Computationally clarifying user intent for improved question answering

Ari-Pekka Härkönen

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 1.12.2021

Supervisor

Prof. Mikko Kurimo

Advisors

MSc Juho Leinonen

MSc Jeri Haapavuo

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Ari-Pekka Härkönen

Title Computationally clarifying user intent for improved question answering

Degree programme Master's Programme in Computer, Communication and Information Sciences

Major Machine Learning, Data Science and Artificial Intelligence **Code of major** SCI3044

Supervisor Prof. Mikko Kurimo

Advisors MSc Juho Leinonen, MSc Jeri Haapavuo

Date 1.12.2021 **Number of pages** 43+8 **Language** English

Abstract

Chatbots have become a standard feature of modern websites and other digital services. Although they are able to perform many basic functions, they still lack essential features of human interaction. One of them being the ability to use domain knowledge and information about the conversation history to infer the user's meaning. This is often replaced by simply asking for the user to rephrase their message. This thesis explores the possibility of improving this functionality by adding domain knowledge to the chatbot via ontologies. The goal of the implementation is to improve the chatbot's abilities to recommend related articles and data using the information from the ontology, allowing the user to find the knowledge they were looking for even if the chatbot was unable to find the correct answer at first.

Ontologies are graph presentations that detail a part of a language or some specific domain. In this thesis an ontology describing the field of computer science (Computer Science Ontology) is integrated into a chatbot system built to answer technical questions about the Ubuntu operating system. This chatbot is built using Rasa, an open source framework for building machine learning powered dialogue systems. Rasa's existing features are extended with a custom implementation for handling low-confidence user inputs. These inputs are analyzed and the relevant entities found from the message are used to query the aforementioned ontology for related terms. These are then formatted into follow-up messages and sent back to the user.

This implementation was evaluated by a test group consisting of 20 people. The results of this user evaluation do not show a significant change when compared to the same groups evaluation of the baseline model. The main cause for this seems to be the ontology used, which doesn't cover the Ubuntu domain well enough for the recommendations to be relevant. The implementation built is able to retrieve the related entities from the ontology, but further research is required to discover the actual performance of the implementation.

Keywords Natural Language Understanding, Ontology, Chatbot

Aalto University
School of Science

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Ari-Pekka Härkönen

Työn nimi Käyttäjän tarkoituksen selventäminen laskennallisesti

Koulutusohjelma Tietotekniikka

Pääaine Koneoppiminen, tilastotiede ja tekoäly **Pääaineen koodi** SCI3044

Työn valvoja Prof. Mikko Kurimo

Työn ohjaajat TkT Juho Leinonen, Di Jeri Haapavuo

Päivämäärä 1.12.2021 **Sivumäärä** 43+8 **Kieli** Englanti

Tiivistelmä

Chatboteista on lyhyessä ajassa tullut perusominaisuus moderneilla nettisivuilla sekä muissa digitaalisissa palveluissa. Ne pystyvät hoitamaan yksinkertaiset tehtävät ja interaktiotilanteet hyvin, mutta useita ihmismäisen vuorovaikutuksen piirteitä yhä puuttuu. Yksi näistä on toimialaan liittyvän terminologian ja keskusteluhistorian hyödyntäminen. Tämän tilalla on usein yksinkertainen kehotus kysyä joku toinen kysymys. Tämän tutkielman tavoitteena on selvittää, voisiko chatbottien toiminnallisuutta tällä saralla parantaa lisäämällä niiden alakohtaista tietoa ontologioiden avulla. Ontologiat ovat graafikuvauksia kielien ja toimialojen rakenteesta. Implementaatio hyödyntää erästä tällaista ontologiaa koittaen sen avulla jatkaa keskustelua ja vastata alkuperäiseen kysymykseen, vaikka chatbotin perustoiminnot eivät olisi siihen pystyneet.

Tässä tutkielmassa käytetty ontologia, Computer Science Ontology, kuvaa tietotekniikan kenttää kokonaisuutena. Se integroitiin chatbottiin, joka on rakennettu vastaamaan Ubuntu-käyttöjärjestelmään liittyviin kysymyksiin. Tämän chatbotin kehitykseen on käytetty Rasa-kehikkoa, joka mahdollistaa tekoälypohjaisten chatbottien kehityksen. Käyttäjien kysymykset, jotka jäävät ilman vastausta analysoidaan ja merkittävistä termeistä muodostetaan ontologiakyselyt. Näiden kyselyjen tuloksista muotoillaan käyttäjälle näytettävät jatkoviestit, jotka pyrkivät vastaamaan heidän alkuperäisiin kysymyksiinsä.

Tutkielman implementaatiota evaluoi 20 hengen testiryhmä. Tämän käyttäjätestin tulokset eivät merkittävästi eroa toteutuksen pohjana toimineen mallin tuloksista. Suurin syy tähän vaikuttaa olleen käytössä ollut ontologia, joka ei kattanut Ubuntuun liittyvää terminologiaa riittävän hyvin, jolloin muodostetut vastaukset jäivät epärelevanteiksi. Implementaation oikean potentiaalin selvittäminen vaatii jatkotutkimusta ja toteutuksen vientiä tilanteeseen, jossa ontologia kattaa lähtödatan paremmin.

Avainsanat Luonnollisen kielen ymmärrys, ontologia, chatbot

Preface

Firstly I would like to thank the good people of Atostek for making it possible for me to delve into an interesting topic and complete my thesis while surrounded by such motivated and smart people. Thank you to Jeri Haapavuo for your valuable guidance throughout the process, and to Tomi Javanainen, Ilkka Hannula and Simo Antikainen for providing me with the necessary framework and support for me to work on the implementation of this thesis.

I am grateful to my supervisor, Professor Mikko Kurimo, and my advisor from Aalto, Juho Leinonen, who provided me with first-class advice and insight. Your expertise on the domain has been beyond helpful.

Finally I would like to thank my friends and family who supported me when I needed it most.

Otaniemi, 1.12.2021

Ari-Pekka Härkönen

Contents

Abbreviations

AI	Artificial Intelligence
AIML	Artificial Intelligence Markup Language
CFR	Conditional Random Field
CNN	Convolutional Neural Network
CP	Content Ontology Design Pattern
CSO	Computer Science Ontology
HMM	Hidden Markov Model
LSTM	Long Short-Term Memory
NLG	Natural Language Generation
NLU	Natural Language Understanding
NLP	Natural Language Processing
OP	Ontology Design Pattern
RNN	Recurrent Neural Network
SVM	Support Vector Machine

1 Introduction

Chatbots have made their way into virtually every website and service. While they have advanced rapidly during the last decade, there are still considerable drawbacks: goal-oriented chatbots struggle to comprehend inputs that do not directly relate to their purpose, often produce strange or rude responses, and fail to provide a user experience consistently up-to-par with a human performing the same task. [1] Touching on all of these drawbacks, this thesis explores the possibilities of introducing the ability to generate follow-up messages by extending modern chatbot frameworks and using them to gather more relevant information from the user while simultaneously trying to make the flow of the conversation more natural. In situations, where the chatbots can not comprehend the user with a decent probability, the chatbot can utilize domain knowledge to generate follow-up questions and suggestions. These are used to clarify the user's intent to the system and thus reduce the amount of meaningless chatter and stalemate situations in the conversation.

Utilizing follow-up questions and domain expertise to gain more information in a conversation is something humans do regularly and keeping the end-goal in mind over many exchanges is a given. Current chatbot technologies however often can not keep track of the end goal as proficiently as humans. Furthermore, combining information from multiple different messages across the conversation history is hard for them, often leading to "unnatural" messages in long conversations.

1.1 Methodology

A chatbot developed using the Rasa Open Source framework is experimented with. We examine the possibilities of using a domain-specific ontology, a presentation of how the entities in the data relate to one another, to improve the way the chatbot understands the context it is dealing with. In addition, Rasa's existing functionality is extended with a more sophisticated module for handling low-confidence interpretations of the user input and forming follow-up messages combining the information from the input together with domain-specific knowledge. The user's intents are expressed in Rasa as distinct entities, also called intents, that lead to specific actions. In other words, the implemented module will try to help in deducing the correct intent when Rasa can not do it by itself.

1.2 Research questions

The research questions this thesis seeks to answer are:

- Can ontologies help in forming follow-up messages in cases where the output can not be correctly interpreted by an NLP system?
- How much can the implemented fallback-module improve performance in terms of giving the expected response faster and with less effort (than the default implementation)?

- What methods or ontologies provide the most relevant follow-up messages?

1.3 Thesis structure

First, this thesis will provide the necessary background on current chatbot technologies and the technical details needed for understanding the proposed solutions. The concept of ontologies is introduced together with its current state-of-the-art applications and the Rasa framework and its relevant functionalities are described.

After the reader is familiarized with the relevant concepts and technologies, the thesis introduces the implementation created and the methods used to evaluate that solution. The implementation includes a custom Rasa module for handling low-confidence user inputs and a description of the dataset and ontology used. The evaluation methods are introduced along with their applicability and potential downsides.

The next chapter goes over the results of applying the aforementioned solutions to the problem and assesses how well they performed using the metrics introduced. These results are then reflected and expanded on in the Discussion and Conclusions chapter. The results are evaluated and topics of further research and points of interest are discussed.

2 Background

This section introduces the central concepts and technologies of the thesis. First, the general concepts behind modern chatbots are presented with an emphasis on goal-oriented chatbots. Then the core concepts of ontologies are summarized and the usage of them in similar chatbot implementations is presented. Lastly, the background information about chatbots and ontologies is combined with the Rasa framework, and its tools and relevant functionality are introduced.

2.1 Chatbots

A chatbot is a system that can engage in a conversation using natural languages in either text or speech form or both. In this thesis, the terms chatbot, dialogue system, and conversational agent are used interchangeably. These systems are built for example to converse with humans, complete a set of tasks or answer user's questions about a topic. Next, the history of chatbot development is briefly introduced to help understand how modern chatbot frameworks came to be.

Modern chatbots can be traced back to ELIZA, developed by Joseph Weizenbaum [2]. Developed during the 1960s, ELIZA utilized early NLP techniques, such as word substitution, to mimic a conversation based on the user's latest message. Some keywords from the user's input are chosen and fitted into templates, thus forming the next output of the program. While sometimes engaging, ELIZA could not consistently pass the Turing test [3], a common goal for early conversational agents. The test consists of a human evaluator engaging in two conversations via computer screens or other written media, one of the two other agents is a machine, and the other one a human. If the evaluator can not tell which one of the two is the machine, the conversational agent passes the test. Turing test remains a relevant metric to this day, although in many cases other metrics are needed as well. More on the evaluation methods used in this thesis later in this section.

After ELIZA, the next significant publication took place in 1995, when Richard Wallace developed A.L.I.C.E., short for Artificial Linguistic Internet Computer Entity [4]. A.L.I.C.E. was an important step in the development of chatbots for its XML schema called Artificial Intelligence Markup Language (AIML). This was the first time the rules of a dialogue system could be easily modified with a standardized specification. AIML consists of words, spaces, and wildcard symbols that allowed the pattern matching technique to find the most suitable pattern. This improved way of specifying the rules for the program made it possible for A.L.I.C.E. to form more complex linguistic structures and thus allowed improved performance compared to its predecessors.

An integral part of modern dialogue systems is the ability to understand audio input and turn speech to text and vice-versa. Combining this capacity with the advances in NLP techniques made modern customer support solutions and dialogue systems such as Apple's Siri and Google Assistant a reality. These applications that brought dialogue systems to regularly used devices and services made them a mainstream solution and made them a must-have feature in heavily used platforms and modern web solutions. Smart home solutions have also proven as an important application for dialogue systems that continue developing.

This concludes the history of dialogue system development thus far. In the context of this thesis, a chatbot refers to a system that can engage in a turn-based interaction with its user utilizing an interface of natural language. The user's input, which can be either a written piece of text or a predefined option, is processed by the dialogue system and a response message is generated as a result. The modules inside a dialogue system that perform these tasks depend on the architecture of the software. Common divisions between dialogue systems and their architectures are introduced in the next sections, which describe how chatbots are developed. First, the differences between rule-based and AI-based models are described.

2.1.1 Rule-based and AI-based models

Rule-based models are built to follow a set of rules that guide the dialogue system from one state to another. The historical systems introduced above fall into the rule-based category since their actions are based solely on following a set of relatively simple algorithms. As an example, ELIZA uses its pattern matching algorithm to build a response from the user's input. This kind of approach has remained relevant until this day, controlling a dialogue system with a set of rules provides perfect control of the dialogue states and possible outcomes of inputs. A key feature of this approach is representing the relevant information in the user input as slots in the dialogue system [5]. The system identifies the value for the slots from the user input and proceeds to fill the remaining required slots to carry out the user's intent. For each intent that the system is capable of understanding, there must exist a sequence of actions that the system performs after it recognizes the intent. Defining the rules and actions for such systems by hand can quickly become a cumbersome task when the system scales. Syntaxes such as the aforementioned AIML are thus critical for easier scaling and management of large dialogue systems. Figure 1 demonstrates a simple rule-based dialogue and how the system fills the slots by directly asking the user for each value it needs to complete its task.

Rule-based systems lack the flexibility of AI-based chatbots and are not as easily scalable to handle large cases. They are also restricted by the ruleset and inputs that do not match any defined rules can not be properly interpreted by the system. In addition, this means that developing the rules for such a system requires some amount of linguistic expertise, as the words can be written in multiple forms and grammatical tenses. AI-based models offer greater scalability than rule-based models by removing the need for defining rules for the dialogue system. Instead, they are built by training a neural network or a machine learning application on a set of relevant data.

AI-based chatbots utilize machine learning algorithms to generate a response based on the data provided. This allows machine learning algorithms to keep on learning

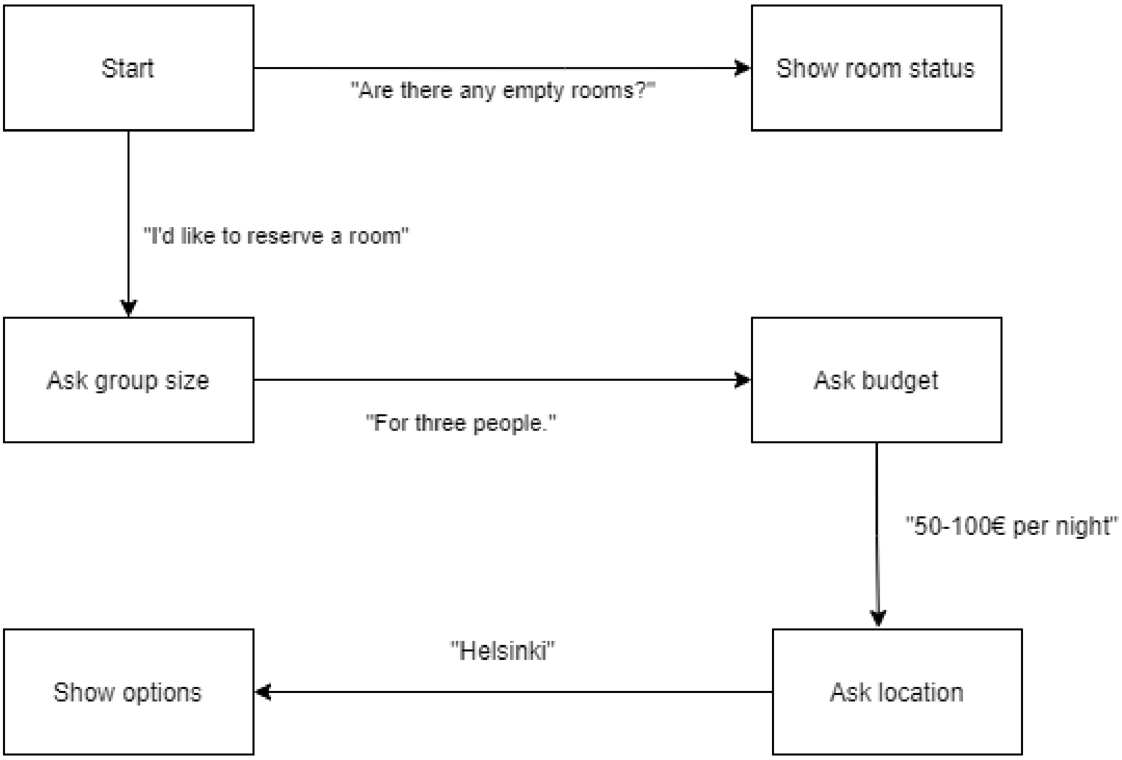


Figure 1: A figure displaying how a very simple rule-based chatbot handles user's inputs and moves the conversation forward. the lines represent the user's messages and the boxes detail the functionality that these messages trigger.

and improving based on the previous learning models and data, enabling constant improvement. With the rapid advancements in the field during recent years as well as increases in computational power, developing such models has become efficient and many proficient frameworks for developing AI-based models exist. More details on the different natural language understanding and generation methods of rule-based and AI-based models are introduced in the upcoming sections. First, another way to categorize dialogue systems is described.

2.1.2 Goal-oriented and conversational models

Another typical categorization of chatbots is the division to goal-oriented models or non-goal-oriented, conversational models. This section describes this division, providing more context on common design patterns of chatbots. Research surrounding dialogue systems started around conversational models with close ties to evaluation metrics such as the Turing test [2]. With technologies such as AIML emerging, goal-oriented chatbots became a viable application for dialogue systems. Building a dialogue system with a specific goal in mind with such rule-based approaches was relatively fast since the technologies allowed the developer to focus only on the use case at hand. In this section, the differences between goal-oriented chatbots and conversational models are reflected.

Goal-oriented systems are built to complete some specific task, such as booking a hotel or reserving a table from a restaurant. Customer service chatbots like these have paved their way to the mainstream during the last decade and have become a viable option for completing these manual, repetitive tasks. A critical component of a goal-oriented system is a slot-filling mechanism [6]. Slot-filling is a way to perform entity recognition and store the found entities during a dialog. Slots are a set of values that the dialogue system keeps track of during the conversation and uses to complete the task it set out to do. In the case of a hotel reservation, this could mean tracking the desired price, location, and the number of people in the group. The values for these slots are extracted from the user's messages. The slot system is reliable and easy to configure but does not scale well because of the manual labor needed to determine the slots. As a solution to this, end-to-end systems that have all of their components trained on actual dialogues have been able to circle this problem [7].

Conversational models have improved swiftly due to the advancements in machine learning techniques and the increase in computational power by allowing the development of AI-based models that learn the context data instead of using a strict set of rules. Although their real-world use cases are not as easily recognizable as the use cases of their goal-oriented peers, their role in research is indisputable: building a conversational model requires one to solve many different tasks related to natural language processing which all contribute towards the larger goal and have important real-world applications themselves, such as sentiment analysis and text summarization. Conversational models require a solid understanding of their target language so that they can keep a conversation natural no matter the topic. Thus they have been considered a harder challenge to build than goal-oriented models. Passing the Turing test has been the grand goal of such conversational models and around it, in 1991 Hugh Loebner developed the Loebner prize competition [8]. Loebner's idea was to start an annual competition where the first model to pass an unrestricted Turing test would win a prize of $100,000, with smaller prizes for the best performing models on the restricted Turing test.

While driving the research around dialogue systems, methods used to build conversational models also helped goal-oriented chatbots to develop. The next sections describe different ways chatbot systems handle user input and produce output messages. These tasks are performed by natural language processing algorithms, which we will split into two sections. First, the methods to decipher the user's input are described in the natural language understanding section. Then the methods to produce responses are described in the section about natural language generation. These tasks are important for any dialogue system and understanding the background described in the next sections will clarify the motivation for the implementation part of this thesis as well as the parts of a chatbot system the implementation aims to better. Since the chatbot related to this thesis is goal-oriented in nature, a focus on relevant methods is retained.

2.1.3 Natural language understanding

With the different categorizations of dialogue systems (goal-oriented and non-goal-oriented, rule-based and AI-based) in mind, a closer look at the internal functionalities of chatbots is in place. All dialogue systems need to be able to gather some information from the user's messages. This task is called natural language understanding (NLU) and is usually understood as two subtasks: intent classification and entity recognition [9]. Historically rule-based chatbots rely on a set of different algorithms to perform these operations, whereas AI-based solutions use methods such as probabilistic modeling, linear classifiers and likelihood maximization or neural networks trained with large datasets of vector representations of words. Intent classification is the task of determining what the user wants to achieve with their message, for example book a hotel. Entity recognition is the task of recognizing the relevant entities from the message, for example the city they want to stay in.

Early conversational models and rule-based systems were built to use simple linguistic methods, such as pattern matching and keyword recognition. These provided a solid base for understanding user's input for the early models and modern rule-based agents with little computational power required: Entity recognition and intent classification are performed by applying a set of rules to the user's messages and if relevant words or sequences of words are found the system acts accordingly. This is still a relevant slot-filling mechanism since the possible values for a model's slots are often limited and known by the developers of such a system. In the aforementioned hotel room booking case, an example of slot-filling by keyword recognition could be that the user is asked for a preferred location of the hotel with possible values for that slot being Berlin, Hamburg, Düsseldorf. If the answer contains one of these names, the slot is filled with the matching value. Additionally, a more sophisticated system can be able to determine the desired answer if the user has typed it wrong or if there are some grammatical alterations in the answer. Regular expressions and hard-coded rules can be used to build quite sophisticated systems, although they are not easily scaled to larger use cases.

Traditional machine learning methods allow the model to perform NLU tasks without the need for handcrafted rules. Instead, statistical methods are used to train a model on a set of training data that allows the model to gather its own knowledge and set of rules. This greatly reduces the workload of the developers but also increases uncertainty over the quality of the results. Since the process of learning is not completely visible to the developer, the final model can only be assessed upon completion. As even small changes to the training data or the model itself can cause significant changes in the results, the process can become cumbersome and time-consuming. Word classification and clustering are both tasks well suited for such models since data points (e.g training sentences) are easy to generate and include in the training set and the task is simple enough to not require very large training sets.

Neural network-based solutions are similar to traditional machine learning models. Because the data fed into a neural network model is a direct vector representation of training sentences, feature engineering is not required. Neural network models require very large training datasets compared to more traditional methods and offer even less predictability during the training process, again the model can only be assessed once training is completed. The task of entity recognition and the subsequent slot-filling has been tackled with deep learning methods, utilizing Hidden Markov Models (HMM), conditional random fields (CFR), and support vector machines (SVM) [10]. Although the subject has been an active topic of research for the past years, slot-filling still gathers much attention and new approaches to the problem are explored [11], [12]. The other component of NLU, intent classification, is well suited for deep learning approaches as it can be reduced to a classification task. Classification using deep learning has been tackled using convolutional neural networks (CNN) [13], attention-based CNNs [14], long short-term memory (LSTM) [15] and hierarchical attention networks [16].

Recent research has introduced models that perform both intent classification and entity recognition such as Bidirectional Encoder Representations from Transformers (BERT), that are able to outperform separate deep learning modules for each task [17].

2.1.4 Natural language generation

After performing NLU tasks, the dialogue system needs to be able to respond to the user's message with a coherent sentence. To produce this, natural language generation (NLG) methods are used. This task of converting the input data into output text is often divided into multiple subtasks [18]. The common division presented in [18] includes the following subtasks:

- Content determination: what information should be included in the output.
- Text structuring: the order in which the information will be presented.
- Sentence aggregation: deciding which information to present in individual sentences.
- Lexicalization: choosing the words and expressions to use.
- Referring expression generation: selecting the words and phrases to identify domain objects.
- Linguistic realization: combining all chosen words and phrases into the final output sentences.

When categorizing the methods used to perform these tasks, a similar division to NLU methods is natural. Early methods utilized rule-based approaches and templates whereas more modern approaches rely on probabilistic methods and deep learning.

Early rule-based models were able to circle around many of the issues by relying on templates. The models contained a set of sentence templates that are populated with the information provided by the user. The system ensures grammatically correct sentences but easily becomes repetitive since building a complex rule system requires a lot of work to be a versatile conversational partner.

As with NLU, deep learning methods have made their way into NLG tasks. A significant architecture that has become a base for many current applications is the Encoder-Decoder framework [19]. A recurrent neural network (RNN) encodes an input into a vector representation. This is then given as input to a decoder RNN. This architecture is capable of handling inputs and outputs of different sizes, making it well suited for sequence transformation problems like machine translation. An example of this is given in figure 2. A system with this architecture can not only use the user's input to provide a response but can also be built to utilize the knowledge that some other model or algorithm is able to extract from the message or the chatbot's own domain knowledge. This of course requires proper training of the models, but can also lead to more intriguing answers from the dialogue systems. The methods used to produce the replies in the implementation of this thesis are described in the implementation section.

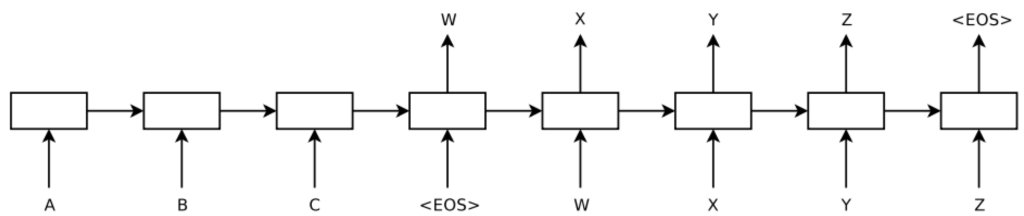


Figure 2: Example of an encoder-decoder architecture from [19]. The model reads the input sentence "ABC" and produces "WXYZ" as the output sentence. The model keeps generating a response until it outputs the end-of-sentence token denoted as <EOS>.

2.2 Ontologies

Ontologies are defined as semantic networks that contain relationally and hierarchically connected concepts [20]. The interconnection between the entities is presented as a graph, allowing applications to search synonyms, related entities, or hierarchical relations by keywords. Ontologies became a point of interest in computer science as a way to bring a deeper understanding of word relations to NLP programs. One of the first applications to do so was WordNet, a large domain-independent ontology that describes linguistic concepts as sets of synonyms [21]. It has been applied to many NLP subproblems, including summarization [22], translation of texts [23] and building smaller domain-specific ontologies [24]. Applying these domain-specific ontologies to increase the domain knowledge of an NLP system by providing information about the relationships between the entities in the domain has been an active topic of research [25]. One that is closely related to the use case of domains in this thesis.

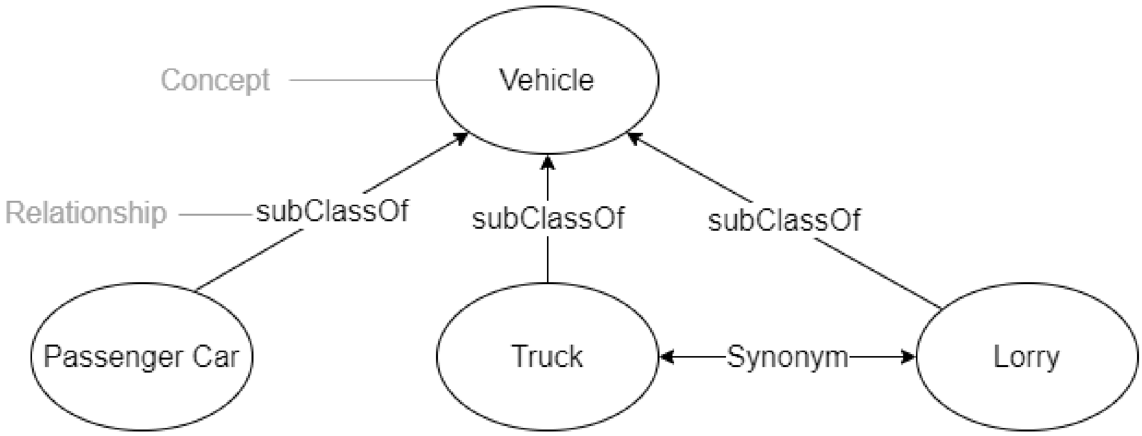


Figure 3: Example ontology that defines the relationships between vehicle entities.

Figure 3 illustrates a simplified domain ontology. The entities themselves usually do not contain much information, but the relationships between entities can be used to model very specific attributes. Relationships between entities can either be directional (subClassOf relations in figure 3) or bidirectional (synonym relation in figure 3). The types of relationships present in an ontology depend on the modeling languages used.

Chatbots built using ontologies have been experimented with as a way to bypass the strict requirements that AIML or other markup languages pose on a chatbot [26]. As machine learning models have grown dominant in the field such implementations have lost their competitive edge. Ontologies have found a new application recently as they have been introduced into chatbot systems as a tool for improving the system's domain knowledge [27]. They have proven useful in chit-chat scenarios but little research has been conducted on using them with goal-oriented chatbots, especially when the content of the follow-up question needs to be kept in memory for future use to determine the appropriate action the user wants to take.

Ontologies will serve a critical purpose in the implementation of this thesis as they provide easy access to relevant terminology surrounding the misinterpreted entities. Queries built using relevant relationships can be used to explore which of the nearby entities might be better suited to be used in the original query. The next subsection describes in more detail how ontologies are built and what kind of languages are used to describe the relationships between the entities.

2.2.1 Ontology design principles

Building new ontologies have largely been driven by a need for extended domain knowledge. This has lead to a situation, where a decent amount of ontologies exist but most of them are crafted for very specific use and reusability in the design process has been neglected. So-called core ontologies have been built, which are designed to contain only the bare minimum knowledge from their given domain, increasing their reusability and allowing more sophisticated use cases by combining multiple small ontologies [28]. But even these core ontologies often suffer from extensive terminology and developers find themselves stuck with the knowledge that is not relevant to their use case. This has caused the need for the development of Content Ontology Design Patterns (CP), small ontologies that are usable as building blocks for larger use cases [29]. The relevant CPs are combined by adding the relevant dependencies between them and then used as a single larger ontology.

A defining quality of ontologies is the types of relationships it contains. Whereas CPs are ontologies in themselves, Ontology Design Patterns (OP) are a modeling solution to solve a recurrent ontology design problem. These are for example structural, correspondence or lexico-syntactic OPs [28]. These design patterns define the relationships between the entities in the graph, allowing different kinds of relationships in the data to be presented. In the case of this thesis, structural and correspondent relationships are important, since we want to understand the related terms and possible synonyms for misinterpreted entities.

2.3 Rasa Framework

Rasa along with other chatbot frameworks offers easy configuration of topics and tasks that the users can engage in. Using Natural Language Understanding (NLU), the systems can recognize entities and user's intents from their messages and connect these to favorable actions. Rasa and its peers ease the workload of chatbots developers to a great extend by providing a well-established basis for both NLU and NLG tasks that can be modified if needed, but also allows the developer to focus on the domain-specific tasks as the baseline implementations are already quite robust.

Figure 4 gives an overview of the architecture of Rasa. The user's message is fed into the system as an input and it goes through various modules as a reply is generated. The numbers in the figure correspond with the following steps detailed in Rasa's documentation [30]:

1. The message is received and passed to an Interpreter, which converts it into a dictionary including the original text, the intent, and any entities that were found. This part is handled by NLU.

2. The Tracker keeps track of the conversation state: previous actions taken and intents recognized.

3. The policy receives the current state of the tracker.

4. The policy chooses which action to take next.

5. The chosen action is logged by the tracker.

6. A response is sent to the user.

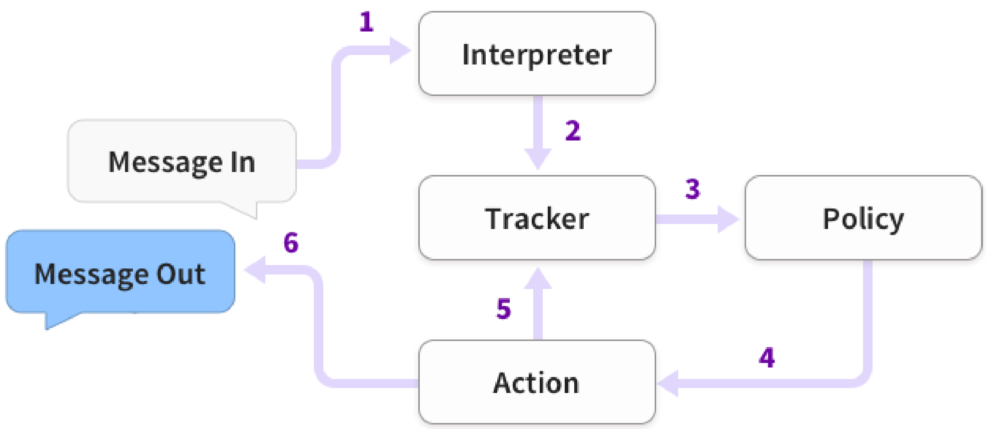


Figure 4: A diagram of Rasa's architecture [30]. The user's message goes through the modules presented here as a fitting reply is generated. This overview matches with the version of Rasa used in the baseline implementation of this thesis. For other versions of the Rasa framework, the architecture might look a bit different but the same tasks are executed.

Rasa is configured into a specific domain by defining the entities, intents, and actions inside the chatbot's configuration files. Rasa NLU model is trained to understand different questions and extract the necessary information with a set of example messages that are grouped under the corresponding actions and intents. These example conversations used in training are called stories. When a user's intent is recognized, the current state of the conversation is sent to the designated action. A common way to build Rasa chatbots is to keep the action server as a separate process from the main chatbot application. The main application sends the current state of the conversation to the action server, where the action executes and returns the output message to the main application, where it will be returned to the user along with being stored in the Tracker. The actions are highly customizable, allowing any kind of custom code to run and generate the responses. Actions can also directly manipulate the Tracker, erasing or adding to the conversation history. Reverting users' messages is especially useful in the case of low-confidence intent recognition and misinterpretations, as messages that steer the conversation in the wrong direction can be removed from the conversation history, helping the chatbot to focus on the actual intent of the user.

By default chatbots developed with Rasa are currently unforgiving in the sense that given an unclear input, the system defaults to a state such as "sorry I did not understand". Rasa offers the tools to build more complex response systems for these scenarios but so far those are not widely adopted and not publicly available. Next, the knowledge base functionalities of Rasa are introduced, which play a part in solving this problem.

2.3.1 Knowledge bases

Knowledge bases can be integrated into the Rasa framework to improve their contextual understanding of the current conversation, or as a way to inject a deeper understanding of objects [31]. Contextual understanding of the current conversation here refers to the linguistic tricks one can use in a conversation to refer to objects not by their real names but by terms such as "it" or "the second one". This information can be hard to keep track of for traditional models. In Rasa, knowledge bases offer a way to keep track of user references to different entities.

Arguably the main function of knowledge bases in Rasa is to give the model access to a deeper understanding of objects in the conversation. Knowledge base integrations can be done to one or more databases or other data structures and custom actions to return the relevant knowledge from each can be created. Revisiting the hotel example, this can mean having access to the current booking status of a hotel, allowing the model to discuss the current booking status and describe the amounts of available rooms in a natural way. Knowledge bases are also a natural way to store room-specific information since the data can be kept in a separate database table and updates made to the data are instantly accessible to the dialogue system.

Knowledge base integrations can be created for ontologies as well by building a custom knowledge base action. One or more ontologies can be presented as a graph database. This database is then integrated into Rasa by building a custom action that can query the ontology database, using the graph structure of ontologies to return the desired values. The implementation for this thesis will utilize this feature to integrate ontologies into the Rasa framework.

3 Methods

This section describes the implementation built for this thesis. First, some background on the project's baseline implementation and performance is provided. Then the extensions to that application are introduced, that try to improve the question clarification performance of the baseline model. Lastly, before moving on to the results section, the methods used to evaluate the results are described.

3.1 Baseline implementation

The baseline implementation is a chatbot system built using the Rasa framework introduced in the background section. The goal of this system is to provide answers and guidance to the user's technical questions about Ubuntu operating systems by either providing a direct answer or by linking the relevant parts of documentation to the user. It does this by recognizing relevant entities from the user's messages and returning responses utilizing a set of machine learning models and a database of Ubuntu's technical documentation, bug reports, and forum conversations. The Rasa framework was chosen for the implementation because it provides easy deployment of its default features while also making it possible to extend its functionality. The dataset used will be introduced in more detail in the next subsection.

In more detail, the Rasa NLU module receives the user's input and uses the model trained on example messages to determine the user's intent. These example messages are partly generated from the source data. After Rasa has predicted the intent of the user, it calls the corresponding action. If the user's question is interpreted correctly, the action extracts keywords from the input message and uses those to query an Elasticsearch instance for answers to the problem in question. The source data stored in Elasticsearch indices are introduced in the next Dataset subsection.

This implementation utilizes the default functionality of Rasa for handling misinterpretations and uncertain inputs. There are two separate thresholds configurable that determine the appropriate response: "fallback_threshold" and "nlu_threshold". If a user's intent can not be distinguished with confidence above the thresholds configured in Rasa, it always defaults to asking for the user to rephrase their question: "sorry I didn't get that. Can you rephrase?" This threshold in Rasa is the "fallback_threshold". In some cases where the confidence is over the "fallback_threshold" but not over "nlu_threshold", Rasa will ask for affirmation: "Did you mean x?" The user's positive response to this will return the answer as it would have with a high confidence and a negative response will lead to similar actions as falling below the "fallback_threshold".

To support the Rasa chatbot, a custom user interface was built. The need for a custom user interface was caused by other aspects related to the further development of the project. The interface provided users with a simple text box for sending new messages as well as a view for seeing the conversation history. The sent messages are logged into an Elasticsearch index for later viewing and analysis.

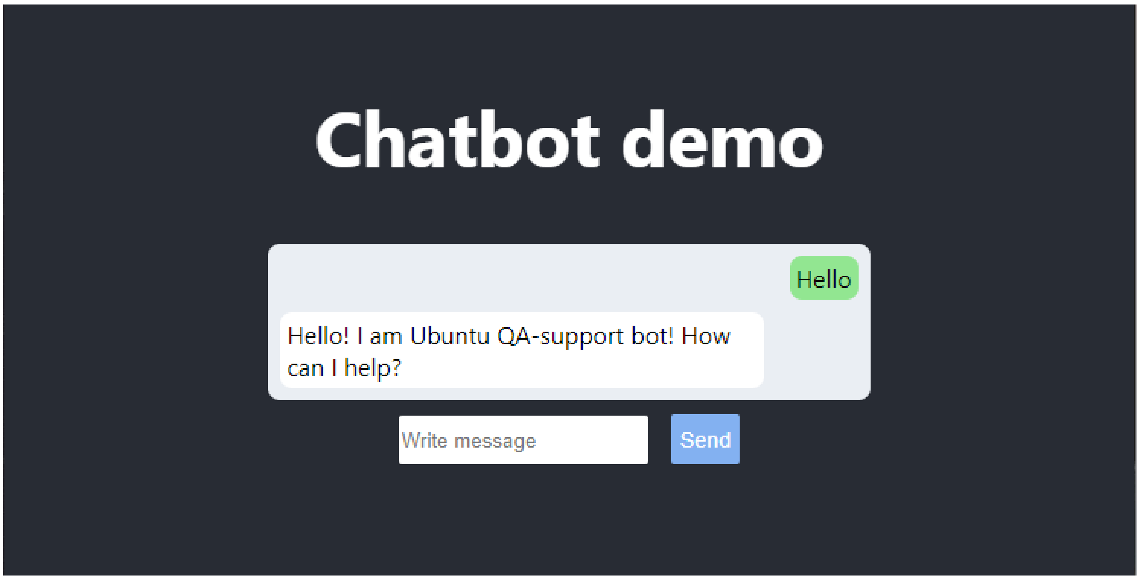


Figure 5: The user interface developed for the chatbot.

3.2 Dataset

The baseline implementation uses a database consisting of Ubuntu's official documentation, Ubuntu launchpad forum conversations, and Ubuntu community help wiki pages. The documents were gathered by scraping the sites with Ubuntu's permission and anonymizing the documents. The reason for scraping the pages instead of opting for a ready-made dataset such as the Ubuntu Dialogue Corpus [32] is that this way both the official documentation and multi-turn forum conversations could be included in the same dataset while having the same format and other features of the baseline implementation (that are not related to this thesis). Some of the features done in the project before the implementation of this thesis require this format to function properly and changing the dataset was thus not meaningful.

The scraped data contains the title of the page or question, raw text of the original document, a formatted version used for training, keywords for the entity that were recognized from the formatted conversation, possible related documents, links, and a status field that describes if the forum conversation got solved, answered but not marked as solved, was left open or was missing information.

To train the Rasa NLU model to the Ubuntu domain, the entities present in the dataset need to be configured into Rasa stories. These stories link the user's input into the correct response and action in the action server. Because of the large size of the dataset, the files containing these stories are generated computationally. These stories contain example phrases that the user might ask concerning the entity in question. The phrases are generated using templates, which allow fast and easy production of the questions but sacrifice flexibility. The Rasa NLU module trained on this data expects simple questions that follow a rigid structure taught to it by these examples. Some additions to the training data have been made by hand but the majority of the NLU training data has been generated in the aforementioned way.

3.3 User intent clarification module

To improve the performance of low-confidence cases, an experimental fallback module is fitted into the baseline implementation. This module is activated whenever the chatbot can not interpret the user's intentions with good enough confidence and neither the "nlu_threshold" nor "fallback_threshold" is met.

The new module utilizes a configurable set of ontologies to find relationships between the user input and other relevant entities that the user might have meant to query for. This data is used to provide the user with suggestions for related query terms. Instead of simple yes or no type questions with often unrelated suggestions and "please rephrase" type feedback, the goal of this implementation is to use factual data provided by the ontology to recommend more relevant results to the user. The system checks the responses linked to the entities retrieved from ontologies and validates them before showing them to the user, thus preventing empty responses and enabling more fitting answers if no related entities are found. This way the system can make sure that the new recommendations return meaningful responses instead of just resulting in more follow-up questions. Although generating more follow-up questions can be a useful outcome in itself, the generated questions should further define the context and help in defining the original intent of the user, not confuse them more.

Rasa's existing functionality allows the extraction of the user's input inside the fallback module. The message is parsed and preprocessed, after which the words deemed relevant to the given context of Ubuntu are used to query the ontologies for synonyms or parent terms. Words that have a length of fewer than three characters are omitted from the ontology queries, as they are very often prepositions, pronouns, or other words that are not by themselves related to the entities the user meant to query for. These short words often return many results from the ontologies with the queries used in the implementation, but the results are not relevant to the given context. Another preprocessing step for the ontology queries is omitting all special characters and numbers from the query, as these will cause otherwise correct results to be missing from the result.

The ontologies are integrated to Rasa with GraphDB. The next subsection introduces the ontology integration as well as the used ontologies and queries in more detail. After the ontology is queried for related terms, the fallback function takes the resulting list of terms and performs a query into the Elasticsearch instance for each of the result words. At most three of the most fitting results are returned to the user, this restriction is needed because more results than this will produce too much text for the user, significantly harming the user experience. The results to be shown are also filtered by keeping in memory a list of used keywords, that are returned from

Elasticsearch along with the actual results. The list of used keywords is checked every time a new answer is found in order to prevent the same answer from being shown to the user twice, since querying for a specific keyword alone always returns the same answer from Elasticsearch. After all of the keywords have been iterated through, the results of these queries are formatted into messages, with each message containing the result of one query. The first three messages are shown to the user along with a message that contains the keyword that returned the answer in question as a suggestion, e.g. "Did you mean to ask about virtual environment?" when the user asks about virtual spaces.

If there are no results from the ontology query, the system defaults back to its original functionality: the user's message is erased from the conversation history and the chatbot asks for the user to rephrase their question. During the development of this module, some alternative functionalities were tested so that the system could have advanced the conversation even if the ontology did not provide any useful results. This proved to be problematic since without any added domain knowledge the conversation history is the only data available. Being able to generate better messages with only this information essentially means building a better performing chatbot inside the fallback function than the main system, which is both very hard and not the problem this implementation aims to solve. Although using some other forms of domain knowledge here would be interesting to see, this thesis focuses on exploring the possibilities of using ontologies to provide this contextual information.

This subsection provided an overview of the functions of the question generation module developed for this thesis. The next subsection will introduce the ontology integration in more detail and provide more background on how the ontologies are queried for relevant information in this implementation.

3.4 Used ontologies

The used ontologies are hosted in a GraphDB repository. GraphDB is the technology of choice because of its comprehensive integrations, support of SPARQL queries, and the possibility of commercial usage for the free license solution. SPARQL stands for SPARQL Protocol and RDF Query Language and it is an semantic query language for databases that is able to retrieve and manipulate data stored in Resource Description Framework (RDF) format [33]. GraphDB allows the simultaneous usage of many different ontologies, making it relatively simple to swap and append ontologies or subsections of them.

The main ontology of this implementation is the Computer Science Ontology (CSO) [34]. It is a large-scale, automatically generated ontology. It was generated using the Klink-2 algorithm [35] on the Rexplore dataset [36], which consists of about 16 million publications in the field of Computer Science. The Klink-2 algorithm is a combination of machine learning, semantic technologies, and external knowledge to automatically generate a fully populated ontology of research areas, although some entities were manually reviewed by experts. The main root of CSO is Computer

Science, however, the ontology includes also a few secondary roots, such as Linguistics, Geometry, Semantics, and so on.

CSO is used here because of its two advantages over manually crafted ontologies previously used to describe Computer Science. Being automatically generated, it can describe research areas by describing hundreds of sub-topics and relevant terms. Comprehensive mapping of topics and terms allows even more specific terms to be linked to their respective research fields. The second advantage over manually crafted ontologies is the possibility to easily update the ontology by running Klink-2 on a set of new publications of the Computer Science field. Keeping the ontology up-to-date with the current state of the art research this way requires significantly less manual labor.

A downside of CSO for our specific use case is that while it details the field of Computer Science in great detail, information about Ubuntu-specific entities is not present. Since the main use case of the chatbot is to provide technical support for Ubuntu users, many of the keywords used require understanding the Ubuntu-specific context, which is not present in CSO. Although appending information to the ontology is possible, it was out of the scope of this thesis. Running the Klink-2 algorithm and updating the ontology requires a lot of computational power and such resources were not available during development.

CSO details eight different types of relationships between the entities in the ontology. These are listed and described below as they are represented in CSO's documentation.

- **owl:sameAs**: lists entities from other knowledge graphs from the Linked Open Data Cloud (DBpedia, Freebase, Wikidata, YAGO, and Cyc) that refer to the same concept.
- **relatedEquivalent**: two topics can be treated as equivalent for the purpose of exploring research data, most often details the same term with different word separator characters or the same words plural form.
- **superTopicOf**: indicates that a topic is a sub-area of another topic.
- **contributesTo**: indicates that the research outputs of one topic contribute to another.
- **preferentialEquivalent**: this relation is used to state the main label for topics belonging to a cluster of relatedEquivalent. For instance, the topics ontology and ontologies will both have their preferentialEquivalent set to ontology.
- **rdf:type**: this relation is used to state that a resource is an instance of a class. For example, a resource in our ontology is an instance of a topic.
- **rdfs:label**: used to provide a human-readable version of a resource's name.

- **schema:relatedLink**: links CSO concepts to related web pages that either describe the research topics (Wikipedia articles) or provide additional information about the research domains (Microsoft Academic).

For the use case of this thesis, the most useful ones detail synonyms for the same term and possible child-parent relations of terms. We are interested in synonyms because they offer direct access to other wording that might be used in the source material to detail the entity the user is searching for. Synonyms in CSO are defined by the "SameAs" attribute of the OWL web ontology language and the "relatedEquivalent" attribute. Out of these two, the "SameAs" attribute proved to be more useful in our case, since it fetches knowledge from other sources that are more likely to detail a completely different wording for the term than the terms available through querying the "relatedEquivalent" attribute. The terms related to one another by the "relatedEquivalent" attribute are most often different conjugations of the same word, and thus do not provide any meaningful information for us to use to better the original question. Child-parent relationships are examined since the hypothesis is that providing a broader term to query for than the original term, meaning its parent, is likely to return some related pages from the source material that the user was looking for. These relationships are defined by the "superTopicOf" attribute of CSO. The problem with this attribute is that the limit for a topic to be considered as a sub-area of another topic is set quite low. This attribute most often returns many results, which are useful to a varying degree. Using this attribute in the custom module thus proves to be problematic, since filtering the results for the most related terms requires additional knowledge of the semantic structure of the domain. Although it is easy to generate recommendations to the user using this term, the results are too far off to be useful and thus this attribute was not used in the final implementation of this thesis. The "contributesTo" attribute suffers from the same problem as the "superTopicOf" attribute, there are too many results that only loosely relate to the user's input, if at all.

The rest of the attributes in CSO suffered from the opposite problem, they return results that do not help in understanding the domain further. They only provide extra knowledge of the initial term used in the query and how that term is described in the ontology. Although with some other approaches utilizing a more complex ontology query mechanism these attributes could be useful, in the scope of this thesis using those was not meaningful since the key function of the ontology module here is to return related terms. The "sameAs" relation captures these relations well and thus it was chosen as the main semantic relation to use.

As an example of how these attributes are used below is the template used to search the ontology for synonyms utilizing the "sameAs" attribute. In the query, the keyword being searched for is represented by the "query_term". This term can be any kind of string that is extracted from the user's question.

```
PREFIX cso: <http://cso.kmi.open.ac.uk/schema/cso#>
```

```
SELECT ?s ?o
WHERE { ?s cso:sameAs ?o .FILTER regex(str(?s),query_term) .}}
```

The query above returns all entities from the ontology repository that are classified as synonyms of the query term, or a term that contains the string representation of the query term as a substring. The substring is checked to gain more flexibility in the search since exact string matches were rare during development. After the synonym entities have been returned from the ontology, additional processing is done to format the result strings into more suitable forms for both viewing and to serve as better inputs for querying the source data. If the source data contains items related to the ontology result, the best options are returned to the user using the mechanism described in the previous subsection.

3.5 Methods of evaluation

User evaluation is used as the main method of assessing the model's performance. Because of the nature of this work, user evaluation plays a significant role in determining the performance of the model: no automatic metrics are able to capture the situation of not understanding the initial message to then improving the "quality" of the conversation by offering clarifications. The literature around chatbot evaluation also notes that evaluation of such systems should be adapted to the application and user needs [37]. An evaluation methodology should not be adopted just because a standard has been established, such as the Loebner Prize evaluation methodology adopted by most chatbot developers. Instead, the implementations should be evaluated based on whether they achieve their specified services or tasks and for this, no general automatic metrics currently exist.

The user evaluation of the implementation is performed by having a test group first solve different use cases on the baseline model. These use cases can be any technical issues they have had with Ubuntu themselves, or if they are not experienced Ubuntu users, any topic they can find from the Ubuntu forums. In the test scenario, another forum is referred to the testers than the one used to gather the dataset used to train the model. The chatbot is made available to the users by publishing it to a test server used during development. The simple user interface developed as a part of the project was used in the tests. This interface consists of a simple text field for inputting new messages as well as a view for the messages sent back and forth so far. This user interface is the same as seen in figure 5. After they have tried solving a couple of different problems using the chatbot, they move on to answering a questionnaire about their experience with the use case. The full questionnaire can be found from the appendix of this thesis. It seeks to quantify information about the performance, relevancy, and grammatical correctness of the model. Special attention is directed to the responses of the model in cases where it can't find a correct answer such as the usefulness and relevancy of the suggestions. The users evaluate each of the qualities on a scale from "strongly disagree" to "strongly agree", and are asked to provide brief descriptions of why they chose their answers on an open text field. This kind of scale from "strongly disagree" to "strongly agree" is used because it offers a decent amount of flexibility compared to a "yes or no" question, while not being too open-ended in its options. [38].

It should be noted that although surveys might measure task success well, they may not explain why the tester's fail to use the system being tested or what caused the task failure [39]. It can also be possible to complete the task successfully but through several mistakes or confusions, that are not apparent by examining the survey results. To prevent this the questionnaire contains free text fields that the testers are encouraged to write a more extensive review of what they experienced during the tests. This provides a lot more material for more extensive reviews of how the system performed during the tests as well as some support to the scale questions and information about the testers' reasoning behind their answers. The functionality being tested here is dependent on the testers failing to get the correct answers with their initial input, which makes the design of the survey a significant factor in how the results turn out. Designing the questions to target the error cases without being all about them is achieved by dividing the test into two sections, the first being a broader overview of the chatbot's functionality and the second part targeting the custom fallback module specifically.

After the performance of the baseline model has been assessed, the test group is asked to perform the same tasks and answer the same questionnaire again, this time with the baseline model being extended with the implementation of this thesis. After this phase, the results are ready to be reviewed. The hypothesis is that the questions that were correct in the first phase, should also be correct in the second phase. But for the questions that were wrong or misunderstood in the first phase, the system should now be able to recommend some related content instead of only prompting the user to rephrase their question. The results and analysis on them can be found in the next section, along with some example cases that seek to clarify how the model performs and how it could be further improved.

4 Results

This section contains the results of using the evaluation methods described in the previous section. After introducing the overall results, the most interesting aspects are looked at in more detail. Additional proof for the causes behind the results is provided by estimating the ontology coverage and providing example cases.

4.1 User evaluation

The test group consisted of people who work in Atostek in various tasks. Roughly half of the 20 people involved in testing have substantial knowledge of Ubuntu as either a user or as a developer. The other half of the test group had little to no experience with Ubuntu, and they used a referred online forum as an inspiration for their conversations with the system. The referred forum was different from the one used as a part of the training data for the Rasa model.

The first phase of the testing consisted of evaluating the baseline performance for the system prior to introducing the custom fallback module. In the second phase, the testers evaluated the performance of the implementation introduced in this thesis. The first row of the Table 1 shows the results of evaluating the baseline model. The second row contains the corresponding results for the second phase (implementation of this thesis). Table 2 contains the results of the fallback performance for both phases in a similar manner. The users were asked to rate the relevancy and grammatical correctness of the answers they received on a scale from 1 to 5 taking into consideration all of the answers they received in their answers. The tables below show the average of the answers the testers provided.

Test Phase	Relevancy	Grammatical Correctness	Asked for affirmation
1	3.35	3.74	70%
2	2.95	4.05	58%

Table 1: Results for the baseline implementation are presented in the first row of the table. The second row contains the results for the implementation of this thesis. The answers scale is from 1 to 5 except for "Asked for affirmation", which is in percents.

Test Phase	Relevancy	Usefulness	Grammatical Correctness
1	2.29	1.86	3.74
2	2.00	2.38	4.05

Table 2: Results for clarifying user intent when the initial question is not understood. The answer scale is from 1 to 5.

The differences between the results shown in Table 1 are quite small and the causes for the changes can be caused by multiple different variables. Based on the results visible in the two tests presented, it is not trivial to say that the implemented fallback module can better the initial implementation at all. With a more detailed analysis of the individual answers, some clarity on why the results look as they do can be obtained. Before that, let us take a closer look at the numerical values and explore their implications about the model performance.

As can be seen from Table 1, the relevancy of the answers dropped by 0.4. This could be explained by the higher threshold values used in phase 2, which caused some of the correct answers to be left out and replaced by the ontology queries, which in turn did not succeed in returning answers as good as the Rasa NLU. By looking at the text answers the testers provided, most of the ontology-generated answers returned in phase two were not directly related to the user's question, which directly affects the relevancy of the answers. Another point of interest in comparing these two tables is that the percentage of times when the system asked for affirmation fell from 70% to 58%. This is partly due to the same change in thresholds as the change in the relevancy column. Another cause for this might be that by looking at some individual answers, some testers deemed results from the ontology module as the answers returned by Rasa itself, not as something for which the system asked affirmation. This is a problem in the way the question was presented, which caused some people to misinterpret the question. This caused the number of answers for the second part of the questionnaire (the part about the misunderstood questions) to be lower for phase two than for the baseline tests. While the result itself is positive since the number of times the system asked for affirmation was lower in phase two, in this test setting a more meaningful result would have been for the number to stay the same since the implementation is trying to improve the functionality of the program beyond that point of the user being asked for affirmation. Additionally, the underlying NLU model did not change in between phases, which means that the drop in the percentage for affirmation asking can not be due to the model being better. The only changes here are the higher threshold values and the introduction of an updated fallback module, which only affects the performance of the system beyond the point of asking for affirmation.

Table 2 contains the results for questions about the fallback module for phases one and two. An interesting outcome of the tests is that while relevancy of the fallback answers ("did you mean x?" type answers in case of misinterpretations of the original question) dropped from 2.29 in phase one to 2.00 in phase two (difference of -0.29), the usefulness of the answers rose from 1.86 in phase one to 2.38 in phase two (difference of 0.52). This can indicate that while the baseline implementation was unable to recommend any kind of related terms and questions, the "I didn't understand, could you please rephrase your question?" type replies were deemed more relevant by the testers than the more sophisticated but unrelated answers retrieved from ontology terms. The ontology terms however, were deemed more useful since they provided at least some suggestions for the user instead of only encouraging them to ask another question. But again, while looking at these results it should be kept in mind that because of the number of testers that answered the part of the questionnaire about affirmation was smaller in phase two than in phase one, the results are less reliable and uncertainty in the results is increased. This issue is visible in the last column of table 2, where the grammatical correctness of the answers increased from 3.74 in phase one to 4.05 in phase two with no changes to the NLG part of the software. While this may also be caused by better grammar in the answers that changed, a more likely cause for a change this large would seem to be the fact that some answers from phase one were not present in phase two.

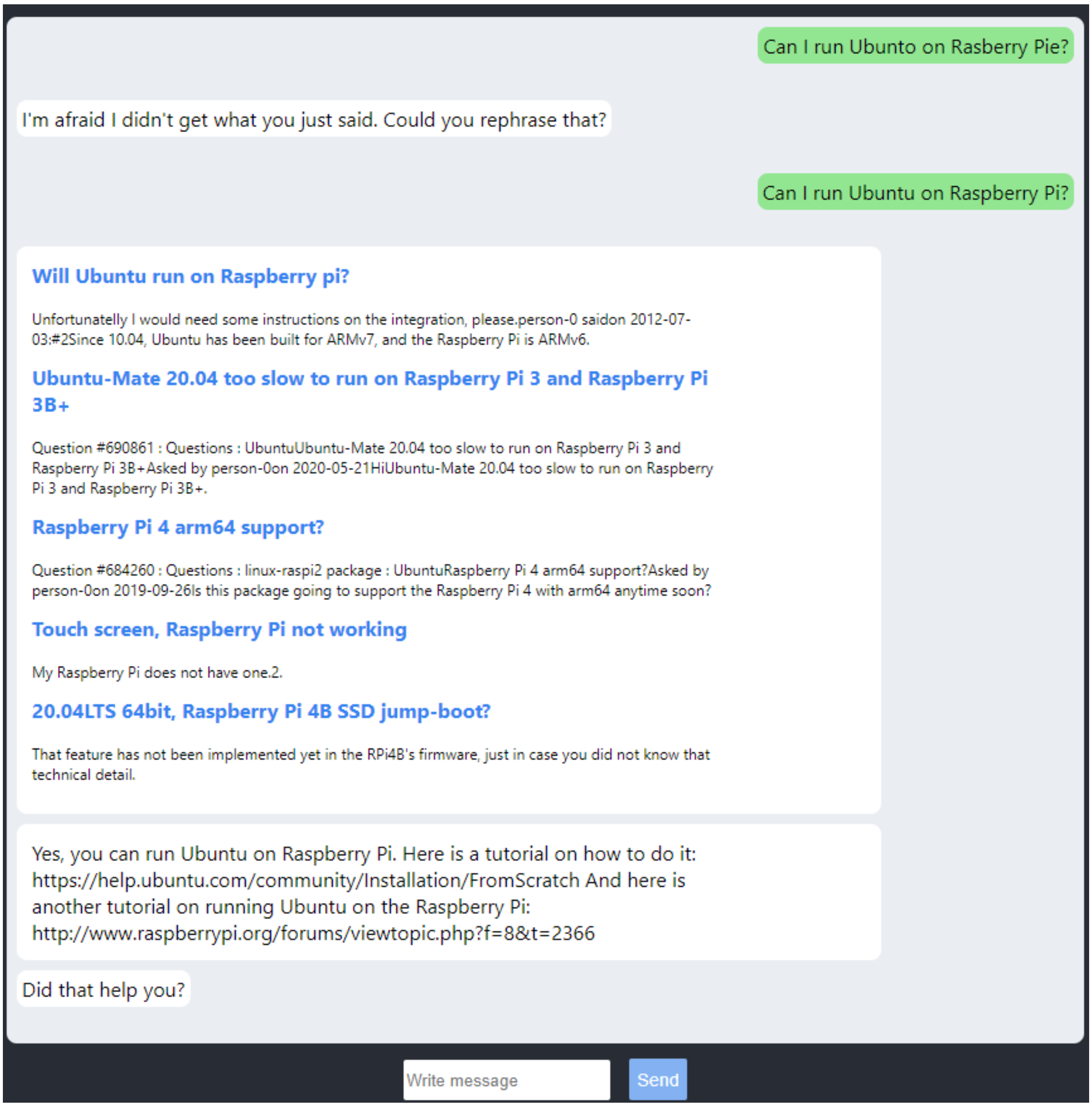


Figure 6: Example of how the chatbot does not understand the user's input if it contains small grammatical errors. The chatbot is unable to infer the correct entities if they are not absolutely correctly written in the input message.

The first phase of testing revealed some interesting qualities of the baseline implementation. As reported by the testers in the open text fields of the questionnaire, a significant part of what made the software less usable was caused by how the NLU module handled grammatical errors in the user's input. For example, when the chatbot was asked "Can I run Ubunto on Rasberry Pie" the chatbot is unable to find the correct meaning of the sentence and defaults to the fallback functionality. When the input is changed to "Can I run Ubuntu on Raspberry Pi" the system is able to give a correct answer. With changing two letters from the input string the system is able to start the correct action and correctly recognize the sentence. This issue is demonstrated in figure 6.

The reason for the bad handling of grammatical errors has to do with how the model is trained. This drawback could likely be solved with more data and some noise in the form of grammatical anomalies being introduced into the training process of the model, which would allow the NLU module to have a broader understanding of the target language and thus be able to link misspelled words to their correct counterparts.

A common outcome of introducing the ontology module to the testers was that more often than before the chatbot was able to recommend some articles it deemed related based on the ontology data. These recommendations, however, were not often linked to the actual topic the user was interested in. As can be seen from the results in Table 1, the relevancy of the recommendations in phase two is smaller than in phase one. It should also be kept in mind that in phase one the chatbot lacked the functionality to make recommendations and instead used only the default implementation of a fallback function provided by Rasa. Although the relevancy is higher in phase one, in reality, the chatbot was not able to generate recommendations, which interestingly seems to be deemed more relevant to the context than altogether unrelated recommendations that were generated in phase 2.

As the results of the tests do not provide clear results, a more in-depth analysis of individual answers is in place. To understand the numerical values better, the text answers were analyzed in detail in order to gain additional metrics to the performance. Looking deeper at the individual responses the chatbot produced, the number of times it produced a coherent recommendation for a related search term was 36.8 % higher in phase two than it was in phase one. This relationship was discovered by looking at each answer and what kind of recommendations they got from the chatbot. Furthermore, these recommendations were deemed useful 25.0 % of the time in phase two, as opposed to 7.1 % in phase one. These numbers indicate that the module implemented was able to outperform the default implementation in terms of producing search term recommendations. This signals that the underlying implementation seems to perform as expected, but the data available to the custom fallback module might not support the terminology used. This seems to be the largest issue with the current implementation of the module and the tests implemented: The terms present in ontology do not fully capture the domain, leaving many of the recommendations without a result or with results unrelated to the current topic. These issues are further explored in the next subsection.

4.2 Ontology coverage

The root cause for many of the irrelevant answers generated by the ontology module seems to be the lack of domain-specific terminology present in the ontology. To prove this is a relevant factor, the individual answers that activated the fallback module but that did not get a relevant reply from the chatbot are further explored. This is done because if the custom fallback module was activated, we can be sure that the messages it generates are directly related to the words in the original question sent by the user. If the original question does contain terminology relevant to Ubuntu and Linux, we can prove that meaningful answers would be generated if the ontology would contain relevant entries for the corresponding entities. The next chapters will introduce some individual questions that the testers asked the chatbot and that did not get a relevant response. If a correct answer can be obtained by changing the wording or the query terms sent to the Elasticsearch index, we can be sure that a correct answer would have been obtained if the relevant entities existed in the ontology, thus proving that the coverage of the used ontology is a part of the drawbacks of the implementation.

4.2.1 Case 1: task manager

The first situation during the tests where the system fails because of the ontology coverage was when the system got asked about opening the task manager. During the baseline test, the system was altogether unable to answer this question. Figure 7 shows how to conversation plays out with the custom module in place. The ontology is queried for related terms, but the only linked entity that the ontology finds is "linked data", which provides quite vague results as can be seen from the figure.

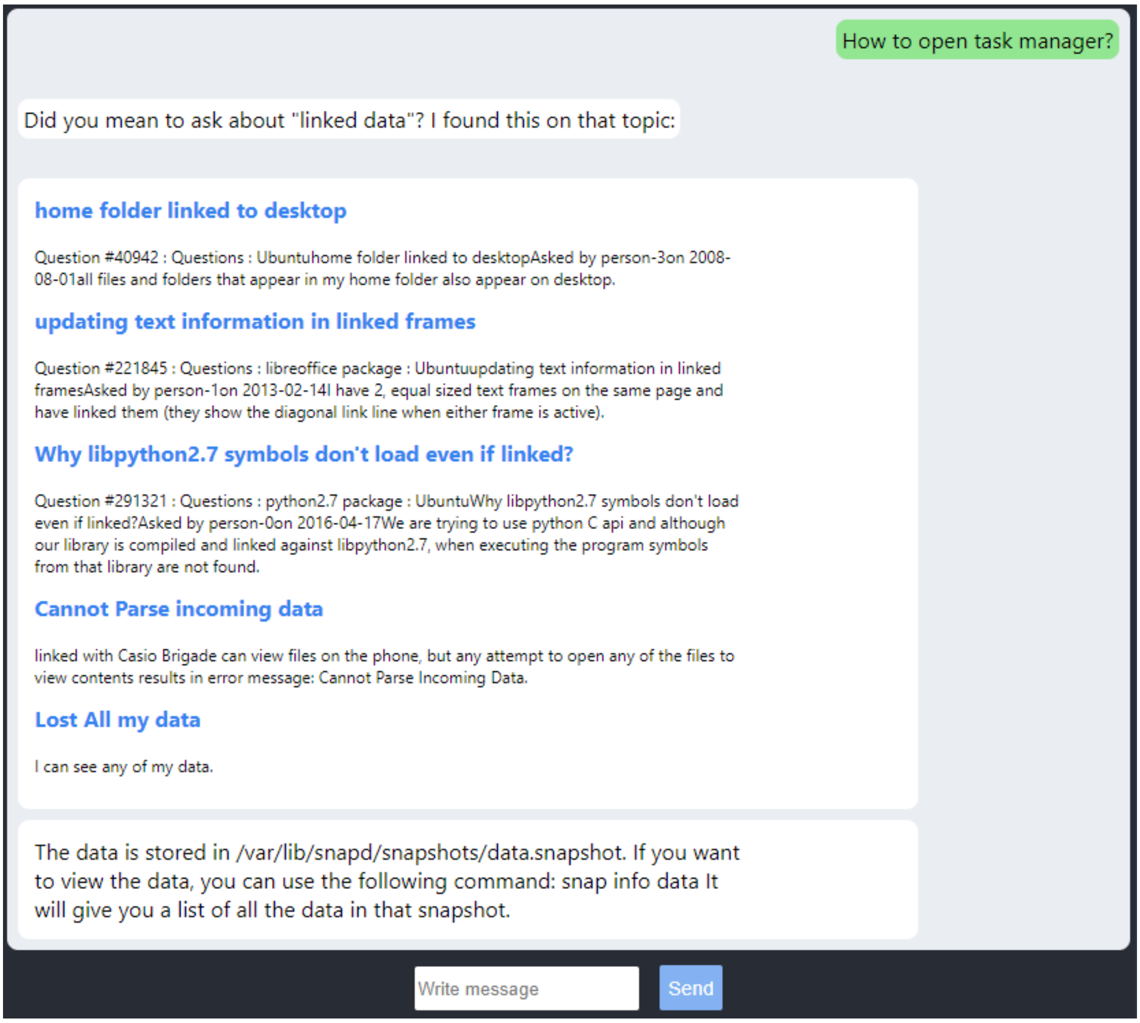


Figure 7: When the chatbot fitted with the ontology module is asked about the task manager, it can't find the correct answers and instead shows only unrelated articles about "linked data" entity found from the ontology.

The trained Rasa model is unable to decipher the meaning behind task manager. Although the term is widely used in everyday conversations, the term is not directly represented in Ubuntu and instead comes from the Windows environment. When talking about Ubuntu, user's often reference the system monitor as task manager, as it provides the equivalent functionality. On a more technical level, the system monitor is executed by running the gnome-system-monitor command. If we now replace task manager with this technical term for an Ubuntu task manager, gnome-system-monitor, we get the correct results. This dialogue is shown in figure 8. If we would now extend the ontology inside the custom fallback module to contain the task manager and gnome-system-monitor entities and add a reference between those two, the system would be able to understand the user's original question without the need for knowing the technical details of Ubuntu to be able to formalize the correct question.

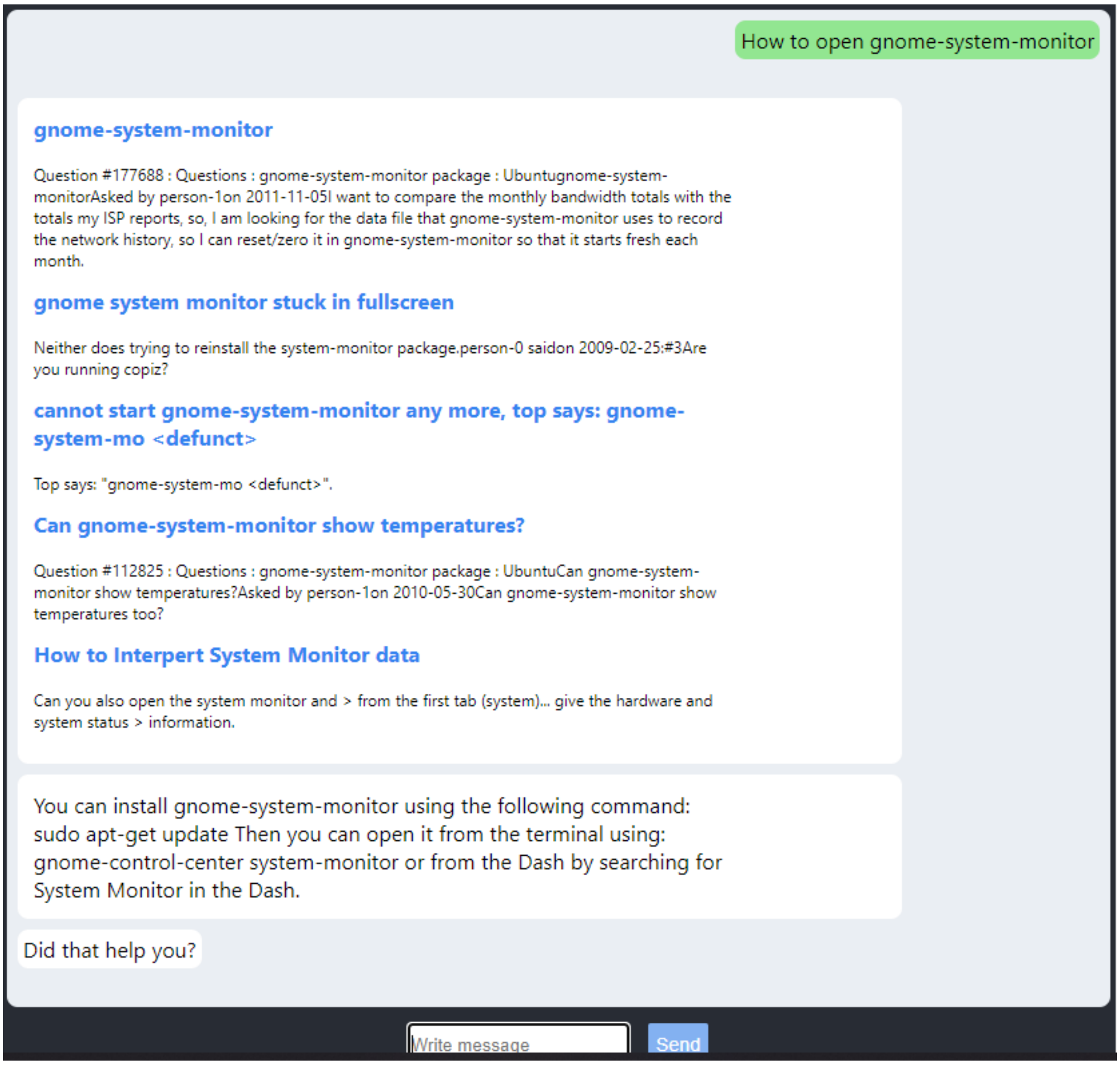


Figure 8: If in the original question "task manager" is replaced with a domain specific term the system is able to return the correct result. This same result would have been achieved with the ontology containing a reference between task manager and gnome-system-monitor.

4.2.2 Case 2: Personal Package Archive

A similar behavior to the task manager case is displayed if the system is asked about Personal Package Archives by referring to them with the acronym PPA. Figure 9 demonstrates how the system is unable to infer the meaning of the acronym from the first question, but as the term is expanded to its full form of personal package archives, the system is able to produce a coherent answer. Like the previous case, this could also be solved in the current setting by an ontology that would have a link between the acronym term and the full term.

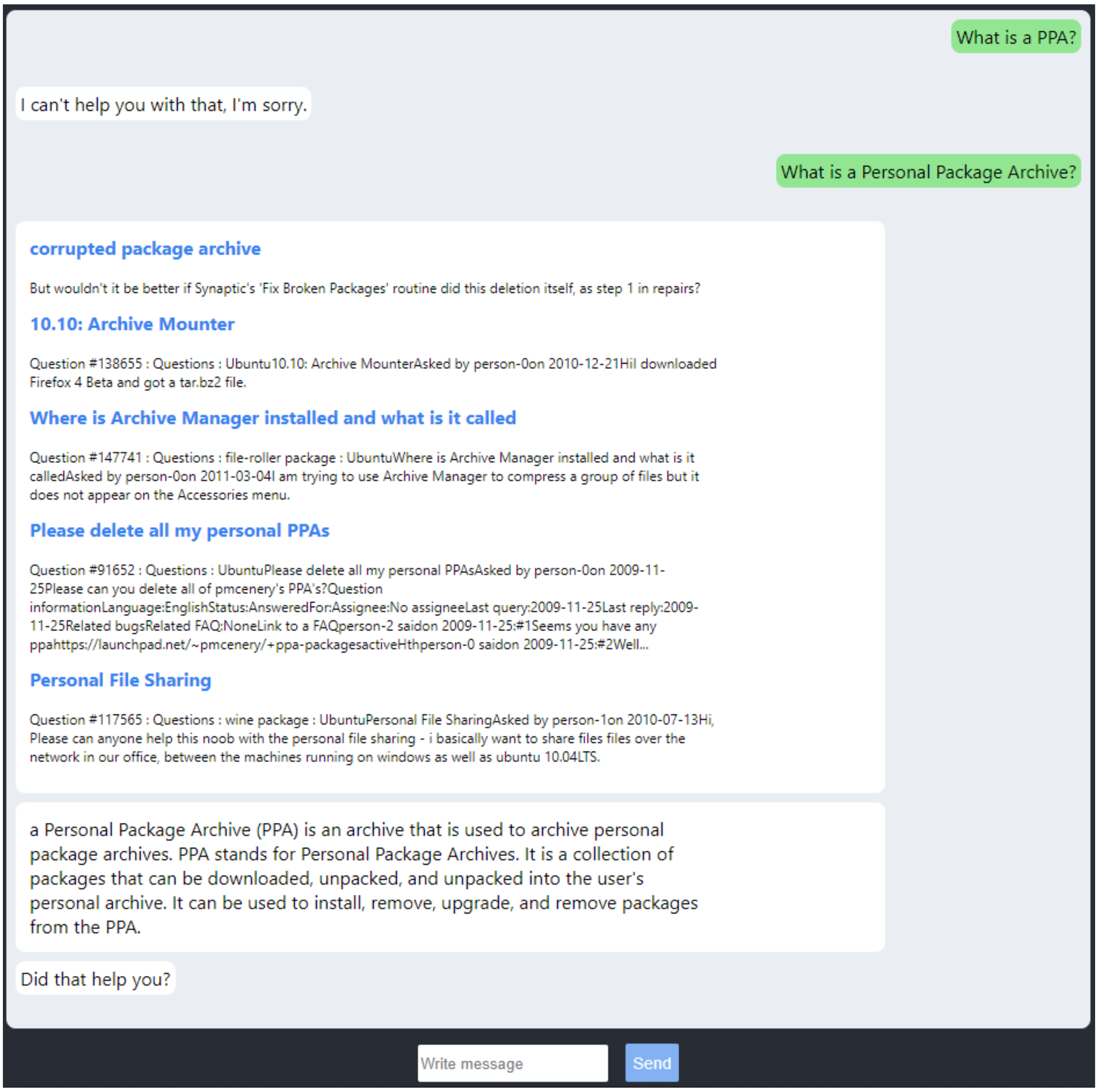


Figure 9: The system can not link the acronym PPA to the actual entity the question is about, personal package archive.

5 Discussion and Conclusions

The previous section reported the results of applying the evaluation methods described in earlier sections to the implementation and provided analysis on the results and what caused them. This section will expand on this analysis by answering the research questions and discussing the limitations, interesting topics for related future research, and applications for an implementation like the one described in this thesis.

The fallback module can help in forming follow-up questions and reduce the number of times the conversation ends up in a situation where the chatbot can not generate a response. However, the performance of the module is very much dependent on the quality of the ontology being used. The ontology needs to cover the domain of the chatbot to be able to provide relevant results consistently. As shown in the Results section, having an ontology that covers the domain of this specific use case would have improved the performance of the ontology module, while the one used did not provide clear improvements in terms of relevant recommendations and follow-up messages. Further research is required to determine the extent to which the module can improve the baseline performance, but it is clear that even with an ontology that does not capture large parts of the chatbot's domain, some error situations can be avoided. In the scope of this thesis, querying for synonym terms provided the most relevant results. With a more fitting ontology, utilizing the other relationships defined in the ontology could yield even better results.

5.1 Limitations and future research

The model built here offers a template for more sophisticated handling of misinterpretations and question clarifying. An important step in improving this model and examining its full potential is applying it to a use case, where there is an ontology specifically for that domain available. In the implementation and tests presented in this thesis, the biggest obstacle was that there does not exist a non-commercial, freely available ontology that details Ubuntu and Linux domains. Using the Computer Science Ontology was the best option available but since it lacks the details of Ubuntu-specific system and software terminology, a massive amount of relevant data detailing the relationships between different entities is missing. This is the single largest cause for the results of this thesis, which are not significantly different from the baseline implementation. Taking the Rasa module developed here into another context where a more comprehensive ontology is available or building a Ubuntu ontology for this use case would probably significantly improve the test results presented. With improved performance, fine-tuning the confidence thresholds in Rasa could prove to be an easy way to further boost the performance of the system. In this implementation, different threshold values were tested but because the results from the ontology were not relevant the values for a working implementation can look very different.

As shown in the results section, having a "perfect" ontology would provide better results as the ontology would always be able to find the relevant entities. It is good to keep in mind that building a decent ontology for any domain is time consuming and requires manual work and there are a lot of obstacles in the way of building this perfect ontology. Taking the implemented module into an application where a tested and validated ontology exists is the easiest way to delve deeper into this direction, but perhaps a more interesting solution to the problem of bad ontology coverage would be building an ontology while training the underlying machine learning models. If a part of the ontology used could be constructed from the training data of the NLP models, the domain could be captured straight from the source. Using an algorithm like Klink-2, the algorithm used to generate CSO, would be a viable starting point for exploring the possibility of domain-specific ontology generation.

While building a custom module for handling error cases, using domain knowledge can help in pushing chatbots in the correct direction but other options for solving this problem exist as well. In the case of the implementation of this thesis, it is reasonable to believe that training the Rasa NLU model on a larger and improved set of data and using a longer training period would improve the model's ability to handle user input to a greater extent than introducing domain knowledge using ontologies. For a larger, more fine-tuned model than the one used in this thesis, an ontology module can be useful in removing some edge cases. However, the same effect could probably be achieved by extending the training dataset to cover the missing information. If the chatbot is able to handle the domain it was built for perfectly, there would be no need for a module such as the one described here.

Another possible avenue of further research is the more effective usage of ontologies. The implementation presented here used only the "sameAs" attribute of the CSO to look for relationships inside the ontology. Because the implementation lacked a fitting ontology with decent coverage over the training data, exploring other attributes and relationships inside the ontology was not meaningful. The "sameAs" attribute provides information about synonyms of the given term. While this is one of the most obvious relationships to utilize in searching for related information, most of the relationships expressed in the ontology were left unused. Many of these connections could likely be used to build a more complex system that can find the correct term the user meant to query for with a higher likelihood. It would be interesting to see a model trained on some ontology data and thus understand these relationships that would then be the "assistant" to the chatbot in that domain, trying to recommend the correct search terms when given the user's message as an input.

In the implementation presented here the need to choose the best option out of the results found from the ontology was not present. This is due to the aforementioned limitations: the ontology did not match the domain and thus was unable to find many results. Once the domains match, however, choosing the most relevant options to show to the user becomes an important aspect of the implementation. A heuristic for choosing the most fitting or the most similar response in terms of semantical meaning needs to be developed. The system needs to understand the quality of the returned entities and how close they are to the original term to be able to give relevant recommendations.

A more complex solution utilizing multiple different ontologies is also a viable topic of research. A chatbot that needs to be able to discuss a broad range of topics would need quality ontologies for each domain. The key questions in a system like this would be how the ontologies are integrated together and how well they can access data across their borders. An algorithm for choosing the most relevant ontology to start from needs to be deployed as the computational costs of a large-scale ontology system would be higher than the single-ontology implementation of this thesis.

Another point of interest for chatbots is the possibility to translate ontologies in cases where the languages differ. For example, a chatbot for Finnish healthcare that needs to function in English would require an ontology that is likely to exist only in Finnish. To gain access to this domain knowledge the terminology must first be translated. Translating ontologies could improve the usability of such systems even more if the domain terminology is bilingual or contains many terms that only exist in the target language.

5.2 Conclusions

This thesis introduced an implementation for handling low-confidence user inputs in a chatbot. The results of evaluating this implementation do not show a clear improvement when compared to the baseline performance. The main cause for this seems to be the ontology used, which does not cover the Ubuntu domain well enough for the recommendations to be relevant. The system is able to form recommendations based on the terms it retrieves from the ontology, but further research is required to discover how relevant the recommendations can be if the ontology would have better coverage. Although the implementation did not significantly better the baseline in terms of giving the expected response faster or with less effort, the ontology used and the architecture of the custom fallback module did produce some useful recommendations to the users. These provide a solid basis for more complex ontology integrations that could potentially prove more useful.

References

[1] Asbjørn Følstad and Petter Bae Brandtzaeg. "Users' experiences with chatbots: findings from a questionnaire study". In: *Quality and User Experience* 5.1 (2020), pp. 1–14.

[2] Joseph Weizenbaum. "ELIZA—a computer program for the study of natural language communication between man and machine". In: *Communications of the ACM* 9.1 (1966), pp. 36–45.

[3] Alan M Turing. "Computing machinery and intelligence". In: *Parsing the turing test*. Springer, 2009, pp. 23–65.

[4] Richard S Wallace. "The anatomy of ALICE". In: *Parsing the Turing Test*. Springer, 2009, pp. 181–210.

[5] Nick Webb. "Rule-based dialogue management systems". In: *Proceedings of the 3rd International Workshop on Human-Computer Conversation, Bellagio, Italy*. 2000.

[6] Oliver Lemon et al. "An ISU dialogue system exhibiting reinforcement learning of dialogue policies: generic slot-filling in the TALK in-car system". In: *In Proceedings of the 11th Conference of the European Chapter of the ACL: Posters & Demonstrations*. 2006, pp. 119–122.

[7] Antoine Bordes, Y-Lan Boureau, and Jason Weston. "Learning end-to-end goal-oriented dialog". In: *arXiv preprint arXiv:1605.07683* (2016).

[8] Michael L Mauldin. "Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition". In: *AAAI*. Vol. 94. 1994, pp. 16–21.

[9] Tanja Bunk et al. "Diet: Lightweight language understanding for dialogue systems". In: *arXiv preprint arXiv:2004.09936* (2020).

[10] Benjamin Roth et al. "Effective Slot Filling Based on Shallow Distant Supervision Methods". In: (Jan. 2014).

[11] Yu Wang, Yilin Shen, and Hongxia Jin. "A Bi-Model Based RNN Semantic Frame Parsing Model for Intent Detection and Slot Filling". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. 2018, pp. 309–314.

[12] Jinhyuk Lee et al. "Learning dense representations of phrases at scale". In: *arXiv preprint arXiv:2012.12624* (2020).

[13] Xiang Zhang, Junbo Zhao, and Yann LeCun. "Character-level Convolutional Networks for Text Classification". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper/2015/file/250cf8b51c773f3f8dc8b4be867a9a02-Paper.pdf.

[14] Zhiwei Zhao and Youzheng Wu. "Attention-Based Convolutional Neural Networks for Sentence Classification." In: *INTERSPEECH*. 2016, pp. 705–709.

[15] Suman Ravuri and Andreas Stolcke. "Recurrent neural network and LSTM models for lexical utterance classification". In: *Sixteenth Annual Conference of the International Speech Communication Association*. 2015.

[16] Zichao Yang et al. "Hierarchical attention networks for document classification". In: *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*. 2016, pp. 1480–1489.

[17] Qian Chen, Zhu Zhuo, and Wen Wang. "Bert for joint intent classification and slot filling". In: *arXiv preprint arXiv:1902.10909* (2019).

[18] Ehud Reiter and Robert Dale. "Building applied natural language generation systems". In: *Natural Language Engineering* 3.1 (1997), pp. 57–87.

[19] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[20] Michael Uschold, Michael Gruninger, et al. "Ontologies: Principles, methods and applications". In: *TECHNICAL REPORT-UNIVERSITY OF EDINBURGH ARTIFICIAL INTELLIGENCE APPLICATIONS INSTITUTE AIAI TR* (1996).

[21] George A Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[22] Alok Ranjan Pal and Diganta Saha. "An approach to automatic text summarization using WordNet". In: *2014 IEEE international advance computing conference (IACC)*. IEEE. 2014, pp. 1169–1173.

[23] Bharathi Raja Chakravarthi, Mihael Arcan, and John Philip McCrae. "WordNet gloss translation for under-resourced languages using multilingual neural machine translation". In: *Proceedings of the Second Workshop on Multilingualism at the Intersection of Knowledge Bases and Machine Translation*. 2019, pp. 1–7.

[24] Zhen Li and Derrick Tate. "Automatic ontology generation from patents using a pre-built library, WordNet and a class-based n-gram model". In: *International Journal of Product Development* 20.2 (2015), pp. 142–172.

[25] Xutang Zhang et al. "Ontology-based semantic retrieval for engineering domain knowledge". In: *Neurocomputing* 116 (2013), pp. 382–391.

[26] Hadeel Al-Zubaide and Ayman A Issa. "Ontbot: Ontology based chatbot". In: *International Symposium on Innovations in Information and Communications Technology*. IEEE. 2011, pp. 7–12.

[27] Ming-Hsiang Su, Chung-Hsien Wu, and Yi Chang. "Follow-Up Question Generation Using Neural Tensor Network-Based Domain Ontology Population in an Interview Coaching System." In: *INTERSPEECH*. 2019, pp. 4185–4189.

[28] Aldo Gangemi and Valentina Presutti. "Ontology design patterns". In: *Handbook on ontologies*. Springer, 2009, pp. 221–243.

[29] Aldo Gangemi. “Ontology design patterns for semantic web content”. In: *International semantic web conference*. Springer. 2005, pp. 262–276.

[30] *Rasa documentation - architecture*. https://legacy-docs-v1.rasa.com/user-guide/architecture/. Accessed: 2021-09-20.

[31] *Rasa documentation - Knowledge Base Actions*. https://legacy-docs-v1.rasa.com/core/knowledge-bases/. Accessed: 2021-11-20.

[32] Ryan Lowe et al. “The Ubuntu Dialogue Corpus: A Large Dataset for Research in Unstructured Multi-Turn Dialogue Systems”. In: *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. 2015, pp. 285–294.

[33] *SPARQL Query Language for RDF*. https://www.w3.org/TR/rdf-sparql-query/. Accessed: 2021-11-30.

[34] Angelo A Salatino et al. “The computer science ontology: a large-scale taxonomy of research areas”. In: *International Semantic Web Conference*. Springer. 2018, pp. 187–205.

[35] Francesco Osborne and Enrico Motta. “Klink-2: integrating multiple web sources to generate semantic topic networks”. In: *International Semantic Web Conference*. Springer. 2015, pp. 408–424.

[36] Francesco Osborne, Enrico Motta, and Paul Mulholland. “Exploring scholarly data with rexplore”. In: *International semantic web conference*. Springer. 2013, pp. 460–477.

[37] Bayan Abu Shawar and Eric Atwell. “Different measurement metrics to evaluate a chatbot system”. In: *Proceedings of the workshop on bridging the gap: Academic and industrial research in dialog technologies*. 2007, pp. 89–96.

[38] Claude Ghaoui. *Encyclopedia of human computer interaction*. IGI Global, 2005.

[39] Emily Geisen and Jennifer Romano Bergstrom. *Usability testing for survey research*. Morgan Kaufmann, 2017.

A Questionnaire and results of user tests

This appendix details the questions presented to the users during evaluation, as well as the summaries of the answers to those questions. For multiple choice questions all answers are summarized with charts, for open text questions the most interesting or relevant answers are presented, while more general and otherwise unproductive answers are left out. Most questions were a combination of multiple choice and a text field to elaborate on the tester's choice. For these, the results are summarized in a similar way.

A.1 Question 1: How would you evaluate your own knowledge about Ubuntu?

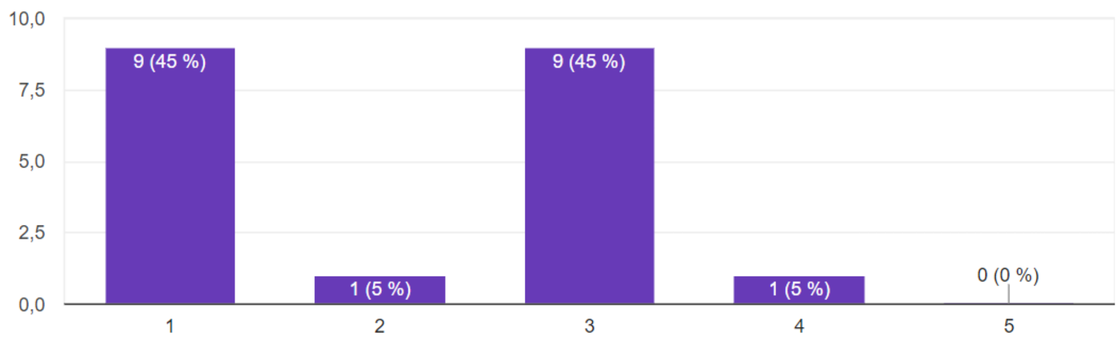
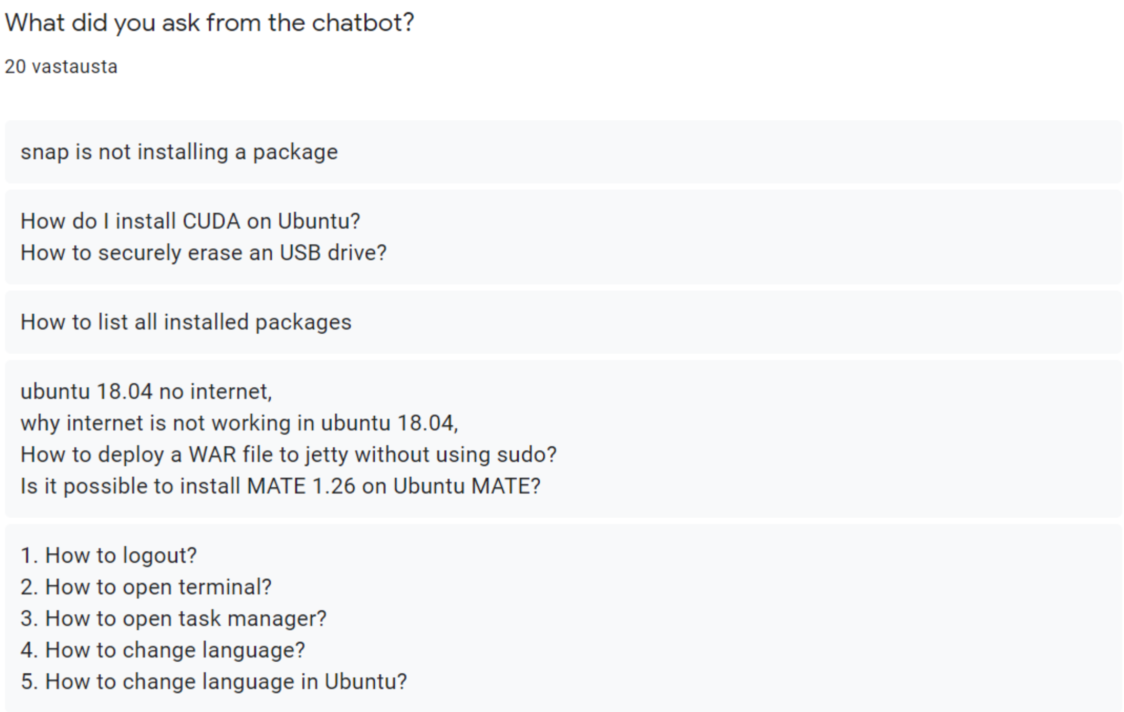


Figure A1: Results for question 1. The answers are on a scale of 1 (very limited experience) to 5 (experienced user).

A.2 Question 2: What did you ask from the chatbot?


What did you ask from the chatbot?

20 vastausta

snap is not installing a package

How do I install CUDA on Ubuntu?
How to securely erase an USB drive?

How to list all installed packages

ubuntu 18.04 no internet,
why internet is not working in ubuntu 18.04,
How to deploy a WAR file to jetty without using sudo?
Is it possible to install MATE 1.26 on Ubuntu MATE?

1. How to logout?
2. How to open terminal?
3. How to open task manager?
4. How to change language?
5. How to change language in Ubuntu?

Figure A2: Results for question 2. Not all questions the users asked are presented here, but the ones displayed here demonstrate the nature of the questions

A.3 Question 3: Did the questions get answered?

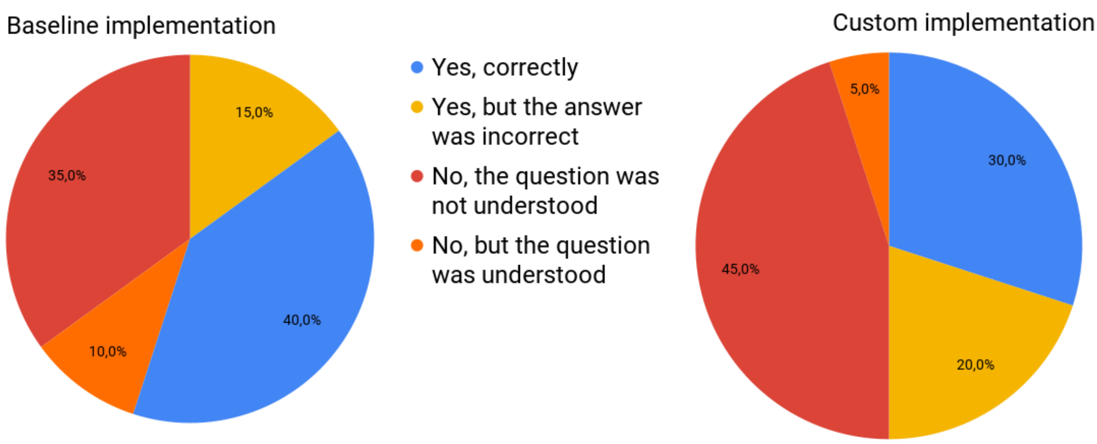


Figure A3: Results for question 3.

Baseline implementation	Custom implementation
Potential fixes were proposed if the bot understood the questions. However, it didn't understand simple beginner questions like "what is sudo" or "what is useradd". Summaries for possible links provided were hard to understand. Not too human-readable.	3 (out of 5) first questions were not understood
The first question led to one follow-up question that I was unable to answer (probably my bad). After that, Chatbot asked me to rephrase. After that, it took Chatbot a minute or several minutes to anything that I wrote. I ended up refreshing the page several times and trying asking again.	Chatbot failed to understand question "How to use Ubuntu?"/"How is ubuntu used?"/"How can i use ubuntu" but recognized rephrasing "how does ubuntu work?". Answer was ok. Did not understand question "shortcut to delete whole line (not just in bash)" but understood "what is shortcut to delete whole line". For question "Suspend mode doesn't work" chatbot asked if I meant "sliding mode control" or "hilbert–huang transform"
The bot gives sudo commands to execute the task. These should work in at least some configurations, though linking the official installation guide for CUDA might be more helpful.	The second question had a better answer earlier, telling to use shred. Now it tells to *search* for a disk utility program, not install it.

Table A1: Comments on question three for both phases of testing. The comments to this question reveal that most testers received a wide array of different responses and choosing an option for the multiple choice part was not trivial.

A.4 Question 4: Were the returned answers relevant to the context?

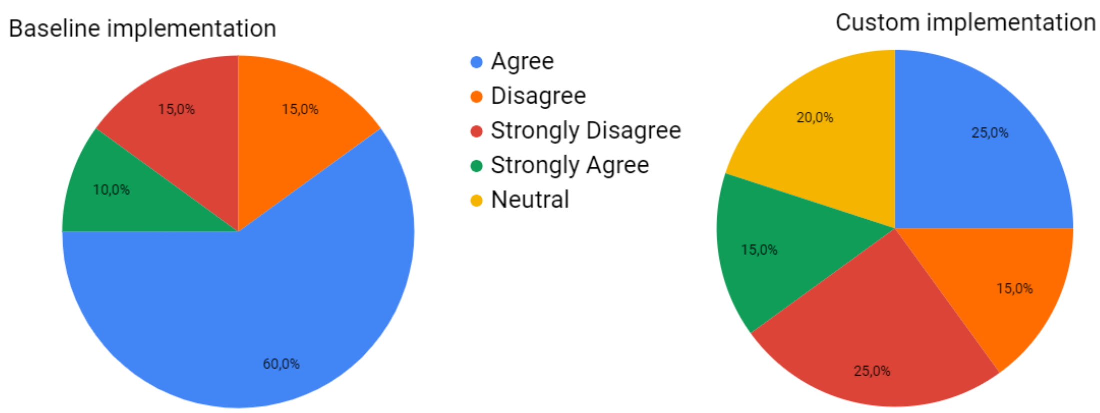


Figure A4: Results for question 4.

Baseline implementation	Custom implementation
They contained the command to run, so yes. But there was also extra somewhat related content in the answers at the end that can confuse and which was cut off abruptly. For the second question, the answer started with " 's answer is good, but if you really want to erase the drive, you can use shred.", after which the helpful and relevant part started.	First question yes, second less relevant.
No mention of Bluetooth. Some of the answers were Raspbian related.	The answer to the first question seemed like something worth trying.
The answers were related to the computer model I replied to the bot, but not the original question	The bot seemed to recognise something from my text and veered off heavily into that direction, disregarding the overall context

Table A2: The text answers for question 4 support figure A4; the responses of the custom implementation are more mixed than the baseline implementations.

A.5 Question 5: Is the answer grammatically well-formed (a fluent english sentence)?

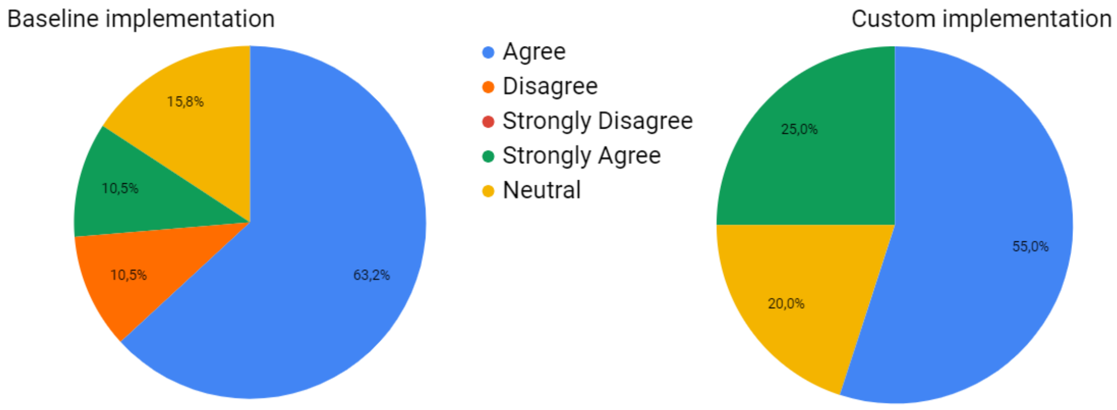


Figure A5: Results for question 5. This question did not gather many text answers and thus they are omitted.

A.6 Question 6: Did any the dialogues involve the chatbot asking for affirmation? (e.g. "did you mean x?")

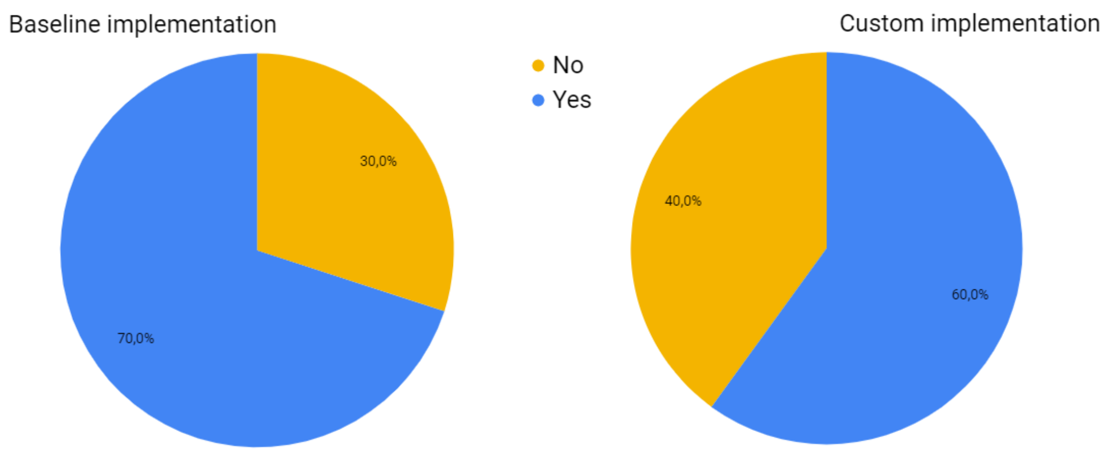


Figure A6: Results for question 6. Only the testers who answered "Yes" to this question answered the rest of the questions, as they were about the chatbot's abilities to handle low-confidence inputs.

A.7 Question 7: Was the suggestion useful?

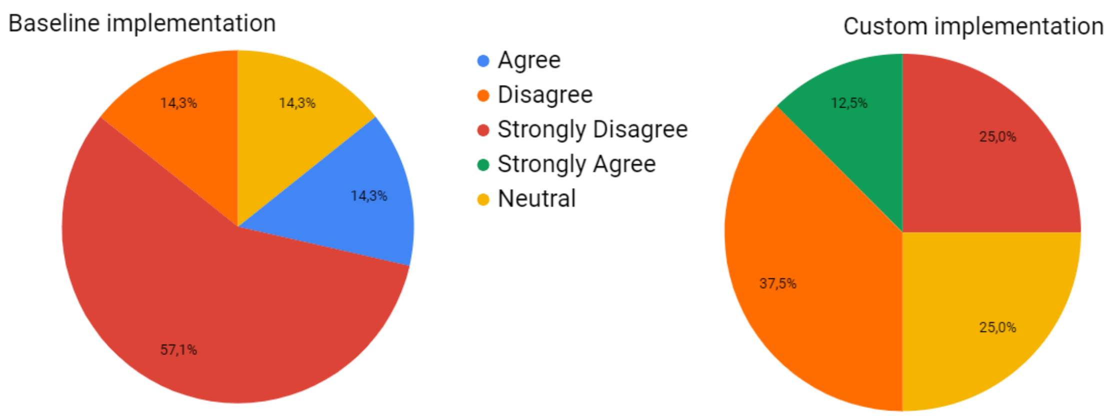


Figure A7: Results for question 7.

Baseline implementation	Custom implementation
The bot asked what operating system I was using. However it threw the bot off.	The suggestion did you mean x was totally out of context.
Did not suggest anything	With my own limited knowlegde of Ubuntu I could not understand how the suggestions were related to my question. However, it guided me to try rephrasing the quesion
Not at all. And when I said no chatbot just gave up and said bye :D	Asked me "Did you mean to ask about "phase-change memory"?" when asked about backgrounds

Table A3: Text answers for question 7 further demonstrate how the baseline implementation is unable to make any kind of suggestions. The reception of the custom module is mixed.

A.8 Question 8: Was the suggestion relevant to the given context?

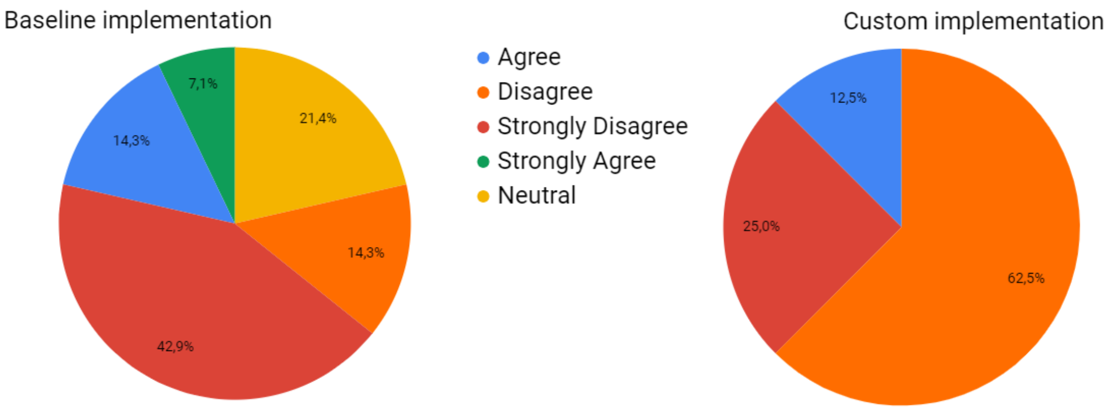


Figure A8: Results for question 8. There were only a few text answers for this question and they were very similar to the answers in the last question. They are not presented here, because they do not provide any new information.

A.9 Question 9: Did the question ask for new information currently not included in the context?

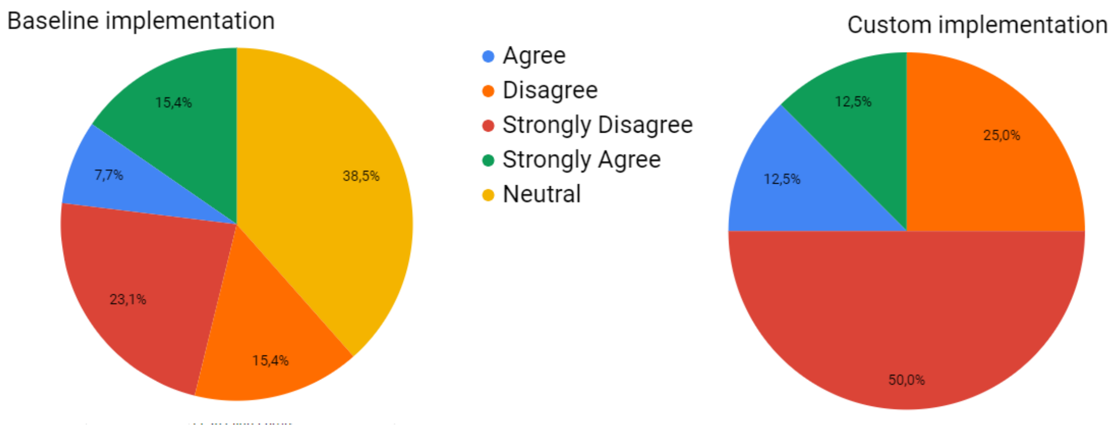


Figure A9: Results for question 9. This question did not receive any text answers.