oriol, who greatly supported me in completing my work and providing a lot of valuable and practical suggestions during this period. I will be grateful for his assistance and guidance forever.

I would also like to thank Prof. Bertrand Meyer and Prof. Mauro Pezze for their continuous support, encouragement, and teachings that contribute to my self-development, knowledge, and attitude towards accomplishing the objectives of this project.

Last but not least, I would like to thank my highly respected mother and father. Without their constant support and encouragement, particularly in difficult times, the whole endeavor of writing my thesis would not have been fruitful.

Contents

Figures

Tables

Listings

Chapter 1

Introduction

In this chapter, we explore the challenges in ensuring chatbot quality, critically review existing testing methodologies, and highlight their limitations. We then introduce a new perspective on enhancing chatbot testing techniques to improve user experience.

1.1 Background

In recent years, chatbots, also known as conversational agents, have gained popularity and provide businesses with an efficient and convenient method of interacting with their clients. However, it is difficult to guarantee the quality of these chatbots since they imitate human conversation and therefore involve a wide range of possible errors and problems. As chatbots often involve complex natural language processing and machine learning algorithms that are difficult to test, traditional software testing techniques may not be sufficient for testing chatbots. Moreover, changes in user behavior, language use change or updating of underlying technologies can affect the quality of chatbots over time. Therefore, in order to guarantee the quality of chatbots during their maintenance cycle, there is a need for more efficient and effective testing methods.

Consider an example in which users interact with chatbots for purchasing a product. Based on user preferences and requirements, the chatbot provides recommendations for products. However, the recommendation is not satisfactory to the user and they request an alternative from the chatbot. Then the chatbot will respond with an irrelevant recommendation, which can lead to a frustrating user experience. The importance of testing and quality assurance for chatbots is highlighted in this scenario. It is necessary to have reliable testing procedures in order to ensure the functionality, accuracy, and efficiency of chatbots which are rapidly becoming popular across various sectors like customer services, e-commerce, and healthcare. A poorly performing chatbot can lead to a loss of income and customer complaints, as well as an impact on the law in some sectors. Therefore, there is a critical need for effective chatbot testing methods to ensure their long-term success.

1.2 Possible Solutions

Various methods and testing tools are available for chatbots, such as the use of manual or automated testing. Testim [19], TestCraft [13], Test.ai [18], and Botium [3] are some of the most popular testing frameworks for chatbots. Testim [19] is a testing tool, which supports multiple communication channels for creating and running end-to-end tests on chatbots using AI. TestCraft [13] provides a cloud based testing platform allowing users to build and execute tests on chatbots using an intuitive visual test builder. Test.ai [18] is an automation platform that makes automated chatbots' tests using AI, making it easier to detect bugs and errors in a timely manner. Another chatbot testing framework is Botium [3] which allows testing chatbots with multiple messaging channels such as Facebook Messenger, Slack, and Skype. Moreover, there are also formal verification techniques and software used for testing chatbots. One example is the RoboSherlock system [2], which uses formal methods to verify the performance and functionality of chatbots.

Although these available methods and software have proven to be effective in testing chatbots, they are still limited. For instance, traditional testing methods may not be able to deal with the complexity and variations of user inputs, while formal verification techniques could cost a lot and take valuable time. Furthermore, the current testing frameworks may not be flexible enough to deal with changes in chatbot's design and functionality. These limitations highlight the need for alternative approaches to testing chatbots.

1.3 Proposed Solution

In order to test chatbots more comprehensively, we propose integrating PEGASUS pre-trained model [22] developed by Google Research into the testing process. This can be achieved by utilizing PEGASUS fine-tuned model [20] for paraphrasing, which allows for advanced natural language processing capabilities. With this integration, we aim to test the chatbot's ability to understand and respond to a wide range of inputs and topics.

To implement this solution, we will identify a chatbot. We will then create a set of test questions based on various topics and paraphrase them using the PEGASUS model [20]. We will input the original and paraphrased versions of the test questions into the chatbot, and recording the chatbot's responses to each question. This allows for a comprehensive evaluation of the chatbot's ability to understand and respond to different types of user input.

We will then compare the responses to the original test questions and the paraphrased versions to see how well the chatbot performs in terms of accuracy and response time. We will then be able to determine the chatbot's strengths and weaknesses as well as areas for improvement. Our proposed solution aims to provide a more comprehensive and efficient method for testing chatbots by utilizing advanced natural language processing capabilities to guarantee chatbot quality and enhance user experience.

Chapter 2

State of the Art

In this chapter, we comprehensively review the state of the art in chatbot testing. We explore significant studies in the field, existing testing frameworks and tools, as well as the prevalent formal verification techniques.

2.1 Review of Related Studies

In the study "A Statistical Simulation Technique to Develop and Evaluate Conversational Agents" by David Griol, Javier Carbo, and José M. Molina, the authors propose a technique for creating user simulators that interact with and assess conversational agents [7]. This method is built upon a statistical model that is automatically derived from a dialogue corpus. This model then guides the user simulator to generate subsequent responses, keeping in mind the complete interaction history. Griol, Carbo, and Molina applied their approach to the development and evaluation of a conversational agent within a multi-agent system, which provides academic information [7]. Their findings suggest that this strategy not only evaluates chatbot performance but also enables the exploration and adoption of enhanced dialogue strategies. This leads to the conversational agent reducing the time taken to complete dialogues and automatically identifying new valid paths to reach predefined task objectives [7]. However, the study does not elaborate on the adaptability of this methodology when faced with complex and diverse user inputs, which are typically unpredictable in real-world scenarios. Moreover, as the focus of the study was a specific use case (providing academic information), it brings into question the methodology's applicability across different domains and tasks [7].

In the study titled "Privacy Concerns in Chatbot Interactions", Carolin Ischen, Theo Araujo, Hilde Voorveld, Guda van Noort, and Edith Smit delve into a largely unexplored area privacy concerns related to the use of chatbots [9]. The authors note that while chatbots are increasingly deployed in commercial contexts to make product or service recommendations, research into privacy concerns arising from chatbot interactions is scarce. The study sets out to examine how the presence of human-like cues in chatbots affects user perceptions of anthropomorphism (attributing human characteristics to non-human entities), privacy concerns, information disclosure, attitudes, and adherence to recommendations [9]. The researchers found that a chatbot with human-like features leads to more information disclosure and adherence to recommendations, mediated by an increased perception of anthropomorphism and consequently, lower

privacy concerns, compared to a machine-like chatbot. However, this outcome does not hold when compared to a website; users perceived both a human-like chatbot and a website as high in anthropomorphism [9]. These results underscore the importance of both anthropomorphism and privacy considerations in influencing user attitudes and behaviors when interacting with chatbots. Nonetheless, the paper could have elaborated more on how these findings might impact chatbot design and deployment across various domains [9]. Additionally, it did not address potential variations in user responses across different demographics and cultures, which could be critical for a comprehensive understanding of privacy concerns in chatbot interactions.

The study titled "A Comparison of Natural Language Understanding Platforms for Chatbots in Software Engineering" by Ahmad Abdellatif, Khaled Badran, Diego Elias Costa, and Emad Shihab evaluates different Natural Language Understanding (NLU) platforms with a focus on their application in software engineering chatbots [1]. The authors recognize that chatbots, powered by NLUs, are set to significantly transform software engineering by allowing practitioners to inquire about their projects and interact with services using natural language. However, choosing the best NLU for software engineering chatbots is an open challenge [1]. The authors assessed the performance of four of the most widely used NLUs: IBM Watson, Google Dialogflow, Rasa, and Microsoft LUIS. The evaluation metrics included the platforms' abilities in classifying intents, the stability of confidence scores, and the extraction of entities [1]. Two datasets reflecting common tasks performed by software engineering practitioners were used for evaluation: one dataset for chatting with a chatbot to ask about software repositories, and another for asking development questions on Q&A forums like Stack Overflow [1]. According to their findings, IBM Watson outperformed the other platforms in terms of overall performance. However, when the results were broken down by each individual aspect, it was revealed that while IBM Watson had the best intent classification performance, Rasa outperformed the others with higher median confidence scores. For entity extraction, both IBM Watson and Microsoft LUIS outshone the others [1]. Though this study provides valuable insights for software engineering practitioners deciding on an NLU for their chatbots, it could have incorporated a more diverse set of evaluation metrics and usage scenarios for a comprehensive comparison [1]. Additionally, the study's findings might be less applicable to chatbots outside the software engineering context, indicating a potential gap for future research.

The paper "Chatbots for Customer Service: User Experience and Motivation" by Asbjørn Følstad and Marita Skjuve delves into the user experience and motivation behind using chatbots in customer service, a field with little in-depth research despite growing application [6]. The authors interviewed 24 users of two customer service chatbots, examining their experiences and motivations [6]. Følstad and Skjuve's findings underscore the value of chatbots in providing efficient and appropriate responses to simple queries. Interestingly, they found that occasional failures of chatbots to provide adequate answers did not necessarily lead to a negative user experience, provided the chatbot facilitated an easy transition to human customer service representatives for follow-up [6]. This finding somewhat counters prevalent notions about users' expectations of chatbots, suggesting that users have more realistic anticipations of chatbot capabilities [6]. Another noteworthy discovery was that the human-like characteristics of customer service chatbots, while potentially relevant to user experience, were significantly less important than the chatbots' ability to handle inquiries efficiently and adequately [6]. This observation offers a contrasting perspective to the common emphasis on making chatbots more "human-like" for improved user experience [6]. While the study provides valuable insights into user

experience and motivation, it may have benefitted from a larger, more diverse sample size for generalizability. Moreover, the study focused on two specific chatbots, which might limit the application of its findings to other chatbots with different designs or functionalities [6]. Future research could aim to address these limitations and extend the understanding of user interaction with various types of chatbots.

In the paper "Survey of Various AI Chatbots Based on Technology Used", authors Siddhant Singh and Hardeo Thakur explore the progression and variations in chatbot technology, highlighting the significant role of AI, particularly Natural Language Processing (NLP) and Machine Learning [14]. Starting from the first chatbot, ELIZA, introduced in 1966, the authors detail the evolution of different versions of chatbots utilizing a myriad of technologies and approaches [14]. The researchers delineate the generic workflow of chatbots, illustrating how NLP aids chatbots in understanding natural language more clearly to generate intelligent responses [14]. They compare different chatbot technologies based on various parameters, providing insights into their operational dynamics [14]. The authors emphasize that for more efficient natural language conversation, a chatbot must analyze and understand the user input accurately to provide a relevant response [14]. Singh and Thakur also underscore the expanding application of chatbots across vital domains such as science, education, healthcare, and more, suggesting the potential of chatbots to enhance human interaction with machines [14]. However, this paper, while offering a comprehensive overview, could have delved deeper into each technology's specific strengths, weaknesses, and opportunities for further improvements [14]. Such nuanced analysis would have offered more insights for the ongoing development of chatbot technologies.

2.2 Existing Testing Frameworks and Tools

In this section, we delve into the details of several existing frameworks and tools used for chatbot testing (see table 2.1). We discuss their creators, functionality, strengths and weaknesses, and potential issues that may arise from their use in chatbot testing.

Framework/ Tool	Creator	Function and Features	Pros	Cons	Limitations
Testim [19]	Oren Rubin	An AI-based software testing tool that enables end-to-end testing. Features include automatic test maintenance, fast authoring, data-driven testing, root cause analysis.	High speed and reliability. Good AI-based maintenance. Excellent support and documentation.	Can be pricey for small organizations. Certain complex scenarios can still require manual testing.	Its focused more on web application testing, thus might not cater to the unique needs of chatbot testing.
TestCraft [13]	Dror Todress	A codeless Selenium test automation platform. Allows manual testers to create test automation flows for web apps. Features include AI-based maintenance, visual modelling, and data-driven testing.	No coding skills required, hence lowering the entry barrier for testers. Good AI-based maintenance.	Can be complex to set up. Some users report issues with stability.	Not specifically designed for chatbots; some unique aspects of chatbot testing might not be covered.
Test.ai [18]	Jason Arbon	AI-powered testing that can work across different platforms. It uses machine learning to identify elements and validate user flows.	Uses machine learning effectively to identify elements. Can work across multiple platforms.	Its still relatively new and might have teething problems. The AI might misinterpret some elements.	Its not specifically designed for chatbots; some unique aspects of chatbot testing might not be covered.

Continued on next page

Framework/ Tool	Creator	Function and Features	Pros	Cons	Limitations
Botium [3]	Florian Treml	Automates conversations with your chatbot using any channel. Provides support for most chatbot technologies and includes features like NLP assertions and scripted conversation testing.	Simple setup, broad technology support, and its Conversation Testing Script (CTS) provides a powerful tool for scripted testing.	It's relatively new and less well-documented compared to more established testing tools. Requires a level of technical know-how to be used effectively.	Chatbot testing heavily depends on the quality of the scripts, which need to be carefully crafted for comprehensive testing.

Table 2.1. Overview and Comparative Analysis of Existing Testing Frameworks and Tools

2.3 Formal Verification Techniques

This section provides a detailed overview of popular formal verification methods used in the field (see table 2.2). Formal verification techniques are good at ensuring the reliability and functionality of chatbot systems. But it is not exclusively designed for chatbots, and it requires deep understanding for effective use.

Technique	Creator	Description	Pros	Cons	Limitations
RoboSherlock [2]	Michael Beetz and his team at Bremen University	An open-source framework for automatic analysis of 3D sensor data. It employs knowledge processing methods to interpret the data and generate conclusions.	RoboSherlock can handle complex data structures, making it versatile for various testing scenarios. It is open-source, so it is accessible to everyone.	It requires significant computational resources due to the complexity of 3D data analysis. It may not be suitable for simple chatbot systems.	The interpretation of results may need domain-specific expertise.

Continued on next page

Technique	Creator	Description	Pros	Cons	Limitations
Spin Model Checker [16]	Gerard J. Holzmann at Bell Labs	An open-source software tool that is used for the formal verification of distributed software systems. Spin Model Checker is used to verify concurrent systems and detect synchronization issues, deadlocks, and safety property violations.	It is particularly useful in testing chatbots with complex concurrent processing. It can also detect obscure deadlocks and other synchronization issues that can be missed in manual testing.	Requires deep knowledge about the system and its interactions. Also, it might be overkill for simple chatbots.	Wrong interpreting of results can lead to inappropriate modifications in the system.
UPPAAL [21]	Uppsala University and Aalborg University	A tool for modeling, simulation, and verification of real-time systems.	Allows testing under different real-time scenarios, which provides comprehensive test coverage.	Its application requires substantial expertise in real-time system modeling. Not suitable for chatbots that don't have real-time interactions.	A steep learning curve might be a hindrance to effectively using UPPAAL.

Table 2.2. Overview and Comparative Analysis of Formal Verification Techniques

Chapter 3

Model

In this chapter, we present the architecture of a testing system for chatbots. The architecture is a client-server model designed to enable seamless communication between the user and the system. We begin by providing the system's architecture using a detailed diagram and discussing the interconnections and interactions between the different components. Next, we elaborate on communication protocol that will connect the chatbots with our system. The chapter will be concluded by describing different technologies which will be used in implementing the system.

3.1 High-Level View of the System

The diagram provides an overview of the system's architecture, illustrating how the client-side and server-side components collaborate to enable chatbot testing (see Figure 3.1).

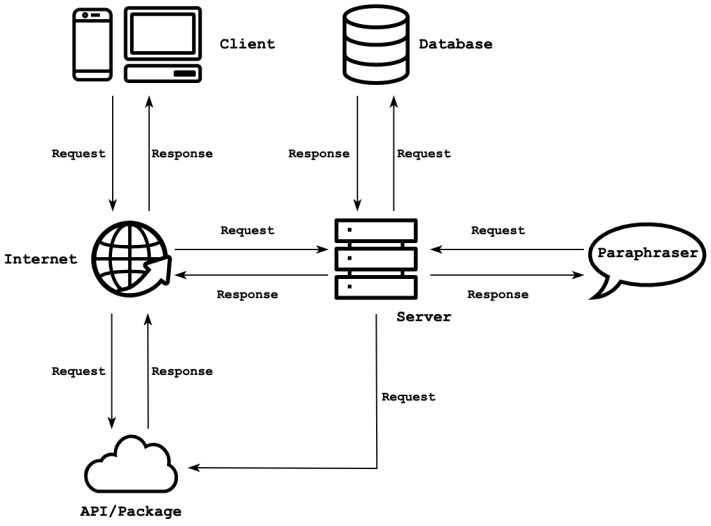

Figure 3.1. High-Level View of the System

The proposed system follows a client-server architecture. The system comprises two major components: the client-side and the server-side. The client-side is responsible for allowing the user to input test questions and paraphrasing them using the PEGASUS model [20]. The server-side is responsible for hosting the chatbot to be tested and comparing its responses to the original and paraphrased test questions.

The client component serves as an interface for the user, allowing them to input test questions, generate paraphrases, and evaluate the chatbot's performance. It is connected bidirectionally to the internet component, sending requests from the client to the internet and receiving responses back. This connection enables the client to access the system's functionalities through the internet.

The internet component acts as a medium for communication between various components in the system. It is connected bidirectionally to both the client and the server components. The requests from the client, such as test questions, are sent through the internet to the server component. The client will receive back via the Internet responses from the server, such as chatbot evaluation results. The client and server can exchange data and information more easily through this connection.

The server component plays a central role in the system's architecture. It is connected bidirectionally to the internet, receiving requests from the internet and sending responses back. The server serves as the backend of the system, processing test questions, generating paraphrases, interacting with the chatbot whether through its API or as an integrated package, and accessing the database. It orchestrates the entire testing process, coordinating the interactions between the different components.

The API/package component is connected unidirectionally to the server component. It enables the server to access the chatbot either through an API or as an installed package, depending on the integration method provided by the chatbot, for testing purposes. The server sends requests to the chatbot, which include the original and paraphrased versions of the test questions, and receives responses that contain the chatbots generated responses. This connection enables the server to interact with the chatbot and retrieve its outputs for evaluation.

The database component is connected bidirectionally to the server. The server sends requests to the database component to store and retrieves data such as test questions and paraphrases. The component of the database responds to such requests by providing the required data. This connection ensures the persistence of the testing data and facilitates data management and retrieval.

The paraphraser component is connected bidirectionally to the server. It sends requests to the server, requesting the paraphrasing of test questions. These requests are processed by the server, paraphrasing is performed using the paraphraser component and responses with the paraphrased versions of the test questions. This connection allows for the transformation of test questions into paraphrased versions.

3.2 Communication Between the System and Chatbots

Our system uses a flexible approach for interacting with chatbots, either through a RESTful API or directly incorporating them as installed packages, depending on the specific provisions of the chatbot. For those chatbots that provide a RESTful API, the original and paraphrased versions of test questions are dispatched from the client-side to the server-side. Here, the chatbot API processes these questions and the responses are subsequently sent back to the client-side for further analysis. In cases where a chatbot can be integrated directly via a package installation, it is incorporated into the environment of the system. Similar to the API-based model, the original and paraphrased test questions are processed and the responses evaluated.

3.3 Technologies Employed in the Development

In the development of the testing system for chatbots, several technologies are employed to facilitate its implementation and functionality. These technologies include:

- **Python Programming Language:** Python [12] is the chosen programming language for the development of the testing system for chatbots. Python's simplicity, extensive library ecosystem, and strong support for natural language processing make it well-suited for this task. It allows for efficient implementation of the system's components and easy integration with external libraries and APIs.
- **Flask Framework:** Flask [5] is a Python framework for building web applications which is an easy and flexible framework. We use Flask to create the web-based interface of the client-side component. It helps us to make a user-friendly interface.
- **JavaScript Programming Language:** JavaScript [10] is also used in the development process. It adds interactivity to the client-side component, facilitating dynamic updates and responsive elements in the web interface. Its event-driven nature makes it ideal for creating seamless user interactions. JavaScript supports numerous libraries, enhancing the functionality of the system and contributing to a user-friendly experience.
- **SQLite Database:** SQLite [17] is employed as the Database Management System for the chatbot testing system. SQLite is a lightweight, serverless, and self-contained relational database engine that operates directly from the application. It offers simplicity, flexibility, and portability, making it a suitable choice for small to medium-sized applications like the testing system for chatbots. SQLite allows for efficient storage and retrieval of test questions and paraphrased versions, while requiring minimal setup and administration.

Chapter 4

Implementation

This chapter provides an in-depth overview of the practical aspects of the research, detailing the process of implementing the system designed to evaluate chatbot performance. It delineates the selection and setup of the chatbot and the PEGASUS model [20], design of test questions, paraphrasing process, and the testing mechanism, along with the metrics used to test chatbot response quality. The comprehensive implementation steps coupled with illustrative code snippets ensure an exhaustive understanding of the project execution.

4.1 Chatbot Selection

In the search for an appropriate chatbot to evaluate, we chose an open-source chatbot called ChatBotAI [8] available on GitHub for a number of reasons. This chatbot stood out from others due to its easy integration via pip, which was vital for integrating with the test system being developed. With pip, the chatbot could be installed directly into the project environment from the PyPI [11]. This made it feasible to automate the feeding of test questions and the capturing of chatbot responses, a core requirement for this project.

In addition, the credibility and reliability of this chatbot were supported by its popularity and usage statistics on GitHub. The chatbot ranked high under the 'best match' sorting feature of GitHub, indicating its relevance and utility in the developer community. This was further corroborated by the number of stars it had received, totalling 549, and the fact that it had been forked 243 times. The large number of stars and forks suggested that it had been widely used by other developers, ensuring a certain level of quality and robustness. Lastly, the chatbot was under an MIT license, which allowed for freedom of modification and usage without worrying about copyright infringement. This ensured that the chosen chatbot was suitable for rigorous testing and potential modifications, if needed, without any legal limitations.

4.2 Setting up the Model

The PEGASUS model [20] used for this study was set up using the Hugging Face Transformers library [4], which provides a variety of pre-trained models for Natural Language Processing tasks. The PEGASUS model [20] we employed was specifically fine-tuned for paraphrasing

tasks.

```
# Initializing the tokenizer and model.
tokenizer = AutoTokenizer.from_pretrained("tuner007/pegasus_paraphrase")
model = AutoModelForSeq2SeqLM.from_pretrained("tuner007/pegasus_paraphrase"
    ↪ )
```

Listing 4.1. Initialization of the Model and Tokenizer

The setup involved initializing a *tokenizer* and a *Seq2Seq* model with the pre-trained parameters (see Listing 4.1). The *tokenizer* was used to convert our test questions into a format that the model can understand, i.e., tokenized form.

```
def paraphrase(question):
   # Prefixing the text with "paraphrase: " to guide the model to generate
       ↪ paraphrase.
   text = "paraphrase: " + question
   # Defining the maximum length of the generated paraphrase.
   max_len = 100

   # Encoding the input text using the tokenizer.
   encoding = tokenizer.encode_plus(
      text, pad_to_max_length=True, return_tensors="pt")
   input_ids, attention_masks = encoding["input_ids"], encoding["
       ↪ attention_mask"]

   # Using the model to generate paraphrases of the input text.
   outputs = model.generate(
      input_ids=input_ids, attention_mask=attention_masks,
      max_length=max_len, num_beams=10, num_return_sequences=1, temperature
          ↪ =1.5
   )

   # Decoding the output from the model and returning it as a string.
   return tokenizer.decode(outputs[0], skip_special_tokens=True,
       ↪ clean_up_tokenization_spaces=True)
```

Listing 4.2. Implementation of Paraphrase Generation Function

The paraphrasing function was then implemented (see Listing 4.2). A text input was prefixed with "*paraphrase:* ", this was a simple yet effective way to instruct the model about the required task - to paraphrase the input text. The encoded version of this prefixed text was then used as input to the model. Padding was applied to ensure a consistent sequence length. The model generated paraphrases of the input text using the *model.generate()* function. The *num_beams* parameter was set to 10 for beam search, which allowed the model to explore a wider range of potential outputs. The *num_return_sequences* parameter was set to 1, so as to get a single unique paraphrase for each input. The *temperature* parameter was set to 1.5, controlling the randomness in the output. Higher values corresponded to more random outputs, providing a

wider range of paraphrasing styles. The output from the model, which was in tokenized form, was then decoded back into a string, with special tokens and unnecessary spaces being cleaned up in the process.

This process allowed for an efficient and effective integration of the PEGASUS model [20] with the testing process. The flexibility of the model allowed for generating diverse paraphrases, contributing to a more robust testing environment for the chatbot.

4.3 Designing the Test Questions

The process of designing the test questions was integrated into a Flask-based web application. This approach allowed us to systematically create, store, and manage the test questions. Each question was paraphrased using the model before being stored in a database. This was done to ensure that both the original and the paraphrased versions of each question could be used in the testing process.

```
@app.route('/', methods=['GET', 'POST'])
   def home():
      if request.method == 'POST':
         questions = request.form.getlist('question')
         for question in questions:
            if question:
               paraphrased_question = paraphrase(question)
               question_record = Question(
                  original=question, paraphrase=paraphrased_question)
               db.session.add(question_record)
         db.session.commit()
         return redirect(url_for('paraphrase_page'))
      return render_template('index.html')
```

Listing 4.3. Processing and Storing Paraphrased Test Questions

The code outlines the web application's core functionality (see Listing 4.3). If a *POST* request is made, the application collects the submitted questions, paraphrases them using the *paraphrase()* function, and stores both versions of the question in a database. This process was carried out for each question submitted. If a *GET* request is made, the application simply returns the main page. By adopting this structure, the process of creating, paraphrasing, and storing the questions was made simple, efficient, and easily manageable.

4.4 Paraphrasing Process

The outcome of the paraphrasing process was a list of question pairs, each consisting of an original question and its corresponding paraphrase. For each test question submitted, a single paraphrased version was created. This approach enabled us to compare the chatbot's responses to both the original and paraphrased version of each question.

```
@app.route('/paraphrase', methods=['GET', 'POST'])
   def paraphrase_page():
      if request.method == 'POST':
         if request.form.get('button') == 'clear':
            db.session.query(Question).delete()
            db.session.commit()
            return redirect(url_for('home'))
      questions = [(q.original, q.paraphrase) for q in Question.query.all()
          ↪ ]
      return render_template('paraphrase.html', questions=questions)
```

Listing 4.4. Accessing and Managing Paraphrased Test Questions

The code handles the management of stored paraphrased questions (see Listing 4.4). A *POST* request made with the *'clear'* button results in the deletion of all records from the database, while a *GET* request returns the paraphrase page with all stored original questions and their paraphrases.

Recognizing the fact that the PEGASUS model's [20] paraphrasing may not always align with the desired context or meaning, an editing functionality was introduced (see Listing 4.5). This allows for manual correction or modification of the generated paraphrases, ensuring that the test questions used are meaningful and relevant. This manual editing function was essential to account for the variability in the quality of paraphrases and to overcome the occasional misalignment of context or semantics in the paraphrasing process.

```
@app.route('/edit', methods=['POST'])
   def edit_questions():
      paraphrases = request.json.get('paraphrases')
      if paraphrases:
         questions = Question.query.all()
         for i in range(min(len(questions), len(paraphrases))):
            questions[i].paraphrase = paraphrases[i]
         db.session.commit()
         return jsonify(success=True)
      return jsonify(success=False)
```

Listing 4.5. Editing Paraphrased Test Questions

This code allows the user to send a *POST* request with a *JSON* object containing the modified paraphrases. The function then updates the paraphrases in the database accordingly, thus ensuring the flexibility and precision of the testing process.

4.5 Testing Process and Chatbot Performance Measurement

In order to link the chatbot to the testing system, it was necessary to install the chatbot module from the package via pip (see Listing 4.6).

```
@register_call("whoIs")
def who_is(query, session_id="general"):
   return "I am a bot, created for a demo."

template_file_path = os.path.join(os.path.dirname(
   os.path.abspath(__file__)), "test.template")
chat = Chat(template_file_path)
```

Listing 4.6. Integration of the Chatbot with the Testing System

The *register_call("whoIs")* decorator is then used to register a function *who_is*. This function is set to respond with "*I am a bot, created for a demo.*" when the chatbot is asked about its identity. Next, the path to the *test.template* file, which contains the chatbot's response templates, is defined using *os.path.join*. This allows the chatbot to respond to a variety of predefined queries. Finally, an instance of the *Chat* class is created, with the *template file path* passed to it, making the chatbot ready for interaction.

```
# Loading English tokenizer, tagger, parser, NER and word vectors.
nlp = spacy.load("en_core_web_md")
```

Listing 4.7. Loading spaCy's Language Model for Semantic Similarity Calculation

To evaluate the accuracy of the chatbot's responses, we employed the semantic similarity measure provided by the spaCy [15] library. spaCy's language model "*en_core_web_md*" was loaded for this purpose, which generates word vectors for a given piece of text (see Listing 4.7). These vectors can then be used to calculate the similarity between the input question and the chatbot's response, offering a measure of how accurately the chatbot understood the question.

```
@app.route('/evaluate', methods=['GET'])
   def evaluate_page():
      questions = Question.query.all()

      original_response_times = []
      paraphrase_response_times = []
      original_similarities = []
      paraphrase_similarities = []

      for question in questions:
         # Collecting original question responses and time.
         start_time = time.time()
         original_response = chat.respond(question.original)
         original_response_times.append(time.time() - start_time)
         original_similarity = nlp(question.original).similarity(
         nlp(original_response))
         original_similarities.append(original_similarity)

         # Collecting paraphrase question responses and time.
```

```
20          start_time = time.time()
21          paraphrase_response = chat.respond(question.paraphrase)
22          paraphrase_response_times.append(time.time() - start_time)
23          paraphrase_similarity = nlp(question.paraphrase).similarity(
24          nlp(paraphrase_response))
25          paraphrase_similarities.append(paraphrase_similarity)
26
27      # Computing average response times.
28      original_avg_time = sum(original_response_times) / \
29          len(original_response_times) if original_response_times else 0
30      paraphrase_avg_time = sum(paraphrase_response_times) / len(
31          paraphrase_response_times) if paraphrase_response_times else 0
32
33      # Computing average similarities.
34      original_avg_similarity = sum(
35          original_similarities) / len(original_similarities) if
              ↪ original_similarities else 0
36      paraphrase_avg_similarity = sum(
37          paraphrase_similarities) / len(paraphrase_similarities) if
              ↪ paraphrase_similarities else 0
38
39      accuracy = {"original": original_avg_similarity * 100,
40                  "paraphrase": paraphrase_avg_similarity * 100}
41      response_time = {"original": original_avg_time,
42                       "paraphrase": paraphrase_avg_time}
43
44      return render_template('evaluate.html', accuracy=accuracy,
            ↪ response_time=response_time)
```

Listing 4.8. Performance Measurement of the Chatbot

The code represents a critical component of the evaluation system (see Listing 4.8). When a *GET* request is made, the evaluation process of the chatbot starts. The system first fetches all the test questions from the database. For each of these questions, both original and paraphrased, it marks the start time, prompts the chatbot to generate a response, and notes the end time. The difference between the end time and start time provides the response time of the chatbot for each question. To test the accuracy of the chatbot, the system computes a similarity score between each question and the corresponding chatbot response using spaCy's similarity function. Subsequently, it calculates the average response times and similarity scores for both the original and paraphrased questions.

Chapter 5

Experiments

In this chapter, we present a detailed account of the experiments conducted to evaluate the performance of the chatbot in understanding and responding to both original and paraphrased questions. The setup, metrics and evaluation criteria, and the resulting outcomes are meticulously discussed.

5.1 Procedure and Setup

The experimentation process began at the home page of our system (see Figure 5.1). Users were provided with five fields to enter their original questions. The system would then paraphrase these questions upon the press of the *Paraphrase* button.

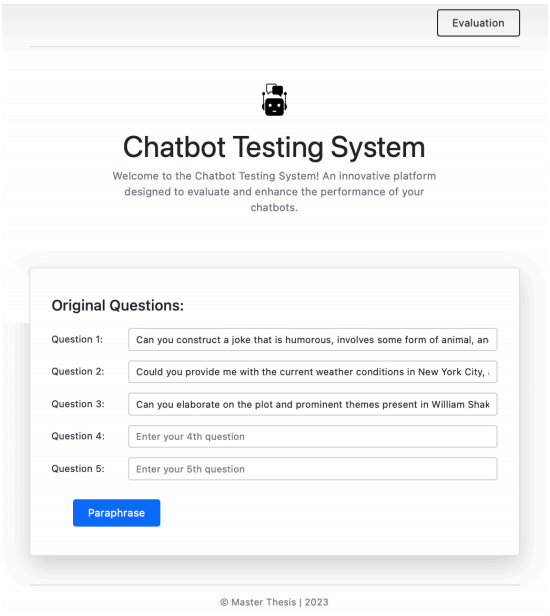

Figure 5.1. Home Page of the System

Upon clicking the *Paraphrase* button, users were taken to the paraphrase page, where the original questions and their paraphrased counterparts were displayed side by side (see Figure 5.2). A *Clear* button was provided for the complete removal of all existing data from the system.

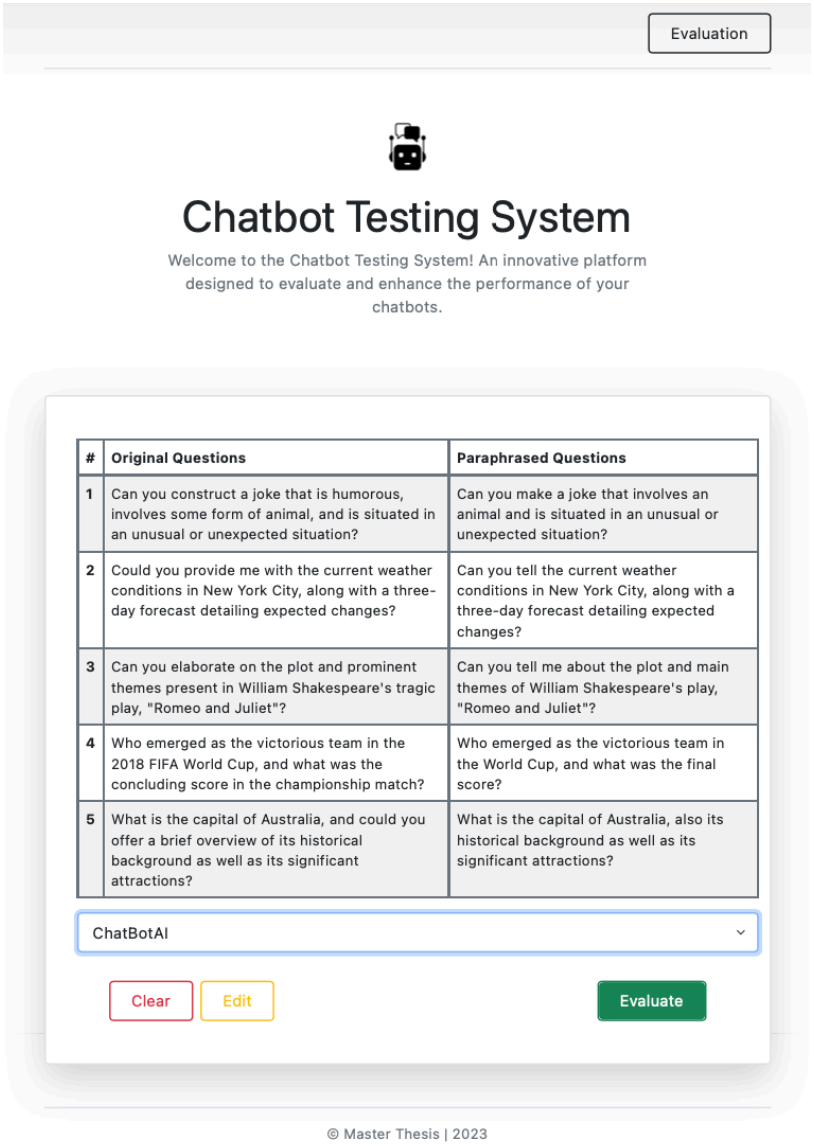

Figure 5.2. Paraphrase Page of the System

In addition, on the paraphrase page, users could select the chatbot for testing their questions. An *Edit* button was available to modify any automatically paraphrased question manually. If the *Edit* button was clicked, a modal popped up to allow for manual editing of the paraphrased questions (see Figure 5.3).

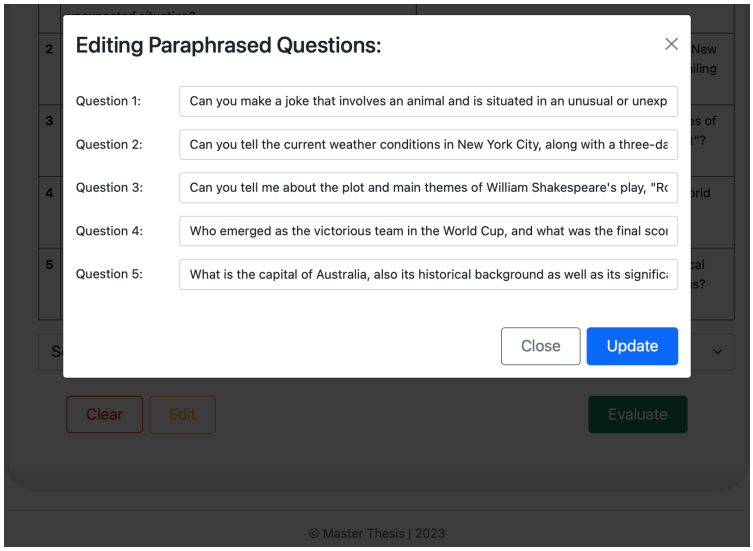

Figure 5.3. Manual Editing of Paraphrased Questions

Once satisfied with the paraphrased questions, the users could then proceed to evaluate them by clicking on the *Evaluate* button. This action would lead to the evaluate page, where the accuracy and response time of the chatbot responses to the original and paraphrased questions were displayed as a chart (see Figure 5.4).

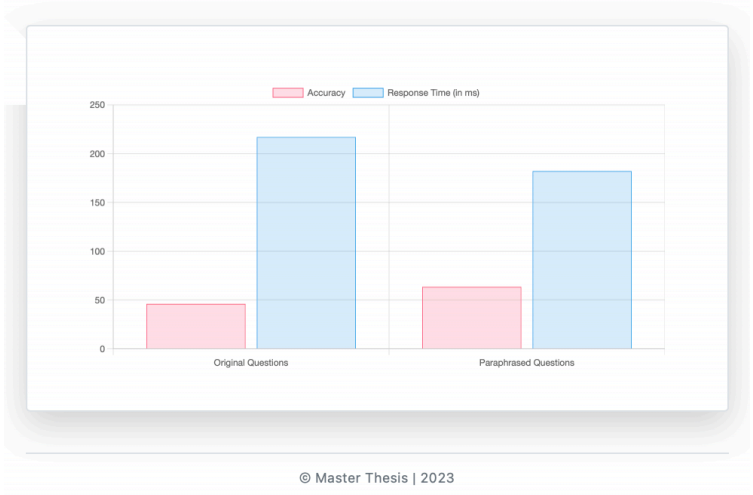

Figure 5.4. Evaluate Page of the System

In this experiment, the following five original questions, and their corresponding paraphrases generated by the system, were used (see Table 5.1):

Original Questions	**Paraphrased Questions**
Can you construct a joke that is humorous, involves some form of animal, and is situated in an unusual or unexpected situation?	Can you make a joke that involves an animal and is situated in an unusual or unexpected situation?
Could you provide me with the current weather conditions in New York City, along with a three-day forecast detailing expected changes?	Can you tell the current weather conditions in New York City, along with a three-day forecast detailing expected changes?
Can you elaborate on the plot and prominent themes present in William Shakespeare's tragic play, "Romeo and Juliet"?	Can you tell me about the plot and main themes of William Shakespeare's play, "Romeo and Juliet"?
Who emerged as the victorious team in the 2018 FIFA World Cup, and what was the concluding score in the championship match?	Who emerged as the victorious team in the World Cup, and what was the final score?
What is the capital of Australia, and could you offer a brief overview of its historical background as well as its significant attractions?	What is the capital of Australia, also its historical background as well as its significant attractions?

Table 5.1. Original Questions and Their Corresponding System-Generated Paraphrases

5.2 Metrics and Evaluation Criteria

The performance of the system was evaluated using the following two key metrics:

- **Accuracy:** Accuracy is a measure of how closely the responses from the chatbot align with the expected responses. This is crucial as the goal of a chatbot is to understand and respond to queries accurately. To assess this, we used a popular language model, spaCy [15], which has an in-built functionality to measure the semantic similarity between two pieces of text. Each original and paraphrased question was fed to the chatbot, and its responses were compared with the questions itself, assuming the perfect response should be semantically similar to the question. The average of these comparisons gave us the chatbot's overall response accuracy.

- **Response Time:** Another critical factor in evaluating the performance of a chatbot is the speed at which it responds. A successful chatbot should be able to provide responses promptly to ensure efficient interactions. We measured response time in milliseconds, noting the time before and after the chatbots response, and the difference between these times was considered as the response time. We recorded this for each original and paraphrased question and calculated the average response time to gain a measure of the chatbot's speed.

5.3 Results

Starting with the accuracy of the chatbot's responses, the original questions had an average accuracy score of 46.159% (see Figure 5.5). This score signifies the semantic similarity between the question asked and the response given by the chatbot.

Figure 5.5. Chatbot's Average Accuracy Score for Original Questions

The paraphrased questions, however, performed better with an average accuracy score of 63.55% (see Figure 5.6). This indicates that the chatbot was more accurate when dealing with paraphrased questions.

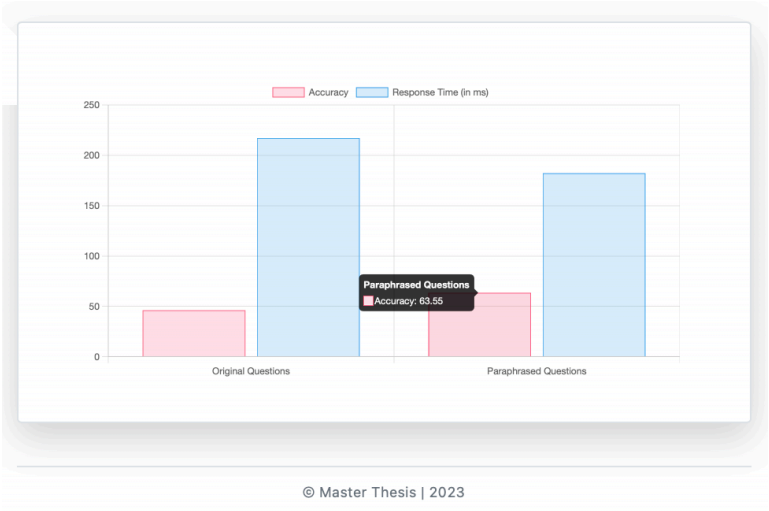

Figure 5.6. Chatbot's Average Accuracy Score for Paraphrased Questions

When analyzing response times, the original questions had an average response time of 217.212ms (see Figure 5.7).

Figure 5.7. Chatbot's Average Response Time for Original Questions

In contrast, the paraphrased questions had a slightly lower average response time of 182.249ms (see Figure 5.8). This suggests that the chatbot was quicker to respond to paraphrased questions than to original ones.

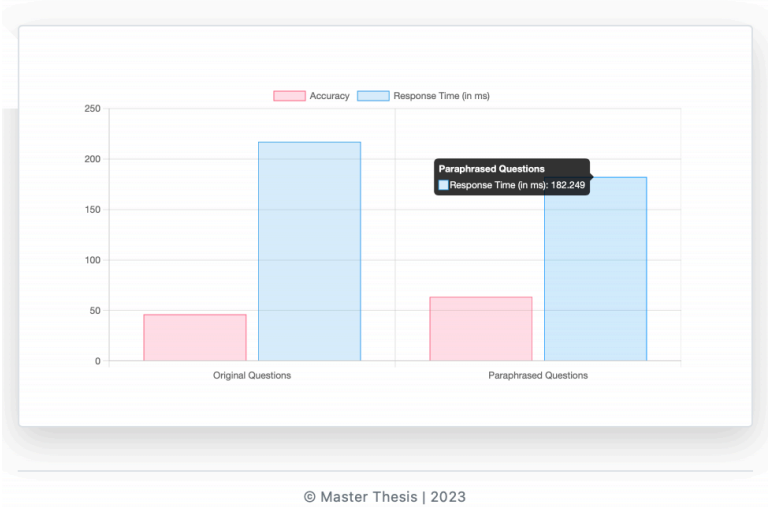

Figure 5.8. Chatbot's Average Response Time for Paraphrased Questions

It's important to note that these results may vary depending on the complexity and nature of the questions fed into the chatbot, as well as the quality of the paraphrasing.

Chapter 6

Conclusion and Future Work

In this closing chapter, we summarize our research findings and outline the potential improvements for future development in the realm of chatbot evaluation systems.

6.1 Conclusion

In conclusion, this study has sought to implement a system capable of evaluating chatbot performance through automated and manual paraphrasing of questions, leveraging semantic similarity and response time as evaluation metrics. Through an extensive exploration of chatbot integration, and the development of an intuitive and interactive user interface, this project presents a novel approach to chatbot evaluation. The investigation into existing testing frameworks and tools, along with a study of formal verification techniques, formed the groundwork for this research, laying the path for a specialized methodology to examine chatbot efficiency and accuracy. These investigations underlined the need for versatile testing methodologies that are adaptable to the constantly evolving landscape of AI and machine learning. This study provides a foundation upon which further research can be conducted, serving as a stepping stone to creating a more intelligent and efficient testing system for chatbots. By continuing to refine and expand the capabilities of this system, we can better understand chatbot behavior, ultimately contributing to the wider field of AI and machine learning. This endeavor embodies the spirit of continual learning and improvement, which is central to the advancement of any technological domain.

6.2 Future Work

In future, the potential for development and enhancement is plentiful. More advanced paraphrasing techniques could be integrated, such as the ChatGPT which has shown potential not only in paraphrasing but also in generating questions. As language model technology is advancing rapidly, the integration of newer and more sophisticated models is a promising direction. The evaluation system is effective, but the introduction of additional metrics could enhance it further. Metrics such as grammatical correctness, contextual appropriateness, and user satisfaction could provide a more rounded understanding of chatbot performance. Moreover, refining the way we predict accuracy could also lead to more precise results. The implementation of a

feedback feature that allows users to rate and comment on the chatbot responses would be a valuable tool. This could provide real-world user feedback that could be used to continuously improve the system. Overall, the future work for this project aims to expand its functionality, improve its performance, and fine-tune its user interface, thereby creating a more versatile and reliable system for evaluating chatbot performance.

Glossary

AI Artificial Intelligence

PEGASUS Pre-training with Extracted Gap-sentences for Abstractive Summarization

NLU Natural Language Understanding

NLP Natural Language Processing

CTS Conversation Testing Script

API Application Programming Interface

PyPI Python Package Index

Bibliography

[1] Ahmad Abdellatif, Khaled Mahmoud Badran, Diego Elias Costa, and Emad Shihab. A comparison of natural language understanding platforms for chatbots in software engineering. *IEEE Transactions on Software Engineering*, 48:3087–3102, 2020.

[2] Michael Beetz, Ferenc Bálint-Benczédi, Nico Blodow, Daniel Nyga, Thiemo Wiedemeyer, and Zoltán-Csaba Marton. Robosherlock: Unstructured information processing for robot perception. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1549–1556. IEEE, 2015.

[3] Cyara. What is botium? URL: `https://cyara.com/products/botium/`.

[4] Hugging Face. Hugging face transformers library. URL: `https://huggingface.co/docs/transformers/index`.

[5] Flask. URL: `https://flask.palletsprojects.com/en/2.3.x/`.

[6] Asbjørn Følstad and Marita Skjuve. Chatbots for customer service: user experience and motivation. *Proceedings of the 1st International Conference on Conversational User Interfaces*, 2019.

[7] David Griol, Javier Carbo, and José M. Molina. A statistical simulation technique to develop and evaluate conversational agents. *AI Communications*, 26:355–371, 2013.

[8] Ahmad Faizal B H. ChatBotAI. URL: `https://github.com/ahmadfaizalbh/Chatbot`.

[9] Carolin Ischen, Theo B. Araujo, Hilde A. M. Voorveld, Guda van Noort, and Edith G. Smit. Privacy concerns in chatbot interactions. In *Chatbot Research and Design Workshop*, 2019.

[10] JavaScript. URL: `https://www.javascript.com`.

[11] PyPI. Python package index. URL: `https://pypi.org`.

[12] Python. URL: `https://www.python.org`.

[13] Tim Russell. Introducing testcraft for codeless, automated testing. URL: `https://www.perfecto.io/blog/introducing-testcraft#authortimrussell`, June 2020.

[14] Siddhant Singh and Hardeo K. Thakur. Survey of various ai chatbots based on technology used. *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pages 1074–1079, 2020.

[15] spaCy. Available trained pipelines for english. URL: `https://spacy.io/models/en`.

[16] Spin. Spin model checker. URL: https://spinroot.com/spin/whatispin.html.

[17] SQLite. URL: https://www.sqlite.org/index.html.

[18] Test.ai. URL: https://test.ai/.

[19] Testim. URL: https://www.testim.io/.

[20] Arpit Rajauria (Tuner007). Pegasus paraphrase. URL: https://huggingface.co/tuner007/pegasus_paraphrase.

[21] UPPAAL. URL: https://uppaal.org.

[22] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. URL: https://arxiv.org/pdf/1912.08777.pdf, 2019.