University of Applied Sciences Ulm

Department of Computer Science

Computer Science program

Bachelor Thesis

Chatbot for Patient Guidance During Preparation of Colonoscopy

Author: Juan Arturo Abaurrea Calafell

Matriculation Number: 6007481

First reviewer: Prof. Dr. Alfred Franz

Second reviewer: Prof. Dr. Joachim Hering

September 30, 2024

Declaration of Originality

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Palma de Mallorca, September 30, 2024 _____

Abstract

This bachelor's thesis presents the development of a chatbot designed to guide patients through the preparation process for colonoscopy. The aim is to improve patient adequate bowel preparation, which is a prevalent issue, in order to obtain a higher success rate for colonoscopies. Failure to achieve adequate preparation can result in missed diagnoses and induce anxiety in patients due to the necessity of repeating the procedure.

The study aims to create a prototype of a chatbot that utilises a rule-based artificial intelligence for natural language processing and is available in multiple languages. By also incorporating the ability to remind users and send images, this chatbot is designed to provide accessible and comprehensive guidance to patients. Its development was done using the Microsoft Bot Framework and C#, with a focus on security, user engagement, and facilitating healthcare professionals to modify the bot's content with ease.

The research approach involved implementing and comparing various algorithms for the processing of user input, including Levenshtein distance, Jaccard index, TF-IDF, BM25, and sentence embedding. The chatbot's effectiveness was evaluated through algorithm performance metrics and limited patient feedback.

Key findings indicate that the BM25 algorithm provided the best balance between accuracy and efficiency in matching user queries to predefined answers. The chatbot successfully incorporates features such as automated reminders, image integration, and support for ten languages. However, challenges were encountered in integrating with WhatsApp and ensuring translation accuracy.

This study contributes to the field of medical chatbots by demonstrating the potential of AI-driven, rule-based systems in improving patient preparation for medical procedures. The developed chatbot offers a foundation for future enhancements, such as the integration of large language models or advanced semantic processing. By providing personalised, accessible guidance, this technology has the potential to improve colonoscopy outcomes, reduce patient anxiety, and serve as a model for similar applications in healthcare communication.

Acknowledgements

I would like to express my gratitude to those who assisted me during the course of my Bachelor's thesis:

To my first reviewer, Prof. Dr. Alfred Franz from the University of Applied Sciences Ulm. Your feedback on the writing of the thesis was extremely helpful and your recommendation on me extending the deadline was invaluable.

To my second reviewer, Prof. Dr. Joachim Hering from the University of Applied Sciences Ulm. As with Prof. Dr. Alfred Franz, I always felt that I could ask for your help. Moreover, it was greatly reassuring to have your support during my initial period in Ulm.

To Prof. Dr. Benjamin Walter, from the University Hospital Ulm. Your feedback, along with that of your patients, on the chatbot was of great value.

Contents

1. Introduction

Colorectal cancer (CRC) represents a significant global health concern, ranking as the third most common cancer and affecting approximately 10% of all cancer patients [1]. The prevalence of CRC has been steadily increasing, with worldwide cases rising from 1.8 million in 2020 to over 1.9 million in 2022, including more than 60,000 cases in Germany alone [1, 2]. Projections suggest a further increase to 3.1 million global cases by 2040, with Germany potentially facing 70,000 cases [2].

Colonoscopy remains the gold standard for CRC detection [3], while also offering additional diagnostic benefits beyond cancer screening [4]. However, the effectiveness of this procedure heavily relies on adequate patient preparation [5-9]. Inadequate preparation increases the probability of missed diagnoses, such as failing to identify polyps and adenomas [3, 5]. It also can lead to procedure repetition [3, 9], which causes additional stress on the patient [5].

A colonoscopy is a diagnostic procedure that involves the examination of the large intestine (colon) and rectum using a flexible tube equipped with a camera [10]. Preparation typically includes dietary restrictions, laxative use, and adherence to specific guidelines, all of which are crucial for the procedure's success [6]. Specifically, the created chatbot provides patients with a preparation plan, beginning three days prior to their colonoscopy, provided by the University Hospital of Ulm. For the initial two days, patients are advised to abstain from consuming foods containing seeds (such as kiwis and strawberries) and high-fibre items (like whole grains). On the day preceding the procedure, patients must avoid high-fat foods (e.g. fried foods) and coloured beverages (such as milk, cola, and coffee). Alcohol consumption should also be avoided during this time. That day, they are instructed to take a laxative at approximately 5 p.m. On the day of the colonoscopy, patients are required to fast and administer another dose of laxative three hours before their scheduled appointment. After the procedure, patients may resume their normal eating and drinking habits, though a gradual reintroduction to their previous diet is recommended.

In recent years, there has been a growing integration of digital technologies in healthcare [11]. As the number of individuals with access to smartphones continues to grow [6, 11], and with a significant proportion of this integration focusing on the utilisation of smartphones [11], patients can now access healthcare anywhere at any time [12]. Chatbots have emerged as one of the successful technologies that make this possible [13, 14].

The concept of chatbots has its origins with the proposal of the Turing Test by Alan Turing, which sought to assess the ability of machines to exhibit human-like intelligence [15]. The first healthcare chatbot was ELIZA, which simulated a psychotherapist by rephrasing patients' statements as questions and offering pre-planned responses [16]. Nowadays, chatbots can help monitor symptoms, provide information about procedures and answer health-related questions

in real time [5, 13, 17]. They can also reduce the burden of preparing for a colonoscopy by informing the patient [6].

There have already been numerous chatbot implementations in the medical field that have been positively received by patients [13, 14, 18]. Certain key features have been identified as essential for an optimal health software program, including natural language processing (NLP), support for multiple languages, image integration and use of proactive messages [5-7, 13]. Building on this, the thesis will focus on developing a chatbot to assist patients in preparing for a colonoscopy while evaluating various algorithms to assess their effectiveness in processing patient messages. The chatbot will be integrated into a messaging application for the University Hospital of Ulm, specifically Telegram, with the implementation allowing for an easy integration into WhatsApp. It will incorporate the previously mentioned features to ensure it meets all the necessary criteria for effectively assisting patients. In addition, critical considerations such as the security of sensitive patient data will be addressed.

2. Related Work

2.1 The Cause of Inadequate Bowel Preparation

Colonoscopies are often associated with considerable anxiety in patients, predominantly due to its invasive nature and concerns pertaining to the preparation process [5]. This anxiety can have an adverse effect on the patient's preparation, which is frequently perceived as burdensome, potentially impeding a successful outcome [5, 6]. In certain instances, it may even result in patients refusing to undergo the colonoscopy, thereby potentially missing a crucial diagnosis that has the potential to benefit them [5]. This perspective often stems from concerns about bowel preparation, embarrassment, potential pain, procedural complications, and the possibility of a serious diagnosis [5].

Other factors that negatively affect bowel preparation are misunderstanding dietary recommendations and cleansing instructions, as well as noncompliance [5, 7].

Hence, it is not surprising that inadequate bowel preparation is reported from 15% up to 25% of all patients undergoing colonoscopy [6, 7, 9]. These factors substantially decrease colonoscopy effectiveness and can cause a follow-up examination within one year [9].

2.2 Enhancing Colonoscopy Success with Digital Education

Despite the previously outlined challenges, a number of effective solutions have already been implemented to address these issues. For instance, providing easily accessible and comprehensible information to patients prior to a colonoscopy can lead to a notable reduction in anxiety [5]. Digital tools, such as smartphone applications, Short Message Service (SMS) reminders, and chatbots, have been demonstrated to enhance patient education and comfort, which causes an improved quality of bowel preparation, a crucial factor for the success of the procedure [5-7]. Chatbots have the potential to enhance the quality and experience of care by facilitating efficient, equitable, and personalised medical services [17]. Studies indicate that patients who receive digital guidance perceive the preparation process as less burdensome and are more likely to comply with instructions, leading to a better quality of bowel preparation [6, 7]. Furthermore, positive evaluations from patients support the use of these digital interventions [6, 7, 13, 14].

Chatbots are especially accessible for older cancer patients [13], which is ideal given that many patients preparing for a colonoscopy are over 60 years old [7, 19].[Title] Therefore, the adoption of mobile apps will likely become more viable as technology use among older individuals continues to grow.

2.3 Defining the Chatbot to Enhance Colonoscopy Success

The previously discussed solutions have been identifying key characteristics for effectively addressing both patient concerns about the procedure and preparation, as well as improving the quality of bowel preparation. For a chatbot to possess these characteristics, it must be user-friendly, meaning that it must be easy for patients to interact with, as well as accessible and engaging [5, 20]. From this point forward, the term "user" will refer to a patient interacting with the chatbot. Incorporating Natural Language Processing (NLP) is also crucial, as it can improve the accuracy of responses and facilitate more meaningful interactions with patients [13]. In the context of dietary recommendations, the use of images is recommended [5]. Additionally, adopting a multilingual approach could expand participation in colon cancer screening [6, 7]. Given the critical importance of data security, as patients may withhold information if they feel their privacy is at risk [14], strong data protection measures should be prioritised [13]. Finally, proactive messages are also recommended to further increase the quality of bowel preparation [13].

When developing a chatbot, it's crucial to rigorously assess its accuracy and reliability to prevent harmful errors. This can be achieved by providing the chatbot with a comprehensive dataset of medical knowledge [16]. A key consideration is the potential for the chatbot to generate false or misleading information, known as hallucinations [21]. Large Language models are particularly prone to this issue, with up to 46% of their responses potentially containing factual errors [22].

The integration of the previously described features would facilitate a positive user experience, ensuring that patients are effectively informed, thereby enhancing bowel preparation quality [13], which in turn would lead to more successful colonoscopies on average [6].

2.4 Additional benefits

The decline in birth rates and the growth of the elderly population have placed significant pressure on healthcare systems [23], which are also experiencing a shortage of health professionals [24]. In light of the current situation, there has been a growing interest in exploring the potential of automating tasks that have traditionally been performed by health workers [14]. For example, endoscopy clinicians may encounter difficulties in obtaining comprehensive family histories, a task that could be effectively managed by chatbots [20].

The utilisation of chatbots has the potential to alleviate the workload of healthcare professionals, thereby enabling them to see a greater number of patients and focus on more complex tasks [20], as evidenced by the ability to evaluate three patients in the time it previously took to evaluate one [20]. Nevertheless, while chatbots can extend patient care and

reduce workload [20], they cannot replace human healthcare providers and should be viewed as complementary tools [16, 17].

Additionally, chatbots have the potential to reduce healthcare costs and resource usage [17]. For example, improved bowel preparation using chatbots can decrease the need for repeat colonoscopies, which is linked to reduced adenoma detection rates and increased costs [9].

3. Objectives:

3.1 Implementation of a Chatbot Prototype for Real Time Guidance During Colonoscopy Preparation

The thesis aims to create a prototype of a chatbot able to guide patients during colonoscopy preparation with the Microsoft Bot Framework. This framework offers official support for multiple messaging platforms and has community-contributed support for those that are not officially supported [25, 26]. It also provides layers of abstraction that facilitate the development process, both in terms of pre-existing coded logic and behaviour [27], and by reducing platform-specific complexities [25], allowing developers to focus on the core logic of the chatbot.

The C# programming language will be used in most of the code due to its strong integration within Microsoft's .NET ecosystem [28, 29]. This choice facilitates the development, debugging, and deployment, particularly when using Microsoft's Integrated Development Environment (IDE), Visual Studio, which further enhances this integration [30].

The chatbot will be accessible through one or more messaging applications. In particular, the thesis aims to integrate the bot with WhatsApp, as requested by the University Hospital Ulm.

3.2 Development of a Concept for a Rule-Based Artificial Intelligence

Several algorithms will be studied and compared to select the one that provides the best performance, which will subsequently serve as the fundamental component in the rule-based Artificial Intelligence (AI) of the final chatbot. The rule-based paradigm ensures reliable and accurate responses. Furthermore, this method avoids any unpredictable or false responses, in contrast to generative models [22], thus ensuring that all interactions are pre-approved by medical professionals to maintain trust and reliability.

3.3 Maximising User Interaction and Accessibility

To guarantee the effectiveness of the chatbot, it is essential to prioritise user engagement and accessibility [5, 20]. It should also feature Natural Language Processing (NLP) to enhance conversational interactions [13], automated reminders to assist patients to remain on track with their preparation [5], and images to supplement instructions [5]. Furthermore, the chatbot will support multiple languages, thus ensuring that language barriers do not impede the preparation process [6, 7].

3.4 Ensuring High Data Protection Standards

In light of the sensitivity of medical data, the chatbot will be equipped with robust security measures to ensure compliance with the rigorous data protection standards of the medical sector. This involves the use of secure encryption methodologies for data transmission and storage.

3.5 Testing

The chatbot will undergo a small testing phase where it will receive feedback for necessary adjustments. This testing and evaluation will be conducted in collaboration with healthcare professionals at University Hospital Ulm.

3.6 Modifiable Bot Content and Behaviour

The chatbot's underlying data will be easily modifiable, allowing healthcare professionals to make changes without needing programming skills. This approach enables them to add or update questions the bot could answer and their answers, ensuring that the bot's responses remain current with the latest medical guidelines and patient feedback. Additionally, reminders, images, and supported languages can be modified, added, or removed as needed.

4. Technical background

A chatbot is a conversational bot, typically a web application, which in itself is a computer program. Throughout this document, the terms ‘bot’, ‘application’, ‘program’ and ‘system’ will be used interchangeably to refer to the developed chatbot when mentioned in isolation.

As already stated, the original idea of a bot was concised as a program that tries to appear as a human via conversation [15], and they are usually connected to the internet, allowing users to interact with them virtually anywhere [13, 14, 31]. A user is an individual who interacts with a system or application, in this case, a patient engaging with the chatbot. All interactions between users and the bot take place within chats. A chat refers to a screen in a messaging application where messages from two or more parties are displayed, serving as the medium for communication. For this thesis, Microsoft bot framework was chosen to develop the bot, which has a specific view of how a chatbot is.

Microsoft Bot Framework, together with Azure AI Bot Service are a collection of libraries, tools, and services that let the developer build, test, deploy, and manage intelligent bots [32]. The Bot Framework includes a modular and extensible Software Development Kit (SDK) for building bots and connecting to AI services [32]. With this framework, developers can create bots that use speech, understand natural language, answer questions, and more [32].

Bots are automated systems designed to interact with users in a conversational manner, either through text, speech, images, or video [32]. Bots function as web applications with a conversational interface, which can be any messaging app, a social network, or a custom application [32]. They process user inputs, perform tasks, and provide responses [32]. For the purposes of this document, "input" will refer to any message sent to the bot by the user.

Connecting a bot to a platform for user interaction is achieved through channels, which serve as links between a communication application and the bot [25]. Some channels are officially supported by Azure AI Bot Service, while others are accessed through community-developed adapters [25, 26]. Examples of channels include Telegram, WhatsApp, and Facebook Messenger [25, 26]. Certain channels have specific features that must be individually implemented to leverage their full capabilities [25].

Before connecting the bot to a channel, it is common practice to test it in the Bot Framework Emulator [33]. This desktop application allows developers to test and debug their bots locally before deployment, simulating interactions that the bot would have with users across different channels [33]. The emulator provides real-time feedback and diagnostics, helping developers quickly iterate and refine the bot's handling of various conversation scenarios [33], proving invaluable in the development of this thesis' chatbot.

A basic bot in Bot Framework SDK defines a bot class, referring to a class in Object Oriented Programming (OOP), that handles the conversational reasoning for the bot application [27]. It recognizes and interprets the user's input, and performs relevant tasks [27].

In addition to the bot class, there is an adapter class that facilitates connectivity with various channels, serving as a crucial link between the bot and any user interface [27]. It provides functionalities for handling incoming requests from users and generating requests to the user's channel [27]. Adapters translate messages between the bot and the specific protocols used by each platform, ensuring consistent user experiences across all channels [27].

Serving as the endpoint for incoming HTTP requests from users, the controller acts as the intermediary for processing interactions, typically delegating them to the adapter [27].

Every interaction between the user (or a channel) and the bot is represented as an activity [27]. Activities can represent human text or speech, app-to-app notifications, reactions to other messages, and so on [27]. Some examples are, Message Activity, which usually contains text, and Conversation Update, which can be generated when a user enters a conversation [27].

In a conversation, the bot generally reacts to user input, like a conversation taking turns [27]. A turn consists of the user's incoming activity to the bot and any activity the bot sends back to the user as an immediate response [27].

One limitation of the SDK is its lack of built-in storage solutions for state management [27]. However, it does offer a caching mechanism with state property accessors [34]. The developed bot's architecture defines user data as a property for these accessors, enabling them to initially verify whether the user data is cached and retrieve it if available. This approach ensures low access time during user-bot interactions.

To gather data from users, prompt dialogs are typically employed [35]. These dialogs solicit user input, with the bot repeatedly asking for the user’s data until it provides a response that the bot can understand [35]. They are designed to work with waterfall dialogs [35], which define a sequence of dialogs that act as steps, allowing the bot to guide users through a linear process to collect multiple data [35].

All of the bot’s code was written in C#, a strong typed object-oriented programming language developed by Microsoft as part of its .NET initiative [28]. Some of its features used in the program are async programming (important for handling multiple conversations concurrently) and LINQ (for data manipulation). C# was chosen primarily to ensure the project of this thesis remains within the .NET ecosystem, aligning seamlessly with the Microsoft Bot Framework [29].

Visual Studio was selected as the integrated development environment (IDE) due to its seamless integration with .NET and C# [30]. It offers features such as IntelliSense (code

completion), advanced debugging tools, and project templates specifically tailored for .NET applications [30]. Visual Studio also supports the entire software development life cycle within a single IDE, simplifying the development process [30]. Its integration with Azure facilitates deployment [30], and it easily accommodates Continuous Integration and Continuous Delivery (CI/CD) workflows with Git, as well as managing dependencies and libraries through NuGet [36, 37].

Azure is Microsoft’s cloud computing platform, providing a wide range of services for building, deploying, and managing applications in the cloud [38]. It offers several benefits, including security [39, 40] and seamless integration with other Microsoft products [41]. The Azure AI Bot Service, a part of Azure, enables bots to connect to built-in connectors for multiple communication channels [27]. To make use of these channels, bots must be deployed on Azure, as was done for the bot developed in this thesis.

For chatbot translation, DeepL was employed. It is a translation software with a free Application Programming Interface (API) plan that was utilised to translate all bot texts [42]. Multiple studies have shown DeepL's effectiveness [43-46], often outperforming its competitors [43, 44]. The translation scripts were written in Python, a high-level, interpreted programming language known for the simplicity of its syntax [47].

5. Implementation

The chatbot was developed in C# using the Microsoft Bot Framework as the core technology. The project started with the EchoBot template, which includes the typical directories and files in a Bot Framework project for elements such as bots, controllers and adapters [27]. During development, additional directories were incorporated. These include the Resources directory, which contains images, the bot's messages, and other static files; DataManagement, responsible for processing the files of the program; and NaturalLanguageProcessing, which contains the algorithms used to process and interpret user messages.

5.1 Application Initialisation

When the application starts, the Program and Startup classes are executed. The Startup class then calls the initializers for most of the bot's classes. Some operations done during initialization include: reading the static files in all the supported languages to ensure that the files are correctly formatted, loading these files into memory as data structures, and setting the reminders for all the users stored in the database.

Once initialisation is complete, the bot enters a waiting state, wherein it awaits the arrival of HTTP requests. These requests can be either from user messages or the reminder scheduler. For the reminder scheduler to send a HTTP request, it first has to determine whether a reminder should be sent. If so, it sends the request to a specific bot controller called the NotifyController.

It is important to make a distinction between the wait state and being idle. When an Azure web application is idle, it does not perform any internal processing and effectively enters a sleep state until a HTTP request is received [48]. Therefore, the reminder scheduler cannot send requests to the bot while the application is idle. The implemented solution comprises the use of a cron job that "pings" the bot at regular intervals. The term "pinging" refers to sending a small request to a server [49], which in this case is to prevent it from entering an idle state. Cron jobs are capable of executing scripts automatically at fixed times, dates, or intervals [50]. This specific cron job was created using cron-job.org, which provides a free service for this purpose.

5.2 Reminder Scheduler

The Reminder Scheduler is the system that manages the creation and deletion of user reminders. It creates them based on guidelines loaded from a file that specifies when each user should be reminded and with what message. It is built upon the Quartz.NET library, an open source job scheduling system [51]. Its functionality revolves around the creation of jobs,

characterised as the processes that send the HTTP requests to the notify controller, and triggers, which initiate these jobs at designated dates and times.

The scheduler calculates reminder dates for users both after they initially specify their appointments and when they modify them. In addition, each time the application starts, it recalculates reminders for all users stored in the database, as only appointment dates are stored, not reminder dates. To prevent unintentional behaviour, the system first deletes any existing reminders for a user before creating new ones, which is intended for when a user changes their appointment details.

The reminders are stored in a tab-separated values (TSV) file. This format is used across all sheets within the project due to its simplicity and the utilisation of the tab character as a separator. Moreover, this format allows for a clear visualisation of the file contents when opened in a text editor. In the event that a tab character is required within a column, it can be simulated by substituting it with a specific number of whitespace characters, such as four. This flexibility is not available with comma-separated values (CSV), where commas cannot be easily substituted in text, thereby preventing the use of commas within cells. All sheet files support comments, which are unprocessed lines of text that are typically employed to provide clarification regarding the file's content. In this project, comments are identified by the presence of two slashes (//) at the beginning of the line.

Each row in the reminders file corresponds to a reminder for a user, while each column denotes specific attributes such as the day, hour, and minute of the reminder. This design choice enables health professionals to modify reminders with ease and will be consistently applied throughout the rest of the implementations.

When editing this file, it is crucial to ensure that no two reminders are scheduled for the same time, as jobs and triggers are identified uniquely by the user identification (ID) and the reminder date. If a user had two reminders for the same time, they would share the same ID, which is not allowed.

5.3 User Profile

The appointment date is organised alongside additional relevant user data within an instance of the UserProfile class. It is the instance of this class which represents a user in the program. The class acts as an intermediary, facilitating access to and storage of user data, which can be present in both database and cache systems.

The retrieval of user data is conducted in a methodical manner. Initially, the system attempts to retrieve the data from the cache. In the event of an unsuccessful outcome, it queries the database. If no existing user record is found, a new user profile is created.

A key feature of the UserProfile class is its scheduled data cleanup process. This routine, executed monthly, identifies and removes user profiles with appointment dates exceeding one year in the past. This mechanism effectively manages database growth, preventing the accumulation of inactive users, therefore optimising system resources.

Each user profile is uniquely identified by a channel-provided identifier. The profile encapsulates various user-specific attributes, including language preferences for communication and the appointment date. The latter is stored in two time zones: the user's local time zone and Coordinated Universal Time (UTC). This approach allows for communicating appointment times to users in their local time zone while facilitating standardised UTC-based operations, thus reconciling potential time zone differences between users and the server.

Each instance also stores the number of consecutive unsuccessfully answered messages, defined as misses. With this, user experience is then enhanced by suggesting example questions with guaranteed responses once the number of consecutive misses reaches a predetermined threshold. This approach aims to provide users with questions they were previously unaware they could ask and reduce frustration from repeated unanswered ones. Furthermore, the user profile tracks the user's current dialog state and stores which is the current algorithm employed for message processing.

5.4 Storage

Not all user attributes are stored in persistent memory, as some do not provide practical value, such as the count of consecutive misses. In these instances, each time the bot retrieves data from the database, it assigns the user the default value, which in this case is zero. The system focuses on storing essential attributes, including the user ID, language preference, appointment date (in both local and UTC time zones), and a flag indicating whether the user is in the process of modifying an appointment. In this context, a flag is a boolean variable, which can either be true or false, serving as an indicator of a specific condition or state. The latter is particularly crucial when a user leaves the modification process incomplete; upon their return, they are required to complete the process, thereby preventing the retention of an appointment that may not accurately reflect the user's real one.

Persistent memory is necessary for two primary reasons: to preserve data in the event of server closure and to maintain user information when the cache is cleared. This approach ensures continuity in user interactions and data integrity across sessions.

Additionally, when testing mode is activated, the system stores the current algorithm used by the user along with a flag indicating its debugging status in a separate Structured Query Language (SQL) table, distinct from the user table. This separation facilitates the easier removal of test data in production environments without affecting the main user table. In

debugging mode, the bot sends the actual exception message instead of a user-friendly error message when an error occurs.

These operations are managed by the SQLiteEncryptedUserDbHandler class, which implements an encrypted SQLite database utilising the sqlite-net-sqlcipher library [52]. SQLite is a lightweight, serverless, and self-contained relational database engine, selected for this project due to its simplicity and efficiency [53].

Encryption is implemented to safeguard sensitive patient information, particularly user IDs and appointment dates. Although user IDs consist solely of alphanumeric characters and may appear innocuous if they are compromised when viewed in isolation, they become sensitive when considered alongside other compromised databases. In the event of a security breach involving both the chatbot's unencrypted database and another unencrypted database containing personal information linked to these IDs, it would be trivial to associate individuals with the fact that they are scheduled for or have undergone a colonoscopy. Similarly, appointment dates are sensitive, as they can disclose a user's location at a specific time (e.g., at a hospital). This information poses a security risk if exposed, potentially compromising user privacy and safety.

5.5 Bot's Flow Flux

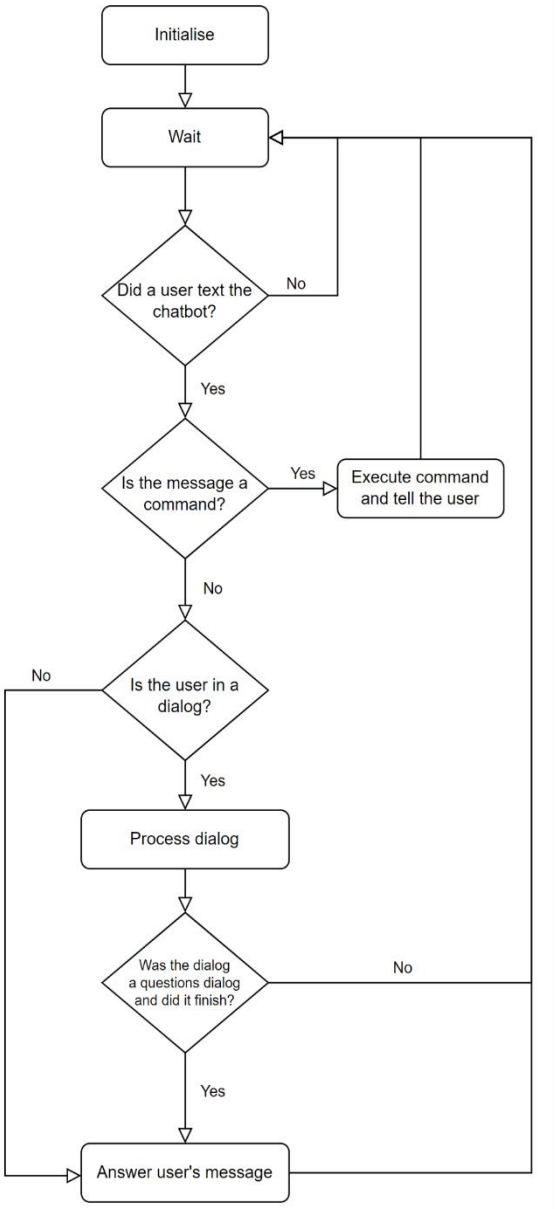

Figure 5.1: Flow diagram of the bot's cycle where squares represent actions and rhombus conditions.

Once the initialisation of the program is complete, the chatbot's life cycle begins. Initially, the bot waits for a message to be sent by a user, as seen in Figure 5.1. Upon receiving this message, it then determines whether it is a command or not. Commands are special messages that begin with a slash (/), triggering specific functionalities. The "help" command displays all available commands, with the "restart" command being of particular relevance to users. In the production version of the bot, however, most other commands should be removed. The "restart" command deletes the user from both the cache and the database, thus allowing the user to restart their initial interaction with the bot. This is particularly beneficial in instances where a user unintentionally selects an undesirable language during the user profile dialog.

The bot features three dialogs: the questions dialog, the change appointment dialog, and the user profile dialog. All of them make use of the waterfall dialog for the purpose of establishing interconnectivity between the various internal dialogs of each main dialog. The user profile dialog is the first one to be presented to the user, following the welcoming message, which is the first message sent by the bot upon the arrival of a new user in a chat where the bot is present.

The welcoming message, along with any messages that are not an answer to a specific question, such as the ones in the dialogs, are sourced from a bot messages file that contains each message's ID and its message translated in every supported language. The user profile dialog begins by asking for the user's language. If the current channel provides information about the user’s language, the question will be presented in that language. Otherwise, it defaults to German, as the chatbot is designed for the University Hospital Ulm, a German hospital. A list of languages is then presented to the user, which allows the user to select their language either by its name or by its position in the list.

Next, the chatbot asks for the month of the already scheduled appointment. Given that appointments are not scheduled more than a year in advance, the year is not required. Users should provide the month as a digit; if they fail to do so, the bot will send a follow-up message clarifying the expected format. The indication of the required response format is provided at each step of the waterfall dialog that the user fails to accomplish successfully on the first attempt. After the month, the bot asks for the day and then the time of the appointment.

An optional step follows: if the channel provides the user’s time zone, the chatbot skips this question. If not, it asks for the current hour to calculate the user’s time zone and subsequently determine the UTC appointment time based on the user's local time.

Finally, the chatbot provides a summary of the collected information and requests the user's confirmation of the data. In the event that the information is incorrect, the process will be restarted, allowing the user to provide it again. Once the user has confirmed the accuracy of the information, the chatbot provides introductory information on preparing for a colonoscopy and creates the necessary reminders based on the calculated UTC appointment and the specified reminders in the reminders sheet. Additionally, it stores the new user's information in both the cache and the database.

From this point forward, the bot enters its standard operating state, wherein it attempts to respond satisfactorily to any user questions.

5.6 Questions And Answers

The bot employs a Questions and Answers (QnA) file to find satisfactory answers by attempting to match the user’s input with predefined "questions". Each QnA file is language-specific and serves as the primary repository of the bot’s knowledge, containing various questions or statements that users can input, each associated with a specific answer. Consequently, the bot is unable to respond to inquiries that are not included in this dataset, which ensures predictable answers.

From now on, the term ‘QnA’, when used in isolation, will denote any QnA, regardless of language, as well as the specific QnA used by the user. Additionally, in the context of QnA,

the term ‘question’, will be used to refer both to its traditional definition and to a question or statement that the user's input can be matched with. It is essential to highlight this distinction, as not all entries in the file represent questions in the conventional sense. For instance, the phrase ‘I want to see a doctor’ is classified as a question within the QnA file because it reflects a potential user message to which the bot is capable of providing an answer.

All QnA files are organised into sections, with each section representing the questions and answers that the bot is capable of addressing at a given time. One exception to this rule is the first section, which contains questions that users may pose at any time, thus making it the largest section. Overall, the QnA file consists of a general section, sections for three days, two days, and one day before a colonoscopy, a section for the same day of the colonoscopy prior to its occurrence, and a final section for any inquiries made after the procedure. Consequently, if a user poses a query four days prior to the colonoscopy, the bot will attempt to identify a suitable question from the general section. In case the user repeats the same query one day prior to the procedure, the bot will conduct a search in both the general section and the one-day-prior section. In the event of a tie, the bot will give preference to the specific instance. Consequently, if the same question is present in multiple sections, the chatbot will respond with the answer pertinent to the specific section rather than the general one if possible. An illustrative example is the question "What am I allowed to eat?". While the general section might state "Eat as usual" as a response, the section for three days before the colonoscopy specifies, "Do not eat foods containing seeds or high-fibre foods", which will be selected if applicable.

The QnA file is a TSV sheet comprising three columns: actions, which trigger special behaviours for specific questions; the question itself, which is matched against user input; and the corresponding answer. Each row can contain zero or more actions, separated by a space.

Four actions are worthy of particular mention. The first action allows for the selection of an image, which, when invoked, sends the chosen image alongside the corresponding answer. The second action facilitates a dialog for changing appointments. It prompts the user to confirm whether they truly wish to alter the appointment. If the user responds "no," the user exits the dialog; conversely, if the answer is "yes," the system completes the change appointment dialog and transitions to the user profile dialog, bypassing the language and time zone steps, so the user is only asked for the date and time. This approach ensures that all user modifications occur within the user profile dialog.

The third action employs a dynamic substitution symbol, which is represented by curly braces by default. Upon invocation of this action, the symbol is replaced with a specified date, thereby converting a response such as “Your appointment is on day {} at {}” into “Your appointment is on day 12 at 10:00”, for instance. The symbol is referred to as a dynamic because its value is influenced by external factors outside the file's content and can change during execution.

Additionally, there is a static substitution symbol, represented by a single dot. This symbol is static in nature, as it is replaced with content within the file that remains unaltered throughout the execution of the program. The aforementioned symbol is used in the fourth action, which enables the copying of questions from the general section to specific sections. This feature is particularly useful when the answer varies based on the date. In order to utilise this action, it is necessary for the entry in the specific sections to also have the symbol in the questions column. In addition, the QnA file also supports copying answers using the same static substitution symbol in the answer column, where the copied answer is derived from the preceding entry that does not contain a static substitution symbol. Moreover, the symbol can be placed after a section to indicate that it should replicate the preceding section. For instance, this is done with section 2, which shares the same diet and considerations as section 3.

These substitution mechanisms serve two primary purposes: to reduce the overall size of the QnA file and to minimise the risk of inconsistencies when updating questions or answers that convey similar information.

5.7 Question Matching Algorithms

As previously outlined, the bot's approach to identifying an appropriate response to a user's input involves matching said input to questions within the QnA, while also accounting for the user's appointment date. When the bot reaches the answering action (see Figure 5.1), a score is assigned to each question in the QnA. The score represents the degree of similarity and relevance between the question and the user input, and thus how close the two are in terms of content or meaning. It is always a continuous value between zero and one, where one represents a perfect match and zero indicates no meaningful connection.

The score is calculated by question-matching algorithms, which are a collection of distinct algorithms that are used to determine the score between a question and the input. The bot's rule-based AI can then be defined as the processing of the scores calculated with a question-matching algorithm over the QnA. The effectiveness of an algorithm in interpreting user input, that is to say, its performance, is determined by the manner in which it calculates the score.

In the pursuit of the optimal algorithm, several algorithms were implemented and tested against each other with the objective of identifying which could produce the highest number of "hits", the primary metric selected for the comparison of the algorithms. A "hit" is defined as obtaining one or more successful matches to a given user input, in contrast to a "miss", which is due to all matches failing to exceed the minimum threshold. The threshold serves to remove any responses that are irrelevant or loosely related. A successful match must exceed the pre-established acceptance threshold. This implies that within the aforementioned thresholds, there will be instances where the user's input is matched with a result that is, at least partially, relevant, yet the system is unable to ascertain with certainty that this is indeed the case. In the

absence of successful matches, the bot employs a questions dialog to ask the user if its input aligns with the questions that have been matched exceeding the minimum threshold but not the acceptance one. Should no matches pass any threshold, the bot apologises for its inability to comprehend the input. In the event of repeated instances of this occurring, the bot suggests example questions that it can answer with confidence.

It is necessary for each algorithm to define its own minimum and acceptance scores. In the case of a simple algorithm, these values can have a straightforward interpretation. However, in the case of a complex algorithm, it is challenging to ascertain the meaning of the score. Consequently, the only viable approach is to conduct a trial and error process.

5.8 Levenshtein Distance

To illustrate, the algorithm that has been implemented utilising solely the Levenshtein distance has a minimum score of 0.8 and an acceptance score of 0.95. In this context, an acceptable match is defined as one with a Levenshtein distance of five or fewer units from the user's input, which produces a score that is easily interpretable.

The Levenshtein distance measures the difference between two strings of text. It is calculated by the number of insertions, deletions, or substitutions that would be required for one string to be transformed into the other [54].

As the first and most basic of the algorithms, it is not the most effective. Given that the score is inversely proportional to the distance, it follows that two questions of very different lengths will never achieve a high score. This is suboptimal, given that there are numerous ways of articulating sentences that may convey the same meaning. To illustrate, consider the input “When is my appointment?” and the question “When is my appointment scheduled?”. The Levenshtein distance is 10, which equates to a score of 0.9. This implies that the algorithm would never register it as a hit, despite the two sentences conveying the same meaning. It is crucial to emphasise that the thresholds cannot be reduced to prevent the assignment of high scores to unsatisfactory matches.

The development of this algorithm, however, yielded valuable insights into the nature of the problem. It suggested that question lengths should have a lower influence, if any. It also gave rise to the notion of employing it to mitigate user typographical errors. Should the user send a word not present in the QnA, the algorithm can transform it into the nearest possible word present there.

For all the algorithms both the questions and the user input are normalised, which can lower even more the influence of user typographical errors. Consequently, the matching is always done with normalised input and normalised questions. Preprocessing is essential, and normalisation is a key step in standardising and removing any potential noise from the dataset

[55], in this case the questions. The applied normalisation consists in the removal of diacritics in the Latin alphabets supported by the chatbot, including the umlaut. Furthermore, all punctuation and superfluous spaces are removed, and all letters are converted to lowercase. For instance, the German input “Hallo, welche Diät muss ich einhalten?” gets normalised into “hallo welche diat muss ich einhalten”. This standardisation process minimises the influence of user typographical errors.

All algorithms are designed to consider the synonyms of the words. This entails processing every combination of synonyms of the user’s input found in the synonyms set. For example, consider the user input “I desire to contact a physician”. This could be transformed into “I want to contact a physician”, “I want to contact a doctor” and “I desire to contact a doctor”. All four ways of expressing the same concept would be processed to attempt to match with a question in the QnA.

5.9 Jaccard Index

The second algorithm developed was the Jaccard index. In contrast to the Levenshtein algorithm, it initially tokenises the input, which in this case involves the separation of questions into words. This procedure enables the comparison at a word level rather than a sentence level, a feature that is shared in all subsequent algorithms.

$$\text{Jaccard}\,(A, B) = \frac{A \cap B}{A \cup B}$$

Figure 5.2: Equation needed to obtain the Jaccard index.

In order to obtain the score, the algorithm first calculates a set of words derived from the user's input [56]. In the field of mathematics, a set is defined as a collection of elements that are distinct from one another [57]. It then calculates the intersection and the union of the two sets [56]. The intersection is the set of words both sets share, whereas the union is the combination of all words that appear in either the user’s input or the question [57]. Ultimately, the number of words in the intersection is divided by the number of words in the union [56]. To illustrate, consider the following example: the input “What should I avoid?” and the question “Which foods should I avoid?”. Once the normalisation and tokensation processes have been completed, the resulting intersection is “should, i, avoid”, comprising three words, whereas the union is “what, should, i, avoid, which, foods”, comprising six words. The score, obtained by dividing 3 by 6, would then be 0.5, which in this algorithm barely passes the minimum score required, while a good score would need at least 0.7.

As demonstrated by this example, the algorithm may still be suboptimal for a robust user experience. However, it represents a significant improvement since the length of sentences and the order of words are no longer determinant. The latter is a particularly important aspect since

in many languages sentences can be described in many ways by ordering the words differently. In this instance, the order is irrelevant due to the algorithm adhering to the bag-of-words model [56]. Furthermore, as previously stated, the algorithm incorporates techniques for reducing typographical errors and processing synonyms.

5.10 Term Frequency–Inverse Document Frequency

The next developed algorithm exhibited a significant increase in complexity. The Term Frequency–Inverse Document Frequency (TF-IDF) algorithm is a method of measuring the importance of a term within a document, as part of a larger corpus or collection of documents [58, 59]. In this project, the collection is the QnA, whereas the documents are questions. The algorithm operates by assigning a higher importance to words that appear frequently in a document and are infrequent in the QnA, and a lower one to words that appear only a few times in the question and are common in the QnA. This ensures that the most relevant words score higher in importance and that popular stop words (such as “a” and “the”) exert less influence on the result.

The importance of a word is determined by multiplying its frequency (term frequency or TF) in the question by the inverse document frequency (IDF), with the latter indicating the degree of information provided by the word and its rarity across the QnA [58]. There are numerous methods for calculating both TF and IDF [58, 59]. In this implementation, TF is obtained by dividing the number of times a word appears in a given question by the total number of words in that question (see Figure 5.3). The IDF for a word is calculated by first dividing the total number of questions by the amount of questions that contain that word, and then applying logarithm to the result. To avoid dividing by zero, a smoothing technique of adding one both in the numerator and denominator is applied.

$$TF\,(t,d) = \frac{f(t,d)}{\sum_{\mathrm{t'}\in d} f(\mathrm{t'},d)}$$

Figure 5.3: Equation for obtaining the Term Frequency.

$$IDF\,(t) = log\left(\frac{1+\,|\mathrm{D}|}{1+\,|\{d : d \in D \wedge t \in d\}|}\right)$$

Figure 5.4: Equation for obtaining the Inverse Document Frequency.

Once these values have been calculated, they can be multiplied to obtain the TF-IDF value for a given word within a question, which represents its importance [58, 59]. It should be noted, however, that the resulting value only measures words and does not take into account the score of the questions as a whole. In order to compare two sentences, a solution was devised which involved the creation of a vector space in which each word correlates with a dimension. Then, each question is assigned a position within this vector space. In order to identify the optimal match, the user's input position within this vector space is calculated, and the question that is the closest in proximity is then identified. The position of a question for a given dimension is

the TF-IDF value calculated for the corresponding word; thus, it is necessary to calculate this value for each word.

Two options were available for determining the proximity of another vector: cosine similarity or Euclidean distance. The selected method was cosine similarity, as it assesses the angle between two vectors, which is particularly beneficial in this context given that the magnitude of the vectors (influenced by factors such as sentence length and total word count) is less crucial than the relative similarity of their content [58]. Consequently, the result is situated between negative one and positive one due to the cosine function. However, to ensure that all scores remain between zero and one, any values below zero are set to zero.

Given that in the English QnA there are approximately 200 unique questions and 300 unique words, and that a question typically comprises around seven words, it follows that the majority of values in this vector space will be zero. This is due to the fact that in the event that a question does not contain a word, the TF-IDF is equal to zero. This would indicate that, on average, only seven dimensions in a vector would be non-zero in the English language, with a similar effect being observed in the other languages. This results in a sparse vector space, which is memory inefficient.

This method provides a more elaborate approach to identifying the optimal question to match the user's input. Subsequently, the threshold scores lack the straightforward and comprehensible correlations observed in the preceding algorithms. Nevertheless, the score provides a more precise and reliable measure of relevance compared to previous algorithms, as the algorithm considers the significance of each word within the context of the entire dataset.

5.11 Best Matching 25

Although TF-IDF represented a notable advance over earlier algorithms, Best Matching 25 (BM25), also known as Okapi BM25, is a more sophisticated version that incorporates several key improvements. While both algorithms employ a similar approach, whereby term frequency is multiplied by the inverse document frequency, their underlying calculations diverge significantly [58-60]. The first notable distinction is the allowance for fine-tuning through adjustable parameters in BM25, which can be optimised for specific query types [60].

$$TF\left(t_i, d\right) = \frac{f(t_i, d) \cdot (k_1 + 1)}{f(t_i, d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\frac{1}{N}\sum_{i=1}^{N}|d_i|}\right)}$$

Figure 5.5: BM25’s version of TF.

$$IDF\left(t_i\right) = log\left(1 + \frac{N - n(t_i) + 0.5}{n(t_i) + 0.5}\right)$$

Figure 5.6: BM25's version of IDF.

Another noteworthy distinction lies in the incorporation of document length into the calculation of term frequency [60]. Using this approach, the presence of a greater number of non-matching terms in a document results in a reduction in the score [60]. This can be understood through the following intuitive example: if a text is thousands of words long and mentions a specific term only once, it is less likely to be as relevant as a shorter text that mentions the same term once.

A further key improvement in BM25 is how it handles term frequency saturation [60]. Unlike TF-IDF, BM25 follows an approach where the score contribution of a term decreases non-linearly as it appears more frequently within a document [60]. This guarantees that documents characterised by an excessive repetition of terms do not prevail over those with a lesser repetition of terms. To illustrate, the score generated by incorporating a queried term into a document that already contained 100 instances of that term will not increase to the same extent as adding that term to the same document but only having that term 10 occasions. This behaviour serves to prioritise meaningful content over mere repetition. However, given that the questions in the QnA are relatively short documents, this characteristic is not particularly beneficial here. The same can be said for the previous improvement, which cannot be applied here due to the questions being small documents.

A significant departure from the preceding algorithm is the manner in which BM25, in contrast to generating a sparse, memory-intensive vector space, calculates a score for each question in relation to the user's input. This results in a comparison of the resulting value with other questions in order to determine which one provides the best match. For each word in the user's input, the BM25's TF-IDF value is computed, and the results are summed to obtain the score [60].

In order to ensure consistency in the scores across different algorithms, the BM25 score is transformed into a number in the range of zero to one. This is achieved by setting all negative values to zero and utilising the formula $1 - e^{-x}$, which ensures that the score remains below one.

5.12 Sentence Embedding

The final algorithm implemented was a sentence embedding algorithm based on word embeddings. This approach involves representing entire sentences as vectors, derived from aggregating word embeddings of the individual words. Unlike the previous algorithms, sentence and word embeddings are distinguished by their capacity to capture semantic meaning [61].

The distinction between this approach and TF-IDF is that the latter employs a vector space defined by the words in the QnA and their synonyms. In this manner, each dimension represents a word. In contrast, the embedder creates a vector space of a selected number of dimensions,

with each dimension representing a specific feature or characteristic of the word embeddings. In this space, the vectors mainly correspond with individual words, but also other language elements such as punctuation marks. The dimensions represent abstract features that define relationships among these vectors. Furthermore, the vector space is dense, where it is rare to find a value of zero, unlike the vector space created with the TF-IDF implementation.

The vectors employed in this algorithm were pre-trained and sourced from the fastText library [62]. A multitude of languages are available for download [62]; however, only German and English were selected for this project, as each language requires approximately 4GB of memory to load [62]. This is due to the fact that the vector space contains 2,000,000 vectors across 300 dimensions [62]. The total amount of memory required to load one of these vector spaces exceeds the limits of the hired Azure plan. Consequently, the vectors were never deployed into the Azure version, making this algorithm unreachable for a user. However, this approach offers a new advantage not present in previous algorithms due to having a big vocabulary. It enables the processing of words not present in the QnA or synonyms files, thereby expanding the range of expressions that can be understood.

As with the TF-IDF implementation, the process of comparing sentences begins with the user's input position being calculated in the vector space. This is achieved by calculating the average vector of all its word vectors. If a word from the input is not found in the vector space, the algorithm can account for up to two typographical errors per word. Subsequently, the cosine similarity for comparison is calculated, in a manner analogous to that of TF-IDF's vector space. Finally, an additional step is applied whereby all negative values are set to zero.

In contrast with the other algorithms, no further operations are conducted, such as normalisation and the extraction of synonyms. The method does not normalise because even the punctuation marks are considered in the vector space; therefore, eliminating them would result in the loss of information. The synonyms, given that they share a similar meaning, are proximate in the vector space and do not necessitate additional processing to consider them.

This algorithm effectively addresses various challenges, particularly by taking semantics into account [61]. Its direct approach, without the need for additional processing such as normalisation or synonym processing, allows it to take advantage of the valuable information from the vector space. Given its ability to capture the nuances of meaning, this algorithm is worthy of further exploration and attention as a potential improvement for the bot. By integrating this method, the bot could enhance its understanding and responsiveness, ultimately leading to a better user experience.

5.13 Synonym Processing

With the exception of the Sentence Embedding algorithm, a synonym processing procedure is applied during the processing of user input. The system developed for this purpose should assist the bot in identifying the various ways in which the same idea can be expressed. In order to achieve this, synonyms are firstly retrieved in the JavaScript Object Notation (JSON) format from the Datamuse API [63]. The API is able to return a list of synonyms along with a score [63], which is interpreted as how close the synonym is to the requested term in meaning.

In order to guarantee that only pertinent synonyms are employed within the algorithms, a score threshold is implemented, effectively eliminating unsuitable alternatives. To provide an illustration, a search for the synonym of "fiber" (in American English) yielded a set of results, including "character", "fibre" (the same word but in British English), and "vulcanized fiber" with scores of 3744, 728 and 24 respectively. The obtained synonym "vulcanized fiber" is of lesser utility than "fibre" for both general purposes and the topic of the chatbot, and thus it is filtered. It has been observed that lower-score synonyms tend to be more specific. Therefore, filtering may help to avoid the use of extremely specific terms as synonyms. Despite the filtering, however, some manual selection is still necessary to ensure the synonyms are contextually appropriate. For example, "character" is not a suitable synonym for "fiber" in the context of the chatbot.

The final output is a synonym file, in which each row contains a group of synonymous words and each column represents an individual synonym. Each word in a row is considered to be equally equivalent to the others within that set. In order to maintain consistency, each synonym should only appear once throughout the file, since all of its synonyms should already be present in the row in which it is included, eliminating the need for a second appearance in the document. However, since certain words may appear as mutual synonyms, for example, "fiber" appears in the synonyms of "fibre" and vice versa, an additional step is required in order to group all the synonyms of both words into the same row.

The next step is to load the synonyms into memory for subsequent use. This requires the use of some form of data structure. The initial concept that comes to mind is a list of lists, as it is a relatively straightforward approach. This method would represent a collection which encompasses all collections of synonyms. However, this solution would result in a cost of, in terms of Big O notation, O(n) both in time and memory for searching the synonyms of a given word [64]. This is because it would require traversing all the words until the desired one is found, and each word would be stored once.

Responsiveness is a crucial aspect of chatbots, as it contributes to maintaining dynamic conversations. However, an excessive number of synonyms could potentially lead to significant latency issues. To address this challenge, an alternative data structure was employed in this project. Despite an exhaustive search of the internet for a data structure that would

resolve the specified issue while taking into account the nature of the problem, no suitable solution could be found. Consequently, an implementation was devised, which was given the name CategoriesDictionary. This is an implementation of a dictionary or mapping that employs sets, specifically hash sets, as values. These are collections that store unique items and allow for constant-time lookups, with the latter also being true for dictionaries. In Big O notation, this is represented as O(1) [64].

In the context of synonyms, the dictionary is composed of synonyms themselves, functioning as the keys, and a hash set of one or more synonyms, serving as the value. The creation of a tree for each group of synonyms is initiated with the selection of a single arbitrary synonym, which serves as the root. These trees are consistently of depth one, with every node pointing either to its root or to its children.

This approach allows for the retrieval of a synonym with a time cost of O(1), but a space cost of O(3n - 2), which is also linear as with the previous solution [64]. The constant time required for access is a consequence of the underlying data structure, which is a dictionary, and therefore allows for the instant retrieval of its members. The space cost is due to the fact that when a node is inserted into an existing tree, this entails the addition of the node, which stores the word in question and the root to which it points. In addition, a reference for the new node must be added to the root set. Consequently, the addition of one synonym results in the storage of three more words. The subtraction of two is a consequence of the intrinsic characteristics of the data structure and should not be a significant concern, given that it does not significantly impact the memory cost. The trade-off of time and space is reasonable when one considers that the file does not contain thousands of synonyms.

Once the synonyms have been incorporated into the data structure, they are subjected to a process of normalisation in accordance with the established procedure. In the event that none of the synonyms in a row are identified within the vocabulary and therefore the questions, they are bypassed and not incorporated into the data structure, given that they could not be matched with any question. The implemented data structure also addresses instances where an invalid operation is performed, such as when a word is present twice within the file. In such cases, the data structure reverts to its previous state, and the program skips that row. Following this processing, the synonyms can then be utilised across the program.

5.14 Deployment and Channel Connectivity

The deployments into Azure were conducted using a Continuous Deployment/Continuous Integration (CD/CI) pipeline with a Git workflow, thereby streamlining the process of updating and deploying new bot versions [65, 66]. This approach allowed for the automatic integration of any updates into the live environment without the need for manual intervention.

Prior to any deployment, a number of essential services were set up and configured in Azure. These required a pay-as-you-go subscription and included the App Service, which hosts the web application responsible for starting and stopping the bot, the App Service Plan, which defines the infrastructure and resources for hosting the application, and the Azure Bot Service, which enabled the bot to connect with various channels, thus allowing it to interact with users across multiple platforms [67].

In order to conduct preliminary testing of the hosted bot, the Web Chat channel provided by Azure was utilised, which is accessible via Azure Bot Service [67]. The subsequent channel through which the bot was evaluated was Telegram, which offers a straightforward approach to integrating bots [68]. This was the setting in which the bot was tested by two patients at the University Hospital Ulm, where it received a rating of 4 out of 5 stars. Specifically, the trial was undertaken with BM25 as the active algorithm. It was observed that the bot exhibited some incorrect sentences due to minor translation inaccuracies from English to German.

Additionally, an integration with WhatsApp was implemented using a community adapter, as no official WhatsApp channel was available [69]. The options available for the C# version of the Microsoft Bot Framework for WhatsApp involved utilising the services of one of three companies: RingCentral, Infobip and MessageBird [69]. Infobip was selected due to its customer support services [70] and the free trial it offers [71]. In order to implement the bot on WhatsApp, the following main steps were followed: the creation of an Infobip account, the configuration of a Meta Business account and the purchase of a phone number.

Despite having completed all the requisite steps, the WhatsApp account linked to the Meta Business account was not approved as it did not pass the terms of service [72]. An appeal was submitted, outlining how the project complied with all of their requirements, while also offering to provide additional information if necessary. However, no response was received.

6. Discussion

The development of the chatbot brought many insights about the challenges and the possible improvements. This will be discussed, but first, the algorithms should be assessed between each other to decide the final one used.

6.1 Algorithms comparison by tests

Algorithm	Interpretable score	Required Specifications	Average Time (ms)	Hits ratio	Misses ratio	Additional comments
Levenshtein distance	yes	Low	240,590	1	0	Length sensitive and does not work with individual words
Jaccard index	yes	Low	264,432	0,85	0,035	Lacks term frequency
TF-IDF	no	Medium	1646,956	0,945	0,04	Memory-consuming vector space
BM25	no	Low	99,326	0,965	0,035	Improvement over TF-IDF
Sentence Embedding	no	High	954,570	0,78	0,22	Considers semantics; there is an extra latency for model initialisation the model

Table 6.1: Comparison of the algorithms

The following section presents a comparative analysis of the implemented algorithms, with a compilation of data summarised in Table 1 (see Table). The table serves two purposes: firstly,

it provides a summary of the algorithms, and secondly, it offers a basis for comparison. The metrics were calculated by first selecting 200 random questions from the English and German QnAs, whereupon for each question a word was substituted with its synonym and two typographical errors were applied to the question. In the case of the Sentence Embedding, however, the number of questions was reduced to 50.

The questions were selected from the QnA because, although there are several datasets pertaining to images and videos of colonoscopies [73], there is a pronounced scarcity of datasets concerning conversations of patients regarding preparation for colonoscopy.

The table provides an overview of the interpretability of the resulting score of an algorithm, the general level of required specifications, including CPU and memory requirements, and the average time required to calculate the best match for a given input. It also presents the total hits ratio, which is the ratio of successful matches, and the misses ratio, which indicates instances where no answer is found. Additionally, the table includes a comments column in which some considerations of the algorithms are outlined.

It should be noted that the metrics are not as representative as they may initially appear. This is due to the fact that the questions tested were highly specific, as they were all already present in the QnA. This does not accurately reflect the manner in which a user would interact with the bot in a real-world setting. Additionally, the ratios are determined by the a-priori defined score thresholds, which are not uniform across all algorithms. As an illustration of this phenomenon, the Levenshtein Distance, the simplest of the algorithms, exhibits the highest level of performance, demonstrating no misses and all hits. However, given that all questions were within a maximum of two typographical errors of the original word and the algorithm's acceptance score considers up to five, it will consistently identify every question. Consequently, it is essential to exercise caution when interpreting these results.

Progressing from Jaccard index to TF-IDF and finally to BM25, there is a noticeable increase in performance, as evidenced by an increase in the number of hits. This is an expected outcome, given that each algorithm addresses the limitations of the previous one. One noteworthy observation is that BM25 does not typically yield matches between the minimum and acceptance thresholds, as the sum of both ratios is equal to one. This could indicate the necessity for further refinement of its threshold values.

It is also notable that BM25, the most performant algorithm, is the optimal in terms of processing speed, in contrast to TF-IDF, which is the slowest of the algorithms. This may be attributed to the vector space, which necessitates the calculation of cosine similarities, an expensive calculation conducted on the CPU. Despite the comparatively complex algorithms employed by BM25 for both TF and IDF, its results are merely summed and then applied to a bounding function, which does not require as many operations. Consequently, BM25 is the

most efficient algorithm among the four, delivering the most optimal results with minimal latency. It is therefore the default algorithm when a user interacts with the bot.

Finally, the Sentence Embedding algorithm appears to be the most promising option, as it incorporates all the features of BM25 while also accounting for semantic meaning. However, the data indicates otherwise, suggesting the need for a closer examination of the underlying reasons for this discrepancy. This phenomenon is especially intriguing given that when it was tested in the Bot Framework Emulator it was found to be promising.

A further irregularity of this algorithm is the length of time required for execution. Although the optimal match is calculated in less than a second, the initialisation process is often lengthy, frequently spanning several minutes. To illustrate, the data for the initial four algorithms was obtained within a 15-minute timeframe, during which the bot processed 800 questions. However, when the data for the Sentence Embedding was obtained separately, the processing time for 50 questions was 10 minutes. When additional overheads and initialisation times are taken into account, the average duration for answering a question for the first four algorithms is approximately 1,125 seconds, in contrast to the Sentence Embedding algorithm, requiring 12 seconds. This, along with the inadequate preliminary conditions, such as the questions being tested, led to the decision not to continue testing this algorithm. Nevertheless, it is important to emphasise its potential as part of the bot's future.

6.2 Language Testing

Although the bot is available in a multitude of languages, its functionality has only been evaluated in English, German, and Spanish. Consequently, it is imperative to extend the testing process to encompass the remaining supported languages.

6.3 Medical Feedback

From the limited tests conducted with the bot, the sole issue that was identified was that of translation inaccuracies between languages. This should serve to emphasise the necessity of verifying all translations, despite the fact that DeepL has often been found correct during the development process. Ideally, this process should be handled by a professional translator.

6.4 More questions

The functionality of the bot is centred upon the questions and answers that have been stored in the QnA. It is unavoidable that the bot may not have a corresponding answer to each message sent by the user. It is therefore essential to include as much information as possible in the form of answers, with the questions serving as access points to that information, in order to enhance the functionality of the bot. Moreover, it is crucial to recognise that each user may express

themselves in a distinctive manner, which can present challenges in identifying a question that closely aligns with their written communication. To enhance the probability of a successful match and provide a satisfactory response, it is advisable to incorporate multiple variations of how the same question can be posed. The information required to write these questions and answers could be obtained through a larger trial of the bot.

6.5 Synonyms

In the present state of the project, synonyms are treated in the same manner as the original words. An additional enhancement would be to optimise the management of synonyms by storing the score and utilising it to influence each match's score. The solution would consist on assigning each word of the user's input a score independently of the algorithm. The initial score would be set at one, reflecting the degree of similarity to itself. In the event that a synonym for this word is identified, another version of the sentence, with the substituted word, would be created, in accordance with the current method. The words would maintain the previous score for all words except the synonym, which would be assigned a lower score. A lower score would indicate a lower degree of similarity between the synonym and the original word.

In the case of BM25 or TF-IDF, the value of a word could be multiplied by the value obtained when calculating the TF-IDF of a word, which would then exert a negative influence on synonyms that are not particularly similar to the original word. To achieve this, however, it is necessary to have one row per synonym in the files where synonyms are stored, thereby enabling the storage of the synonym score for each word. This would necessitate an increase in memory since, in contrast to the previous approach, the synonyms can now appear more than once in the file.

6.6 Typographical Errors

Currently, the bot's approach to identifying words that are not recognized is to search for the most similar one in the QnA questions and substitute it by the best approximation. Consequently, the same level of importance may be attributed to a word that accurately reflects the user's intention and another that may not. The solution is analogous to that discussed with synonyms, whereby the value of one is assigned to unmodified words and 1 - alpha*x score to words with x modifications. In this context, alpha represents the weight by which the typographical errors impact the outcome. For the time being, alpha is set to zero, but this could be adjusted based on the observed output. As a preliminary estimate, alpha could be set to 0.1, with each modification contributing 0.1 to the word's score.

6.7 Cosine Similarity

The Sentence Embedding and TF–IDF algorithms calculate the cosine similarity with the CPU each time it is used. Operations of this nature, which are repetitive, are ideally suited to being performed on a GPU [74]. This would result in a faster calculation, which would in turn result in a reduction in latency [74].

6.8 Reducing dimensions

As previously indicated, the Sentence Embedding is presently unsuitable as a solution due to its elevated specifications. It may be beneficial to reduce the number of dimensions in the vectors from 300 to, for example, 100. However, this would result in a diminished performance in semantic interpretation by the model. The FastText library facilitates this reduction through a few steps [62].

6.9 Large Language Model Integration

In recent years, there have been significant advancements in the use of Large Language Models (LLMs) for communication with users in natural language [21]. Although these models are susceptible to the occurrence of hallucinations, they are still capable of answering a wide range of questions [21], and although some responses may be incorrect, it may be beneficial to attempt a refinement of the model to ascertain whether a significant reduction in the probability of misinformation being disseminated can be achieved, thereby enabling a solution to be considered.

One potential integration within this project would be to transfer the responsibility for answering the user from the rule-based AI to an LLM in the event that the former is unable to find an appropriate response.

6.10 Security

In the medical field, it is of the utmost importance to implement robust security measures, particularly when dealing with systems such as medical bots that handle sensitive patient information. This project employed a multitude of measures to ensure the security of the bot.

For the transmission of data, the bot employs Hypertext Transfer Protocol Secure (HTTPS). This protocol guarantees the encryption of each transmission [75], thereby preventing interception or manipulation of the data by third parties with malicious intent.

In order to minimise the potential exposure of sensitive information, the system employs a data minimisation strategy. Only the user data that is essential for the functioning of the system is

retained; this includes user IDs, appointment dates and language preferences, with the latter being of a less sensitive nature than the former two. The data is stored within an encrypted database employing a 256-bit key, which has been recognised as a highly robust encryption method [76]. Currently, the database encryption key is embedded within the bot's codebase. However, this approach is not suitable for a production environment and must be revised before practical use. One potential solution would be to employ a dedicated key management service, such as Azure Key Vault [77, 78].

To maintain long-term security, it is necessary to implement a key rotation policy. The regular rotation of API and encryption keys serves to minimise the potential impact of a key compromise [79], by limiting the time window during which any compromised key could be exploited.

Compliance with the General Data Protection Regulation (GDPR) is imperative, given that the bot is intended for use within the European Union [80]. The regulation addresses several key aspects related to data processing, including the need for explicit user consent [81]. This can be achieved by informing patients about the chatbot's intention to collect appointment dates and language preferences, along with storing their ID. Furthermore, the regulation emphasises the importance of data minimisation practices [82], which have already been implemented by only collecting the necessary data, and the right to be forgotten [83], which has also been addressed through the inclusion of the "restart" command.

Finally, the selection of a messaging platform also has an impact on the security of the bot. The security features available on different platforms vary considerably. Signal is renowned for its robust security features, including end-to-end encryption by default [84]. However, its restricted adoption among the general public may limit its practical application for a widely accessible medical bot [85]. Telegram and, in particular, WhatsApp have a significantly larger user base, making them potentially more suitable for a medical bot aimed at broad accessibility [86, 87]. However, the security features of these platforms should be subjected to thorough examination. Telegram offers end-to-end encryption for "secret chats", though this is not the default setting for regular chats [88]. WhatsApp implements end-to-end encryption for all messages but as the rest, has also suffered security breaches [84]. The decision to utilise these platforms must take these considerations into account.

7. Conclusion

The goal of this thesis was to develop a prototype of a chatbot designed to assist patients in preparing for their colonoscopy. The chatbot would employ a rule-based AI together with images, reminders, NLP and multilingual support to achieve this objective. Additionally, the chatbot needed to be secure, due to its handling of sensitive data, accessible via WhatsApp, as requested by University Hospital Ulm, and its content easily modifiable by any individual capable of using a computer.

Most of the objectives have been achieved. The final version is a fully operational chatbot featuring a conclusive AI, comprising the QnA and the BM25 algorithm, which performs adequately, as evidenced by the comparative analysis of algorithms and patient feedback. The chatbot effectively communicates with users in natural language through the utilisation of NLP and incorporates images to enhance the user experience. It supports ten languages, namely Arabic, English, French, German, Italian, Polish, Russian, Spanish, Turkish, and Ukrainian. However, professional translation verification is recommended. Security measures have been employed, including the encryption of the database and the utilisation of the HTTPS transmission protocol. The bot's content, including reminders, messages and questions and answers, can be conveniently modified using a spreadsheet or even a text editor. Despite the failure to achieve WhatsApp inclusion as a means of communicating with the bot due to issues with Meta, the bot is fully operational on Telegram with all intended features intact.

The final bot has several limitations, including the fact that it is only available on Telegram, that it is not fully accurate in translation, and that it has only been tested to a limited extent. These challenges could be addressed in a future version in conjunction with further improvements, such as the use of the Sentence Embedding algorithm or the integration of LLMs, both of which have the potential to be highly beneficial. Should this not prove to be the case, the knowledge gained from this project can be shared with researchers and developers in the field of medical chatbots.

The solution developed in this thesis aims to improve patients' bowel preparation quality and reduce anxiety by providing accessible, feature-rich guidance. It highlights the potential of health chatbots to offer real-time support, enhance communication, and improve compliance in medical settings. This chatbot could not only contribute to better colonoscopy outcomes but also demonstrate the broader role of AI-driven chatbots in healthcare. Additionally, it serves as a foundation for future chatbot development, with the potential for real-world application to assist patients daily.

The solution developed in this thesis aspires to be able to enhance the quality of patients' bowel preparation and mitigate anxiety by offering accessible, feature-rich guidance and serving as a reference for future chatbot development.

8. References

1. Y. Xi, and P. Xu, “Global colorectal cancer burden in 2020 and projections to 2040”, in *Translational Oncology*, vol. 14, Oct. 2021

2. World Cancer Research Fund International, “Colorectal cancer statistics”, Feb. 2022, Accessed on 30 September 2024: https://www.wcrf.org/cancer-trends/colorectal-cancer-statistics/.

3. D. Kastenberg, G. Bertiger, and S. Brogadir, “Bowel preparation quality scales for colonoscopy”, *World Journal of Gastroenterology*, vol. 24, pp 2833-2843, Jul. 2018.

4. M. T. Abreu, and N. Harpaz, “Diagnosis of Colitis: Making the Initial Diagnosis”, *Clinical Gastroenterology and Hepatology*, vol. 5, pp 295-301, Mar. 2007.

5. G. Chen, Y. Zhao, F. Xie, W. Shi, Y. Yang, A. Yang, and D. Wu, “Educating Outpatients for Bowel Preparation Before Colonoscopy Using Conventional Methods vs Virtual Reality Videos Plus Conventional Methods”, *JAMA Network Open*, Nov. 2021.

6. B. Walter, Rena Frank, L. Ludwig, N. Dikopoulos, M. Mayr, B. Neu, B. Mayer, A. Hann, B. Meier, K. Caca, T. Seufferlein, AND A. Meining, “Smartphone Application to Reinforce Education Increases High-Quality Preparation for Colorectal Cancer Screening Colonoscopies in a Randomized Trial”, *Clinical Gastroenterology and Hepatology*, vol. 19, pp 331-338, Feb. 2021.

7. B. M. Walter, P. Klare, B. Neu, R. M. Schmid, and S. von Delius, “Development and Testing of an Automated 4-Day Text Messaging Guidance as an Aid for Improving Colonoscopy Preparation”, *JMIR Mhealth Uhealth*, vol. 4, Jun. 2016.

8. C. Hassan, J. East, F. Radaelli, C. Spada, R. Benamouzig, R. Bisschops , M. Bretthauer , E. Dekker , M. Dinis-Ribeiro, M. Ferlitsch, L. Fuccio, H. Awadie, I. Gralnek, R. Jover, M. F. Kaminski, M. Pellisé, K. Triantafyllou, G. Vanella, C. Mangas-Sanjuan, L. Frazzoni, J. E. Van Hooft, and J. M. Dumonceau, “Bowel preparation for colonoscopy: European Society of Gastrointestinal Endoscopy (ESGE) Guideline – Update 2019”, *Endoscopy*, vol. 51, pp 775-794, 2019.

9. B. Lebwohl, F. Kastrinos, M. Glick, A. J. Rosenbaum, T. Wang, and A. I. Neugut, “The Impact of Suboptimal Preparation on Adenoma Miss Rates and the Factors Associated with Early Repeat Colonoscopy”, *Gastrointestinal Endoscopy*, vol. 73, pp 1207-1214, Jun. 2011.

10. M. Cristea, G. G. Noja, P. Stefea, and A. L. Sala, “The Impact of Population Aging and Public Health Support on EU Labor Markets”, *International journal of environmental research and public health*, vol. 17, Feb. 2020.

11. B. Glicksberg, B. Schuller, B. Rodriguez, D. Ting, D. Bates, E. Schaden, H. Peng, H. Willschke, J. van der Laak, J. Car, K. Rahimi, L. Celi, M. Banach, M. Kletecka-Pulker, O. Kimberger, R. Eils, S. Islam, S. Wong, T. Wong, S. Brunak, A. Atanasov, A. Yeung, A. Torkamani, A. Butte, and W. Gao, “The promise of digital healthcare technologies”, *Frontiers in Public Health*, vol. 11, Sep. 2023.

12. B. M. C. Silva, J. J. P. C. Rodrigues, I. de la Torre Díez, M. López-Coronado, and K. Saleem, “Mobile-health: A review of current state in 2015”, *Journal of biomedical informatics*, vol. 56, pp 265-272, Aug. 2015.

13. S. Gomaa, J. Posey, B. Bashir, A. B. Mallick, E. Vanderklok, M. Schnoll, T. Zhan, and K. Y. Wen, “Feasibility of a Text Messaging–Integrated and Chatbot-Interfaced Self-Management Program for Symptom Control in Patients With Gastrointestinal Cancer Undergoing Chemotherapy: Pilot Mixed Methods Study”, *JMIR Formative Research*, vol. 7, Nov. 2023.

14. D. Albino de Queiroz a c , R. Silva Passarello, V. Veloso de Moura Fé, A. Rossini, E. Folchini da Silveira, E. A. I. Fonseca de Queiroz, and C. André da Costa, “A wearable chatbot-based model for monitoring colorectal cancer patients in the active phase of treatment”, *Healthcare Analytics*, vol. 4, Dec. 2023.

15. A. M. Turing, “Computing Machinery and Intelligence”, *Mind*, 1950.

16. J.-E. Bibault, B. Chaix, A. Guillemassé, S. Cousin, A. Escande, M. Perrin, A. Pienkowski, G. Delamon, P. Nectoux, and B. Brouard, “A Chatbot Versus Physicians to Provide Information for Patients With Breast Cancer: Blind, Randomized Controlled Noninferiority Trial”, *Journal of medical Internet research*, 2019.

17. L. Xu, L. Sanders, K. Li, and J. C. L. Chow, “Chatbot for Health Care and Oncology Applications Using Artificial Intelligence and Machine Learning: Systematic Review”, *JMIR Cancer*, vol. 7, Nov. 2021.

18. B. Chaix, J. E. Bibault, A. Pienkowski, G. Delamon, A. Guillemassé, P. Nectoux, and B. Brouard, “When Chatbots Meet Patients: One-Year Prospective Study of Conversations Between Patients With Breast Cancer and a Chatbot”, *JMIR Cancer*, vol. 5, May 2019.

19. R. L. Siegel, N. S. Wagle, A. Cercek, R. A. Smith, and A. Jemal, “Colorectal cancer statistics, 2023”, *CA: A Cancer Journal for Clinicians*, Mar. 2023.

20. B. Heald, E. Keel, J. Marquard, C. A. Burke, M. F. Kalady, J. M. Church, D. Liska, G. Mankaney, K. Hurley, and C. Eng, “Using chatbots to screen for heritable cancer syndromes in patients undergoing routine colonoscopy”, *Journal of Medical Genetics*, vol. 58, Nov. 2020.

21. P. Lee, S. Bubeck, and J. Petro, “Benefits, Limits, and Risks of GPT-4 as an AI Chatbot for Medicine”, *New England Journal of Medicine*, vol. 388, Mar. 2023.

22. A. de Wynter, X. Wang, A. Sokolov, Q. Gu, and S.-Q. Chen, “An evaluation on large language model outputs: Discourse and memorization”, *Natural Language Processing Journal*, vol. 4, Sep. 2023.

23. M. Cristea, G. G. Noja, P. Stefea, and A. L. Sala, “The Impact of Population Aging and Public Health Support on EU Labor Markets”, *International journal of environmental research and public health*, vol. 17, Feb. 2020.

24. C. Golz, A. Oulevey Bachmann, T. Sala Defilippis, A. Kobleder, K. A. Peter, R. Schaffert, X. Schwarzenbach, T. Kampel, and S. Hahn, “Preparing students to deal with the consequences of the workforce shortage among health professionals: a qualitative approach”, *BMC Medical Education*, vol. 22, Nov. 2020.

25. Microsoft, “Configure a bot to run on one or more channels”, Nov. 2022, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/azure/bot-service/bot-service-manage-channels

26. Microsoft, “botframework-sdk”, May 2024, Accessed on 30 September 2024: https://github.com/microsoft/botframework-sdk

27. Microsoft, “Configure a bot to run on one or more channels”, Nov. 2022, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/azure/bot-service/bot-builder-basics

28. Microsoft, “Basics of the Microsoft Bot Framework”, Nov. 2022, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview

29. Microsoft, “Introduction to .NET”, Jan. 2024, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/dotnet/core/introduction

30. Microsoft, “What is Visual Studio?”, Jan. 2024, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide

31. S. Nazareth, L. Hayward, E. Simmons, M. Snir, K. E. Hatchell, S. Rojahn, R. N. Slotnick, and R. L. Nussbaum, “Hereditary Cancer Risk Using a Genetic Chatbot Before Routine Care Visits”, *Obstetrics & Gynecology*, vol. 138, pp 860-870, Dec. 2021.

32. Microsoft, “What is the Bot Framework SDK?”, Nov. 2022, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/azure/bot-service/bot-service-overview

33. Microsoft, “Test and debug with the Emulator”, Oct. 2023, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/azure/bot-service/bot-service-debug-emulator

34. Microsoft, “Managing state”, Oct. 2022, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/azure/bot-service/bot-builder-concept-state

35. Microsoft, “Dialogs library”, Nov. 2022, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/azure/bot-service/bot-builder-concept-dialog

36. Microsoft, “Continuous Integration and Continuous Delivery (Building Real-World Cloud Apps with Azure)”, May 2022, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/aspnet/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/continuous-integration-and-continuous-delivery

37. Microsoft, “Quickstart: Install and use a NuGet package in Visual Studio (Windows only)”, Aug. 2023, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/nuget/quickstart/install-and-use-a-package-in-visual-studio

38. Microsoft, “Azure for developers overview”, Oct. 2024, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/azure/developer/intro/azure-developer-overview

39. Microsoft, “Azure encryption overview”, Apr. 2024, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/azure/security/fundamentals/encryption-overview

40. Microsoft, “Security in Azure App Service”, Jun. 2023, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/azure/app-service/overview-security

41. Microsoft, “How do I create and manage resources in Azure?”, Oct. 2024, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/azure/developer/intro/azure-developer-create-resources

42. DeepL, “DeepL Translate API | Machine Translation Technology”, May 2022, Accessed on 30 September 2024: https://www.deepl.com/en/pro-api

43. A. Yulianto, and R. Supriatnaningsih, “Google Translate vs. DeepL: A quantitative evaluation of close-language pair translation (French to English)”, *AJELP: Asian Journal of English Language and Pedagogy*, vol. 9, Dec. 2021.

44. C. M. Hidalgo-Ternero, “Google Translate vs. DeepL: analysing neural machine translation performance under the challenge of phraseological variation”, *MonTI. Monografías de Traducción e Interpretación*, Jan. 2021.

45. Y. Takakusagi, T. Oike, K. Shirai, H. Sato, K. Kano, S. Shima, K. Tsuchida, N. Mizoguchi, I. Serizawa, D. Yoshida, T. Kamada, and H. Katoh, “Validation of the Reliability of Machine Translation for a Medical Article From Japanese to English Using DeepL Translator”, *Cureus*, vol. 13, Sep. 2021.

46. M. I. Kamaluddin, M. W. K. Rasyid, F. H. Abqoriyyah, and A. Saehu, “Accuracy Analysis of DeepL: Breakthroughs in Machine TranslationTechnology”, *Journal of English Education Forum (JEEF)*, vol. 4, Jun. 2024.

47. Python, “Beginners Guide and Overview”, Nov. 2022, Accessed on 30 September 2024: https://wiki.python.org/moin/BeginnersGuide/Overview

48. Microsoft, “Application performance FAQs for Web Apps in Azure”, Mar. 2024, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/troubleshoot/azure/app-service/web-apps-performance-faqs#how-do-i-decrease-the-response-time-for-the-first-request-after-idle-time

49. Microsoft, “ping”, Sep. 2023, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/ping

50. Dale Mellor, “Mcron - User Requirements and Analysis.”, Jun. 2003, Accessed on 30 September 2024: https://www.gnu.org/software/mcron/design.html

51. Quartz.NET, “Quartz.NET Open-source job scheduling system for .NET”, 2007, Accessed on 30 September 2024: https://www.quartz-scheduler.net/

52. F. A. Krueger, “sqlite-net”, Mar. 2024, Accessed on 30 September 2024: https://github.com/praeclarum/sqlite-net

53. SQLite, “SQLite Home Page”, Aug. 2024, Accessed on 30 September 2024: https://www.sqlite.org/

54. D. Harltey, “Fastenshtein”, Jun. 2024, Accessed on 30 September 2024: https://github.com/DanHarltey/Fastenshtein/blob/master/docs/nuget-readme.md

55. S. Yolchuyeva, G. Németh, and B. Gyires-Tóth, “Text normalization with convolutional neural networks”, *International Journal of Speech Technology*, vol. 21, pp 589-600, May 2018.

56. L. F. Costa, “Further Generalizations of the Jaccard Index”, *arXiv*, Nov. 2021.

57. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, “Introduction To Algorithms”, *The MIT Press*, Apr. 2022.

58. S. Ceri, A. Bozzon, M. Brambilla, E. Della Valle, P. Fraternali, and S. Quarteroni, “An Introduction to Information Retrieval”, in “Web Information Retrieval”, *Springer*, 2013.

59. J. Leskovec, A. Rajaraman, and J. D. Ullman, “Mining of Massive Datasets”, *Cambridge University Press*, Jan. 2020.

60. S. Robertson, and H. Zaragoza, “The Probabilistic Relevance Framework: BM25 and Beyond”, *Foundations and Trends in Information Retrieval*, vol. 3, pp 333-389, Dec. 2009.

61. D. Jurafsky, and J.H. Martin, “Speech and Language Processing”, *Prentice Hall*, Aug. 2024.

62. FastText, “Word vectors for 157 languages”, 2022, Accessed on 30 September 2024: https://fasttext.cc/docs/en/crawl-vectors.html

63. D. Beeferman, “Datamuse API”, Dec. 2016, Accessed on 30 September 2024: https://www.datamuse.com/api/

64. P. Danziger, “Big O Notation”, *Toronto Metropolitan University*, Apr. 2010.

65. Microsoft, “Azure Pipelines architecture for Azure Web Apps”, Jul. 2023, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/azure/devops/pipelines/architectures/devops-pipelines-azure-web-apps-architecture

66. GitHub, “About workflows”, 2024, Accessed on 30 September 2024: https://docs.github.com/en/actions/writing-workflows/about-workflows

67. Microsoft, “Provision and publish a bot”, Jul. 2024, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/azure/bot-service/provision-and-publish-a-bot#azure-resources

68. Telegram, “From BotFather to 'Hello World'”, Accessed on 30 September 2024: https://core.telegram.org/bots/tutorial

69. Bot Builder Community, “Bot Framework Community - .NET Components & Extensions”, 2022, Accessed on 30 September 2024: https://github.com/botbuildercommunity/botbuilder-community-dotnet#adapters

70. Infobip, “Support center”, 2024, Accessed on 30 September 2024: https://support.infobip.com/

71. Infobip, “Infobip API documentation: Build Messaging for Your Customers”, 2022, Accessed on 30 September 2024: https://www.infobip.com/docs/api

72. Meta, “WhatsApp Business Terms of Service”, Feb. 2024, Accessed on 30 September 2024: https://www.whatsapp.com/legal/business-terms/

73. A. Nogueira-Rodríguez, R. Domínguez-Carbajales, H. López-Fernández, Á. Iglesias, J. Cubiella, F. Fdez-Riverola, M. Reboiro-Jato, and D. Glez-Peña, “Deep Neural Networks approaches for detecting and classifying colorectal polyps”, *Neurocomputing*, vol. 423, pp 721-734, Jan. 2021.

74. R. J. Meuth, “GPUs surpass computers at repetitive calculations”, *IEEE Potentials*, vol. 26, Nov. 2007.

75. United States government, “Introduction to HTTPS”, Accessed on 30 September 2024: https://https.cio.gov/faq/

76. A. Poschmann, S. Ling, H. Wang, “256 bit standardized crypto for 650 GE–GOST revisited”, *Springer*, pp 219–233, 2010.

77. Microsoft, “About Azure Key Vault”, Sep. 2024, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/azure/key-vault/general/overview

78. Microsoft, “Azure Key Vault keys, secrets and certificates overview”, Aug. 2024, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/azure/key-vault/general/about-keys-secrets-certificates

79. Microsoft, “Rotate keys in Azure AI services”, Aug. 2024, Accessed on 30 September 2024: https://learn.microsoft.com/en-us/azure/ai-services/rotate-keys

80. Intersoft Consulting Services AG, “General Data Protection Regulation GDPR”, May 2018, Accessed on 30 September 2024: https://gdpr-info.eu/

81. Intersoft Consulting Services AG, “Right to be Informed”, May 2018, Accessed on 30 September 2024: https://gdpr-info.eu/issues/right-to-be-informed/

82. GDPR Advisor, “Less is More: The Importance of Data Minimization in GDPR Compliance”, 2023, Accessed on 30 September 2024: https://www.gdpr-advisor.com/data-minimization/

83. Intersoft Consulting Services AG, “Right to be Forgotten”, May 2018, Accessed on 30 September 2024: https://gdpr-info.eu/issues/right-to-be-forgotten/

84. Signal, “Signal Home”, 2024, Accessed on 30 September 2024: https://signal.org/

85. Google, “Signal”, Sep. 2024, Accessed on 30 September 2024: https://play.google.com/store/apps/details?id=org.thoughtcrime.securesms

86. Google, “Telegram”, Sep. 2024, Accessed on 30 September 2024: https://play.google.com/store/apps/details?id=org.telegram.messenger

87. Google, “WhatsApp Messenger”, Sep. 2024, Accessed on 30 September 2024: https://play.google.com/store/apps/details?id=com.whatsapp

88. C. E. Bogos, R. Mocanu, and E. Simion, “A security analysis comparison between Signal, WhatsApp and Telegram”, *Cryptology ePrint Archive*, Jan. 2023.

9. Appendix

Link to the project's repository: https://github.com/Jarturog/ColpaBot