NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

M.SC. in LANGUAGE TECHNOLOGY

MSc THESIS

A Voice Assistant for cooking based on a natural language to SPARQL transformation pipeline using Rasa

Eleni P. Tsili

Supervisor: Athanasios Katsamanis, Principal Researcher, Athena R.C.

ATHENS

MARCH 2023

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΙΔΡΥΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ”ΓΛΩΣΣΙΚΗ ΤΕΧΝΟΛΟΓΙΑ”

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός και ανάπτυξη ενός φωνητικού βοηθού για μαγειρική βασισμένου στη μετατροπή φυσικής γλώσσας σε SPARQL με τη χρήση του λογισμικού Rasa

Ελένη Π. Τσίλη

Επιβλέπων: Αθανάσιος Κατσαμάνης, Κύριος Ερευνητής, Ε.Κ. ΑΘΗΝΑ

ΑΘΗΝΑ

ΜΑΡΤΙΟΣ 2023

MSc THESIS

A Voice Assistant for cooking based on a natural language to SPARQL transformation pipeline using Rasa

Eleni P. Tsili
S.N.: lt1200024

SUPERVISOR: Athanasios Katsamanis, Principal Researcher, Athena R.C.

EXAMINATION COMMITTEE: Athanasios Katsamanis, Principal Researcher Athena R.C
Vassilis Katsouros, Principal Researcher Athena R.C
Katerina Pastra, Senior Researcher Athena R.C

Examination Date: 1 March, 2024

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός και ανάπτυξη ενός φωνητικού βοηθού για μαγειρική βασισμένου στη μετατροπή φυσικής γλώσσας σε SPARQL με τη χρήση του λογισμικού Rasa

Ελένη Π. Τσίλη
Α.Μ.: lt1200024

ΕΠΙΒΛΕΠΩΝ: Αθανάσιος Κατσαμάνης, Κύριος Ερευνητής, Ε.Κ. ΑΘΗΝΑ

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: Αθανάσιος Κατσαμάνης, Κύριος Ερευνητής Ε.Κ. ΑΘΗΝΑ
Βασίλειος Κατσούρος, Κύριος Ερευνητής Ε.Κ.ΑΘΗΝΑ
Κατερίνα Πάστρα, Εντεταλμένη Ερευνήτρια Ε.Κ.ΑΘΗΝΑ

Ημερομηνία Εξέτασης: 1 Μαρτίου 2024

ABSTRACT

Voice assistants are systems that can interact with humans based on spoken natural language and they are an exciting and fast developing field of Artificial Intelligence and Natural Language Processing. They mainly fall under two categories based on their function,namely chat-oriented systems and task-based agents. In the current thesis we present the design and development of SousChef, an end-to-end, task-based voice assistant for cooking. The goal of the system is to assist a user during the execution of a recipe, by answering recipe-related questions. At the basis of the system lies a pipeline that converts natural language questions to SPARQL queries using Rasa. To this end, a recipe ontology has been created and an NER model has been trained on a set of Greek recipes annotated with recipe-related entities. Rasa NLU pipeline with DIET classifier is utilized for training the classifier. Later on, Rasa Stack is used to train the entire dialogue system on questions related to recipe preparation, by fine-tuning the already trained model. When the assistant receives a URL from the user, it extracts the recipe text from the HTML code and feeds it to the NER model. The extracted entities dynamically fill the ontology which is then ready to be queried. Each time a user asks a question, depending on the intent and the entities found, an action is triggered that executes a SPARQL query on the filled ontology. The query result is converted to an answer which is sent back to the user.

In order to convert the system into a voice assistant, we use a pipeline of microservices that connect to Google's APIs for Speech-to-Text and Text-to-Speech services. The system can be used through a web UI that records speech, sends it to the Speech-to-Text API which converts it to string. The text is then sent to Rasa and an answer is returned. This is then transformed to speech through the Text-to-Speech API and is transmitted back to the user. Finally, we have conducted a thorough evaluation of the assistant, using both an automatic and human-based method. The overall results show that our system is capable of conducting an effective and efficient dialogue.

SUBJECT AREA: Natural Language Processing

KEYWORDS: voice assistants, Rasa, natural language query formalization, SPARQL, ontology, speech recognition

ΠΕΡΙΛΗΨΗ

Οι φωνητικοί βοηθοί είναι συστήματα που μπορούν να αλληλεπιδρούν με τον άνθρωπο με εκφωνούμενη φυσική γλώσσα και αποτελούν ένα ενδιαφέρον και ταχέως αναπτυσσόμενο πεδίο της Τεχνητής Νοημοσύνης και της Επεξεργασίας Φυσικής Γλώσσας. Κατατάσσονται κυρίως σε δύο κατηγορίες με βάση τη λειτουργία τους, τα συστήματα που προσανατολίζονται στη συνομιλία και αυτά που έχουν ως στόχο τη διεκπεραίωση εργασιών. Στην παρούσα διπλωματική εργασία παρουσιάζουμε τον σχεδιασμό και την ανάπτυξη του SousChef, ενός ολοκληρωμένου φωνητικού βοηθού για μαγειρική. Ο στόχος του συστήματος είναι να βοηθήσει τον χρήστη στην εκτέλεση μιας συνταγής, απαντώντας σε σχετικές ερωτήσεις. Στη βάση του συστήματος βρίσκεται ένας μηχανισμός που μετατρέπει τις ερωτήσεις φυσικής γλώσσας σε ερωτήματα SPARQL, χρησιμοποιώντας το λογισμικό Rasa. Για το σκοπό αυτό, έχει δημιουργηθεί μια οντολογία συνταγών και επίσης έχει εκπαιδευτεί ένα μοντέλο αναγνώρισης ονομαστικών οντοτήτων (NER). Τα δεδομένα εκπαίδευσης του μοντέλου αποτελούνται από ένα σύνολο ελληνικών συνταγών επισημειωμένων με οντότητες σχετικές με τη συνταγή. Για την εκπαίδευση του μοντέλου χρησιμοποιείται το τμήμα του Rasa που αφορά στην κατανόηση φυσικής γλώσσας, βάση του οποίου αποτελεί η αρχιτεκτονική DIET. Στη συνέχεια, για την εκπαίδευση ολόκληρου του διαλόγικού συστήματος χρησιμοποιείται το όλο Rasa Stack, με δεδομένα εκπαίδευσης ένα σύνολο ερωτήσεων που σχετίζονται με την παρασκευή μιας συνταγής. Το μοντέλο δεν εκπαιδεύεται από την αρχή, αλλά γίνεται fine-tuning του ήδη εκπαιδευμένου μοντέλου ταξινόμησης. Έτσι, όταν ο βοηθός λαμβάνει μια διεύθυνση URL από τον χρήστη, εξάγει από τον κώδικα HTML το κείμενο που περιλαμβάνει τη συνταγή και το τροφοδοτεί στο μοντέλο NER. Οι οντότητες που εξάγονται γεμίζουν δυναμικά την οντολογία, η οποία στη συνέχεια είναι έτοιμη για αναζήτηση. Κάθε φορά που ο χρήστης θέτει μια ερώτηση, ανάλογα με την πρόθεση και τις οντότητες που βρέθηκαν σε αυτήν, εκτελείται ένα ερώτημα SPARQL στη συμπληρωμένη οντολογία. Το αποτέλεσμα του ερωτήματος μετατρέπεται σε απάντηση, η οποία αποστέλλεται πίσω στον χρήστη.

Προκειμένου να μετατρέψουμε το σύστημα σε φωνητικό βοηθό, χρησιμοποιούμε ένα λογισμικό που το συνδέει με τα API της Google για αναγνώριση και σύνθεση φωνής. Ο φωνητικός βοηθός μπορεί να χρησιμοποιηθεί μέσω μιας διαδικτυακής εφαρμογής που καταγράφει την ομιλία και την αποστέλλει στην προγραμματιστική διεπαφή μετατροπής φωνής σε κειμένο. Στη συνέχεια, το παραγώμενο κείμενο αποστέλλεται στο Rasa και επιστρέφεται μια απάντηση. Μέσω της προγραμματιστικής διεπαφής σύνθεσης ομιλίας, αυτή μετατρέπεται σε ομιλία και μεταδίδεται πίσω στον χρήστη. Τέλος, πραγματοποιήσαμε μια αναλυτική αξιολόγηση του βοηθού, χρησιμοποιώντας τόσο αυτόματες μεθόδους όσο και αξιολογήσεις από χρήστες. Τα συνολικά αποτελέσματα δείχνουν ότι το σύστημά μας είναι ικανό να διεξάγει έναν αποτελεσματικό διάλογο που ως επι το πλείστον επιστρέφει τις κατάλληλες απαντήσεις στον χρήστη.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Επεξεργασία Φυσικής Γλώσσας

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: φωνητικοί βοηθοί, Rasa, τυποποίηση ερωτημάτων φυσικής γλώσσας, SPARQL, οντολογία, αναγνώριση φωνής

To my family

ACKNOWLEDGMENTS

First, I would like to thank my professor, Nassos Katsamanis, for providing me the necessary guidance to complete my master thesis. I would also like to thank Charalampos Leontidis and Kosmas Palios from Athens RC, for creating the framework that gave voice to my assistant and taking the time to help me with the necessary set up. Finally, I would like to thank my family for their constant support over the course of my studies.

CONTENTS

1 INTRODUCTION 23

1.1 Conversing with machines 23

1.2 Motivation 23

1.3 Contribution 24

1.4 Thesis outline 25

2 BACKGROUND AND RELATED WORK 27

2.1 Introduction to conversational agents 27

2.2 Chatbots 27

2.2.1 ELIZA 28

2.2.2 PARRY 28

2.2.3 ALICE 29

2.3 Voice assistants 29

2.4 Architectures 30

2.4.0.1 Modular architecture 30

2.4.1 End-to-end architecture 32

2.4.2 Voice-enabling components 33

2.4.2.1 Automatic Speech Recognition (ASR) 33

2.4.2.2 Text-to-Speech (TTS) 34

2.5 Dialogue systems amd Ontologies 34

2.6 Evaluation of task-oriented dialogue systems 35

2.7 RASA framework 35

2.7.1 Dual Intent and Entity Transformer 36

2.7.1.1 Main components 37

2.7.2 DIET performance 39

2.8 Background Summary 40

3 SOUS CHEF DIALOGUE SYSTEM 41

3.1 Methodology **41**
3.1.1 Recipe Ontology 41
3.1.2 Named Entity Recognition model 43

3.2 System architecture **47**
3.2.1 Rasa NLU 47
3.2.2 Rasa Core 48
3.2.3 Natural language to SPARQL conversion pipeline 49

3.3 Speech Recognition and Speech Synthesis **52**

3.4 Methodology Summary **52**

4 EVALUATION OF THE DIALOGUE SYSTEM 55

4.1 Automatic evaluation **55**
4.1.1 Intent classification and entity recognition 55
4.1.2 Efficacy and efficiency 56

4.2 Human evaluation **59**

4.3 Evaluation Insights and Discussion **59**

5 CONCLUSIONS AND FUTURE WORK 61

ABBREVIATIONS - ACRONYMS 63

REFERENCES 66

LIST OF FIGURES

2.1 An example of PARRY's conversation with ELIZA [1], showcasing ELIZA's effort to mimic a psychotherapist by redirecting the questions and PARRY mimicking a schizophrenic patient by providing crude, incohesive answers. 29
2.2 Architecture of a task-oriented dialogue system where the user input, after it is transformed to text by the Automatic Speech Recognition (ASR) part, it goes through the Natural Language Understanding (NLU) part which extracts the user intent as well as any entities found. The Dialogue State Tracking (DST) part keeps track of the user input and the current dialogue state. Based on the state, the Policy Learning (PL) part predicts the system's next action. Finally, the Natural Language Understanding (NLG) maps the action to natural language and sends it to the Text-to-Speech (TTS) part that produces the audio. [2] 31
2.3 An example of a Fusion network which consists of the neural architectures for each of the tree main components of dialog systems, namely Natural Language Understanding (NLU), Dialogue Management (DM) and Natural Language Generation (NLG). The individual neural modules are learned simultaneously with the end-to-end task of dialog generation.[3] 33
2.4 Rasa architecture flowchart. The user message first passes through the Rasa NLU (Interpreter) that extracts the intent and entities, if any. The Tracker contains the conversation state, based on which the Policy predicts the next system action which returns an answer to the user. The Tracker maintains the logs of the executed actions. [4] 36
2.5 The Dual Intent and Entity Transformer (DIET) architecture that is at the core of Rasa and utilized in the implementation of SousChef. [5] 37

3.1 End-to-end pipeline of SousChef assistant. After the user provides a recipe URL, useful text is extracted from the HTML code and fed to the NER model. The extracted entities fill the recipe ontology, which is now ready to be queried. Each time the user makes a query, this is converted to SPARQL and is made on the ontology. The query answer is formatted in natural language and returned to the user. 42
3.2 Example of a SPARQL query that tries to find the mode of an ingredient from the information in the ontology. There are two main parts in the query: 1. the SELECT clause which identifies the variables to appear in the query results and 2. the WHERE clause which has the triple patterns. 43
3.3 The recipe ontology. On the left side, there are the ontology classes (nodes) and how they are interconnected and on the right side there are the properties (edges) that connect the classes. 44
3.4 The entities the NER model was trained on. 44

3.5 Results from the various configurations the NER model was trained on, in terms of F1-score, precision and recall. The best performing model is the one trained only on sparse features with masking activated. 46
3.6 The intents the Rasa NLU was trained on. 48
3.7 Example of the transformation of a natural language query to a SPARQL query, by utilizing intents and entities extracted by the DIET classifier in order to trigger a SPARQL query with the relevant parameters. This pipeline is described in detail in Section 3.2. 50
3.8 The web application through which users can interact with the SousChef assistant. 52

4.1 Part of a user dialogue (1). The dialogue flow is from bottom to top, as the user (U) asks a question and the bot (B) provides the answer. This example is from the annotation process during the automatic evaluation. 57
4.2 Part of a user dialogue (2). The dialogue flow is from bottom to top, as the user (U) asks a question and the bot (B) provides the answer. This example is from the annotation process during the automatic evaluation. 58
4.3 Part of a user's interaction with the assistant, where we notice that the assistant fails at the execution of one of the tasks as the user doesn't refer directly to a concept but uses anaphora. 60

LIST OF TABLES

3.1 The different configurations the DIET classifier was trained on, for the task of entity recognition. 45

4.1 Results from automatic evaluation. In phase 1, we tested the system's performance on intent recognition and entity extraction. In phase 2, we evaluated the system's ability to complete tasks (Task Completion Rate), the average dialogue turns it needs to complete them (Task Completion Cost) and we measured how similar system dialogues are to the respective ground truth dialogues (Edit distance). 56

1. INTRODUCTION

In this chapter, we introduce the general scope of our master thesis and we briefly present the issues we aim to address. First, we discuss how dialogue systems have rapidly evolved in the field of Natural Language Processing and we explain our motivation behind this work. We then present the main contributions of our work and we finally describe how the rest of the thesis is outlined.

1.1 Conversing with machines

The human need to converse with machines seems to have emerged since the first machines were developed. There seems to be an intrinsic fascination with communicating with one's own creation, which is also portrayed in literature and cinema in various cases throughout the centuries. Language plays a crucial role in human history and culture and is probably the most important and intricate result of human evolution. Language-in-use or dialogue is present in all parts of life and it is the basic means of communication.

Naturally, when the first computers were developed, the need for interacting with them in natural language emerged as well. It became one of the main challenges in artificial intelligence. One of the first references to systems that can communicate with people was Turing's 'thinking machine' [6]. It was described as a machine that can converse with a person through a typewriter, by imitating human conversation well enough as to trick its interlocutor that it is actually human.

It was at this time, in 1950, that the field of Natural Language Processing (NLP) emerged as the intersection of linguistics and artificial intelligence, bridging these two seemingly unrelated worlds. The fast and continuous progress in artificial intelligence and NLP is what made the development of systems that can converse with humans feasible. These programs, which are meant to interact with users in natural language via text, speech or both, are called dialogue systems or conversational agents and today we have groundbreaking cases of such systems.

1.2 Motivation

The motivation behind this work lies in the same intrinsic excitement surrounding the idea of conversing with a machine that also guided the very first attempts in the field of Natural Language Processing. It is the challenge of transferring such an integral characteristic of human existence to a machine in order to interact with it. This field of Natural Language Processing and Artificial Intelligence that deals with conversational agents is a stimulating research area as it brings this human-machine interaction into realization. What makes this process event more intriguing is that it can lead to a deeper understanding of how our own language works and what complex mechanisms lie behind it.

The development of these virtual assistants has also very useful real-world applications in many aspects of human activity. From customer service chatbots assisting customers with troubleshooting and healthcare assistants providing medical advice, to virtual assistants that can help with a variety of everyday tasks, conversational agents have multiple uses and can provide assistance in all domains of human activity.

In the current thesis we have selected to design and develop a conversational agent that will be able to assist users with an everyday activity that entails high levels of multitasking, namely cooking. SousChef is a voice assistant which is meant to help users during the execution of recipes, by providing all the necessary information at the user's request. We consider this type of system an interesting case as it can have actual, real-life use, in such a necessary and often tiresome activity as cooking. It is also an example of how a system can provide assistance with procedural text and it can be extended to other tasks of everyday life, such as following the instructions of a manual. The user in our system's case may want to know which are the ingredients of the recipe, require a specific amount or ask about how they should cut an ingredient. They can also request information related to the execution of the recipe and the steps they need to follow. Finally, they might ask the system to set a notification on a specific time.

At the beginning of the conversation the user needs to provide a URL with a recipe of their choice. Then the system parses the recipe, extracts useful entities from it and fills a recipe ontology. Each time the user asks a question, depending on the user intent, a query is triggered and performed on the ontology. When the relevant answer is extracted, it is converted to natural language and the audio is returned to the user. For the development of the aforementioned voice assistant, we utilized Rasa Open Source[1], a complete and versatile machine learning framework for building conversational agents. Rasa provides multiple opportunities for customization, it is easily configurable and supports a variety of machine learning libraries. A detailed presentation of the framework can be found in Section 2.6.

1.3 Contribution

Within the scope of this thesis, we have designed and implemented a a voice assistant for cooking which answer user questions by querying an ontology using SPARQL. The ontology is filled with all relevant recipe-related information, each time at the beginning of the conversation when the user provides a recipe URL. The main contributions of our work, which are thoroughly discussed in Chapter 3, are the following:

- Design and creation of a closed domain ontology for cooking recipes using the Protégé[2] tool, an open-source ontology editor and knowledge management tool.

[1] https://rasa.com/docs/rasa/
[2] https://protege.stanford.edu/

- Use of Rasa's DIET model in a pipeline aiming at natural language query formalization to SPARQL queries
- Design and implementation of an end-to-end dialogue system that can assist a user during cooking by answering recipe-related questions

1.4 Thesis outline

The rest of the thesis is organized in the following way:

In **Chapter 2** we provide the necessary background information related to the categorization of conversational agents, important aspects of their architecture and we briefly introduce Rasa framework. **Chapter 3** focuses on the methodology followed for the design and development of our system, providing details about the pipeline followed. **Chapter 4** is dedicated to the evaluation of the overall system and discussion on the results. Finally, **Chapter 5** concludes the main points of our work and discusses issues for future work.

2. BACKGROUND AND RELATED WORK

In Section 2.1 of this chapter, we introduce the categorization and main functionalities of conversational agents. Section 2.2 briefly presents the architecture of early chatbots and Section 2.3 introduces the concept of voice assistants. Furthermore, in Section 2.4 we demonstrate the most common architectures of modern task-oriented dialogue systems and in Section 2.5 we present the link of dialogue system to knowledge bases and ontologies. Section 2.6 introduces the metrics used for evaluating dialogue systems. Finally, in Section 2.6 we introduce Rasa, an open-source framework that is primarily used for the development of the dialogue system introduced in the current thesis.

2.1 Introduction to conversational agents

Conversational agents can either freely converse with users or assist them in tasks through dialogue [7]. Based on their function they are generally divided into two categories: task-oriented dialogue systems and open-domain dialogue systems. The former, converse with the user aiming to complete a specific task, such as give directions or book flights. The latter are designed to conduct more extended dialogue with the user by mimicking the unstructured conversations or 'chats' characteristic of human-to-human interaction without domain restrictions. Those systems can be used both for entertainment purposes or be implemented as part of task-oriented agents in order to make the dialogue more natural and engaging. The current thesis focuses mainly on task-oriented systems.

2.2 Chatbots

The first conversational agents developed were chatbots. These are practically the simplest form of a conversational agent as the user can communicate with the bot only through text. These early chatbots were rule-based, meaning that the conversation flow between user and bot was directed by a set of hand-crafted rules [8]. During these initial attempts, rule-based systems were popular as they were easy to implement and their responses were fairly natural, although they could only support limited and pre-determined dialogue flows.

Apart from rule-based systems, traditional architectures include non-neural machine learning based systems [2]. These models use template filling to perform a task. They are more flexible in terms of the dialogue flow compared to rule-based systems, however they still support limited scenarios because of the fixed templates. This is why today state-of-the-art dialogue systems are deep learning-based systems.

2.2.1 ELIZA

ELIZA [9] is one of the earliest and most important chatbots and was developed by the Artificial Intelligence Laboratory in MIT between 1964-1966. The conversation held with ELIZA was designed to resemble a conversation with a Rogerian psychotherapist, which should be non-directive and has as a main goal to reflect the user-patient's statements back at them. Such a conversation is one of the rare cases where it is acceptable from the listener-system to know almost nothing of the real world. This type of conversation has two important advantages. Not only does it help the user feel heard and understood, but from a technical point it also does not demand from the system to store information of the real world. This idea is exactly what makes ELIZA a simple yet successful conversational agent. However, the system has very limited knowledge and can converse only about specific topics [10].

The user can converse with ELIZA through a type-writer, using natural language. Then ELIZA analyzes the user's statement using pattern matching and generates some response. More specifically, when the user types a statement, the program tries to identify the presence of a keyword or keywords in it. In the case where it finds one, it should transform the sentence according to a rule associated with the specific keyword. If no keywords are found, then ELIZA should respond with a content-free remark or to retrieve an earlier transformation. Keywords and the transformation rules associated with them make up the 'script'.

ELIZA was considered successful at mimicking an actual conversation, as many of the users who interacted with the system reported that it really understood their problems and seemed to had developed an emotional bond with it [11]. Revised versions of the base architecture of ELIZA are still used by some modern chatbots, like ALICE.

2.2.2 PARRY

PARRY [12] is another worth-mentioning achievement in the history of chatbots which was developed at Stanford's Psychiatry Department in 1972 by Kenneth Mark Colby, both a psychiatrist and a computer scientist. In this case, the approach used in order to redirect the user's attention from the bot itself was to impersonate a paranoid schizophrenic patient. The bot's answers were meant to be crude and controversial so that the user had to provide more detailed answers. Colby hoped that PARRY could provide useful insights on mental disorders and a useful tool for the training of psychiatrists [13].

PARRY's architecture resembles ELIZA's, but it presents better language understanding. Another important addition is that PARRY's mental state is controlled by a mental model which includes variables indicating the system's level of fear and anger. Depending on the user input, the value of a certain variable might increase indicating that the bot should provide an answer from the respective collection of outputs. For instance, if the user's input triggers the anger variable, PARRY will respond angrily. An example of ELIZA's interaction with PARRY can be seen in Figure 2.1 .

parry: People get on my nerves sometimes.
eliza: I am not sure I understand you fully.
parry: You should pay more attention.
eliza: Suppose you should pay more attention?
parry: You're entitled to your opinion.
eliza: What makes you think I'm entitled to my opinion?
parry: It bothers me just to be around people.
eliza: Please go on.
[Parry continues with some paranoid ramblings about horseracing, bookies, and the Mafia]
parry: I know the mob controls the big rackets.
eliza: Who are you really talking about?
parry: Have you ever really thought about the Mafia?
eliza: Suppose you ever really thought about the Mafia?
parry: Let's talk about something else.
eliza: I am not sure I understand you fully.

Figure 2.1: An example of PARRY's conversation with ELIZA [1], showcasing ELIZA's effort to mimic a psychotherapist by redirecting the questions and PARRY mimicking a schizophrenic patient by providing crude, incohesive answers.

The fact that the user expected to get nonsensical answers is why PARRY is considered a successful chatbot. Psychiatrists who studied transcripts from conversations with PARRY as well as from real interviews with schizophrenic patients were not able to identify which were the actual human-to-human conversations. In fact, it was the first system known to have passed the Turing test.

2.2.3 ALICE

Another important development in the early history of chatbots is ALICE [14], which stands for Artificial Linguistic Internet Computer Entity and was first implemented by Wallace in 1995. The idea behind ALICE's architecture is customization. The chatbot engine is separated from the language knowledge model, so that the latter can be customized according to the developer's goal. The dialogue is made possible through pattern matching. To this end, artificial intelligence markup language (AIML) is used to store the rules for matching the user input to the appropriate output [10]. Although ALICE has been awarded as the most human-like system in 2000, 2001 and 2004, it has important limitations related to AIML as it cannot carry out a conversation for a long time [15].

2.3 Voice assistants

Conversational agents that enable users to use their voice in order to interact with a device instead of using chat or mouse movement are called voice assistants. Although speech recognition is not a newly discovered field and has been present for a while now, it has seen significant progress over the last few years due to the advances in natural language processing. This continuous progress can be credited to: (i) the increase in computational power, (ii) the exponential increase of linguistic data that is available online and can be used to train the assistants, (iii) the advances in machine learning methods and algorithms and (iv) a more in depth understanding of the structure that lies behind human interaction and language and how this is deployed in social contexts [16]. These advances led to the development of widely used voice-based services, such as Amazon's Alexa, Google Home, etc. It is estimated that there are currently 4.2 billion active voice assistant devices in circulation, a number which is expected to double in 2024 [17].

Generally different voice assistants can have different features and functionalities depending on their audience and scope. However, some of the most common tasks they perform are sending/reading texts or emails, making phone calls, answering general information, setting alarms and reminders, controlling connected applications (e.g., Spotify, Netflix, etc.), controlling Internet-of-Things devices (e.g., door locks, lights, etc.), and telling jokes and stories. Some voice assistants, such as Amazon's Alexa, have some additional functionalities often referred to as "skills" that are able to interact with other services with voice commands.

The most common architectures for task-oriented dialogue systems are modular and end-to-end architectures. Another worth mentioning fact is that most task oriented-systems are connected to external knowledge bases in order to retrieve necessary information to answer user questions [8].

2.4 Architectures

2.4.0.1 Modular architecture

As seen in Figure 2.2 the modular approach generally consists of four modules [8]:

- Natural Language Understanding. It transforms the user input into predefined semantic slots, typically the intent of the user's utterance and entities extracted from it.
- Dialogue State Tracking. It keeps track of the user input at each turn while updating the dialogue history, in order to output the current dialogue state.
- Dialogue Policy. It uses the current dialogue state in order to predict the next action of the system.
- Natural Language Generation. It maps the selected action to natural language.

The Dialogue State Tracking and the Dialogue Policy modules comprise the Dialogue Manager of the conversational agent which can be considered as the central controller of the system.

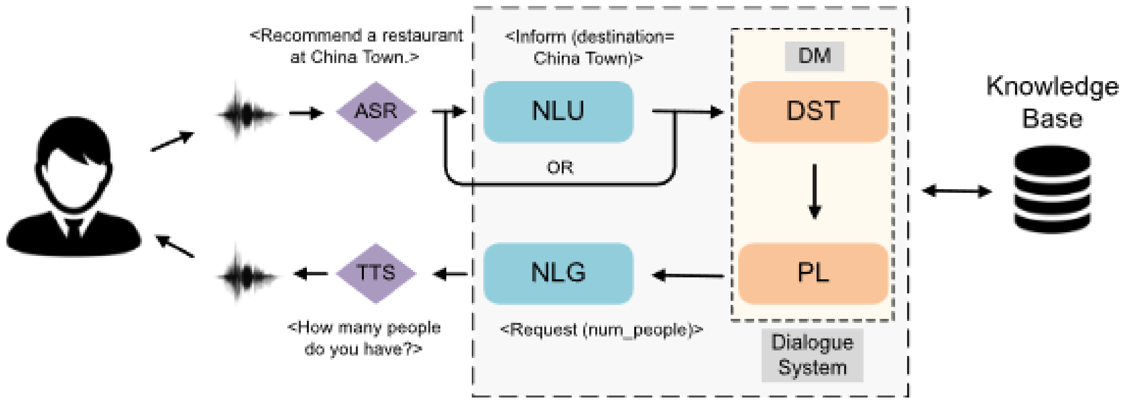

Figure 2.2: Architecture of a task-oriented dialogue system where the user input, after it is transformed to text by the Automatic Speech Recognition (ASR) part, it goes through the Natural Language Understanding (NLU) part which extracts the user intent as well as any entities found. The Dialogue State Tracking (DST) part keeps track of the user input and the current dialogue state. Based on the state, the Policy Learning (PL) part predicts the system's next action. Finally, the Natural Language Understanding (NLG) maps the action to natural language and sends it to the Text-to-Speech (TTS) part that produces the audio. [2]

Natural Language Understanding (NLU)

NLU consists of two sub-tasks: intent classification and entity recognition. Systems that model these two sub-tasks together in a single multi-task architecture are more effective and less error prone than systems that model them separately [5].

Large pre-trained language models have a very good performance in NLU. However, there is a high computational cost both in pre-training and fine-tuning them. Since dialogue systems are not only developed by scientists but also by developers, there is the need for models that can be easily adjusted to a typical software development workflow. Additionally, these large pre-trained models need a considerable amount of available data which poses a problem for low resource languages [5].

Dialogue State Tracking (DST)

This is the first module of the Dialogue Manager and it is responsible for keeping track and updating the dialogue states based on the current user input and the dialogue history. Early dialogue systems used rule-based or statistical methods in DST tasks. However, rule-based methods have a limited generalization capability, increased error rate and low domain adaptability. As for statistical methods, these are also influenced by possible noise in data and ambiguity [2]. More recent works utilize neural trackers which are categorized in two streams: neural trackers with predefined slot names and values and neural trackers with unfixed slot names and values. The former aims to find the most appropriate slot-value pair at each dialogue turn based on the dialogue history. The latter tries to use the dialogue context in order to find or generate the values, since it does not have any predefined slot value list. This approach has proven to be the most flexible of the two.

Dialogue Policy

This is the second module of the Dialogue Manager. It uses the output states from the DST module in order to decide which should be the next action of the dialogue system [8]. More specifically, based on the dialogue state S_t of the current turn and the action set $A = \{a_1, ..., a_n\}$, the Policy needs to learn a mapping function $f : S_t \rightarrow a_i \in A$.

The most common methods used for training the Policy module are supervised and reinforcement learning. Supervised learning is proven to have a good decision-making ability, although it is significantly prone to the quality of the training data. Furthermore, it requires annotated datasets and it is not easily transferable across different domains, as the decision ability of the Policy is subject to each specific task. This is why the most recent dialogue systems rely on reinforcement learning methods [2].

Natural language generation (NLG)

This module is responsible for converting the system action, which has been selected by the Dialogue Manager into an appropriate response in natural language. Conventional NLG systems consist of a pipeline with different components. First the semantic representation of the selected action is mapped into an intermediate representation of the utterance as a tree-like or template structure. Later on, this structure is converted to a response in natural language.[18] However, more advanced systems take advantage of deep learning methods and integrate all previously separated NLG modules in an end-to-end system. Theses have accomplished improved performance in the NLG task [2].

2.4.1 End-to-end architecture

Although modular systems can achieve good performance, there are some drawbacks related to this type of architecture. First, since these are pipeline systems they are usually not differentiable, meaning that back propagation cannot be performed in the whole system but only in each separate module. Furthermore, the improvement of one module does not always help in the improvement of the whole system, creating the need for multiple trainings. Finally, since in modular systems there are some handcrafted rules that are subject to a specific domain, these are not easily transferable to other domains without the necessary alterations. For this purpose end-to-end systems have been proposed.

There are two main types of end-to-end training. The first is to make each module differentiable, in order to be able to perform back-propagation to optimize the parameters in an end-to-end manner. The second, and more recent, is to train only one end-to-end module in the form of a multi-task learning neural model [2].

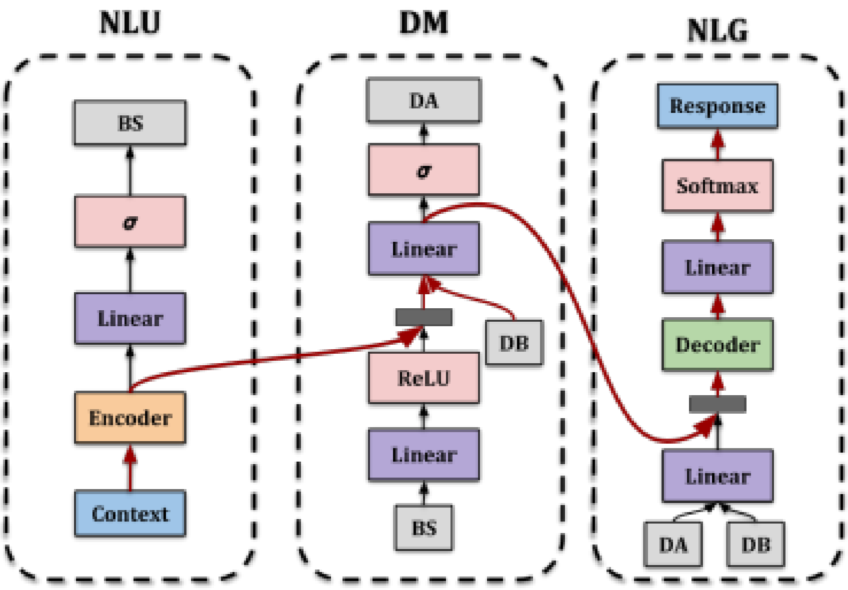

Figure 2.3: An example of a Fusion network which consists of the neural architectures for each of the tree main components of dialog systems, namely Natural Language Understanding (NLU), Dialogue Management (DM) and Natural Language Generation (NLG). The individual neural modules are learned simultaneously with the end-to-end task of dialog generation.[3]

Although end-to-end systems have shown strong performance and achieve high quality answers, they also have a number of drawbacks, including the need for large amounts of annotated data, an inability to generalize, and a lack of controllability. These issues derive from the fact that end-to-end architectures lack the explicit structure of a dialogue. In order to tackle these problems, Fusion Networks have been proposed which try to incorporate structure into neural models. First they learn several neural models that correspond to the different components of a dialogue which are then incorporated in a higher-level generative model [3]. Figure 2.4provides an example of such an architecture.

2.4.2 Voice-enabling components

2.4.2.1 Automatic Speech Recognition (ASR)

Automatic Speech Recognition is responsible for converting the spoken user input into text. In typical ASR systems, the audio signal is first processed by an acoustic front-end in order to generate the relevant feature vector. Next, a decoder tries to match the input sound signal to the most appropriate sequences of words, utilizing a lexicon, a language model and an acoustic model. The lexicon consists of terms related to the current application. The language model is a probabilistic model that generates the most probable sequence of words for the selected language. The acoustic model has learned statistical representations of the sounds that constitute a word and is responsible for predicting the most appropriate representation for the feature vector [19].

In traditional spoken dialogue systems, the output text from the ASR is fed to the NLU module. However, there are more modern approaches that directly pass the audio signal as input to the NLU [20].

2.4.2.2 Text-to-Speech (TTS)

Text-to-speech is a process that models natural language and is responsible for converting text into the corresponding oral representation. In typical TTS systems, the first step is the pre-processing of the input text in order to convert it to the linguistic features of the target language. The next step is to feed the feature vector into an acoustic model which will predict the most appropriate sequence of waveform blocks that will be the output audio [21]. In voice assistants, the output text from the NLG module serves as input to the TTS.

2.5 Dialogue systems amd Ontologies

The need for organizing language in a more meaningful way, by effectively modelling its complexity and connections between concepts, led to the shift from traditional databases that simply store information, to more complex semantically linked knowledge bases. These are repositories of information that also include semantic relationships between concepts. An extension of knowledge bases are ontologies, which are formal representations of a domain-specific knowledge. Ontologies organize knowledge in such a way as to include concepts, entities and the relationships between them [22]. These can be really useful for knowledge base applications, as they model complex domain-specific information in a way that can be understood and processed by a system [23]. However, after modelling the concepts, there is also the need for a formal querying language that can query the ontology and get relevant domain-specific answers. SPARQL, is a case of such a formal language that is designed to query and manipulate an ontology where data is organized in RDF (Resource Description Framework) format [24]. RDF is a directed graph which comprises triple statements in the form of subject-relationship-object. Each part of the statement can be identified by a Uniform Resource Identifier (URI). These graph statements are represented by a node for the subject, an edge for the relationship, and a node for the object.

Conversational systems leverage these types of knowledge representations in order to answer domain-specific queries by introducing ontologies as a component in their pipeline [25]. After the NLU part of the system identifies the user intent, a query can be formulated in order to be performed on the domain-specific ontology, based on the extracted intent and entity. The connection of those ontologies to conversational interfaces makes the domain-specific knowledge and its relations available to any user. This can be especially useful for domains that contain a large number of concepts and relations, i.e. a medical ontology, to be accessible to non-technical professionals that can pose questions in natural language and the respective answers.

2.6 Evaluation of task-oriented dialogue systems

There are various evaluation methods for task-oriented dialogue systems [2]. Since these systems aim at the execution of a specific task, there are some direct evaluation methods, such as Task Completion Rate and Task Completion Cost. Task Completion Rate is computed as the rate of successful events in all task completion attempts and it is a measure of the system's ability to perform a task. The Task Completion Cost is a measure of the required resources when completing a task. An important metric that belongs to Task Completion Cost is time efficiency computed as the number of conversation turns until the completion of the required task. The fewer the turns the more efficient the dialogue is considered.

Additionally, human-based evaluation provides user dialogues and user satisfaction scores for system evaluation. One way to perform this type of evaluation is to test a dialogue system through crowd-sourcing. After users converse with the system, task Completion Rate and Task Completion Cost are calculated. Another way to perform human-based evaluation is to use these metrics in real user conversations with the system after it has been deployed. User simulators can also manage to simulate user dialogues based on pre-defined rules or models. This is a more cost-effective alternative than real-user evaluation as it does not include recruiting human labor.

Metrics like BLEU have also been proposed for the evaluation of dialogue systems which compare the system responses with actual human responses [26].

To evaluate the performance of classification tasks like intent and entity recognition traditional evaluation metrics can be used, i.e., accuracy, precision, recall and F-score.

2.7 RASA framework

Rasa Open Source is an open source machine learning-based framework for building chat and voice-based AI assistants. Rasa framework offers the possibility to create and customize pipelines for intent and entity recognition with high accuracy and while handling context very efficiently. It makes it easy to incorporate and use a vast variety of machine learning libraries. Another advantage of Rasa framework is that conversations with the system can be recorded and reviewed in order to annotate mistakes correctly.

Rasa Stack consists of Rasa NLU (Interpreter) and Rasa Core (Tracker). Rasa Core handles the conversation flow, the bot's responses and the session management. More precisely, Rasa Core keeps track of the current state of the conversation. After the user gives an input sentence, this passes through Rasa NLU in order to extract the respective intent and entities. Based on the user input, the extracted entities and intents and the current state of the dialogue, the Policy decides which action is to be executed next. Rasa

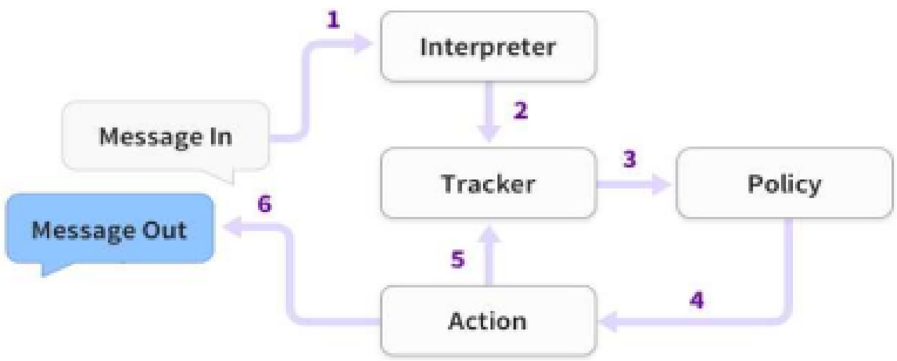

Figure 2.4: Rasa architecture flowchart. The user message first passes through the Rasa NLU (Interpreter) that extracts the intent and entities, if any. The Tracker contains the conversation state, based on which the Policy predicts the next system action which returns an answer to the user. The Tracker maintains the logs of the executed actions. [4]

provides both rule-based and machine-learning based policies that can also be used collaboratively. All policies make a prediction and the one with the highest score is selected. An overview of Rasa's workflow can be seen in Figure 2.4.

What makes Rasa a valuable tool is that it offers the option to use various configurations for the training of both the Rasa NLU and Rasa Core component. One of the main components that Rasa provides is the Dual Intent and Entity Transformer (DIET) that can be used in the NLU part and perform entity recognition and intent extraction. An detailed description of DIET architecture can be found in the following subsection.

2.7.1 Dual Intent and Entity Transformer

Dual Intent and Entity Transformer (DIET) [5] is a multi-task architecture that performs both intent classification and entity recognition by combining pre-trained word and sentence embeddings with sparse features, such as word level n-grams and character level n-grams. Both components can be used in a plug-and-play manner. After the featurization, DIET proceeds with feed-forward layers, a two layer transformer and a Conditional Random Field layer. It also introduces masking, a regularization technique that masks some token in the input sentence that the model has to predict. The following subsections discuss the DIET architecture in more detail. An overview of the DIET architecture can be found in Figure 2.5

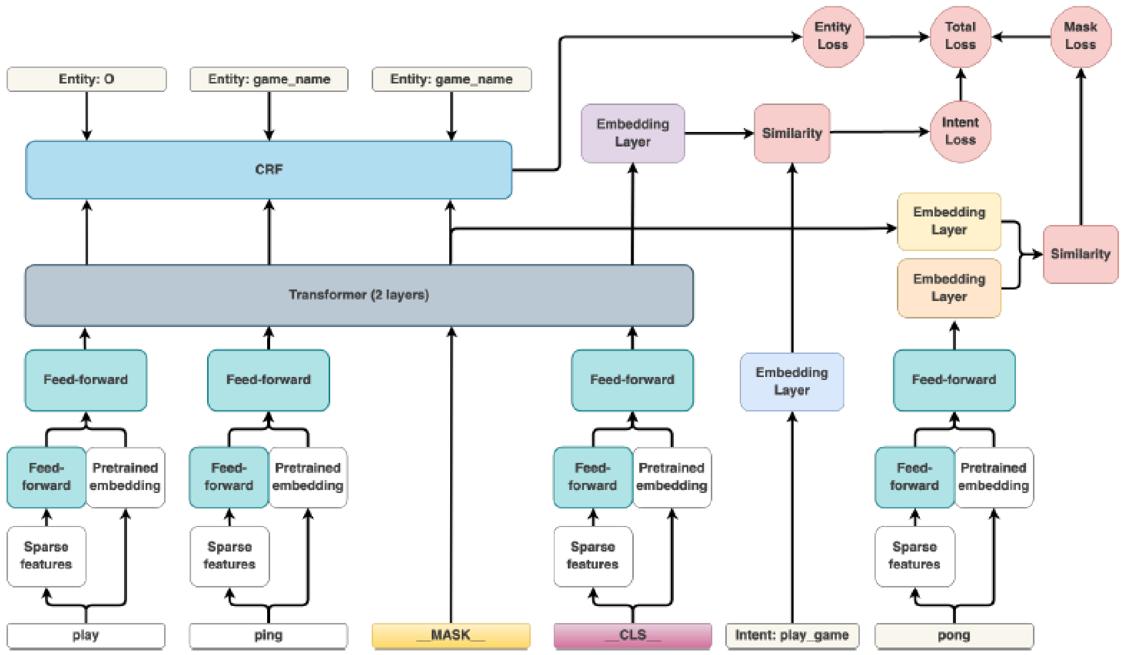

Figure 2.5: The Dual Intent and Entity Transformer (DIET) architecture that is at the core of Rasa and utilized in the implementation of SousChef. [5]

2.7.1.1 Main components

Featurization

Each input sentence is separated in tokens which can be either words or sub-words. A special specification token $\_\_CLS\_\_$ is also added at the end of each sentence. Sparse and/or dense features are generated for each input token. Sparse features include token-level one-hot encodings and multi-hot encodings of character n-grams ($n \leq 5$) which can contain a lot of unnecessary information. For this reason, dropout is applied by randomly setting a portion of the elements to zero in order to avoid overfitting to noisy data and improve the model's ability for generalization. Dense features include pre-trained word embeddings. ConveRT [27], BERT [28] and GloVe [29] embeddings are supported. In the case of ConveRT embeddings, the initial embedding of the $\_\_CLS\_\_$ token is set as the sentence encoding of the input sentence obtained from ConveRT, so as to add contextual information for the whole sentence. In the case of BERT, it is set to the respective output embedding of the BERT $[CLS]$ token. Finally in the case of GloVe, it is set to the mean of the embeddings of all the tokens contained in the input sentence.

All sparse features pass through a feed-forward layer. All sequence steps have shared weights in order to match the dimension of the dense features.

Transformer layer

The output of the previous step is concatenated with the dense features and it is inserted into another feed-forward layer. All sequence steps have shared weights in order to match the dimension of the transformer layer that follows. Later on, it is passed through a two-layer transformer with relative position attention.

Named entity recognition

A Conditional Random Field (CRF) tagging layer is responsible for classifying which tokens belong to certain entities. The transformer output sequence α passes as an input sequence of tokens through the CRF layer which predicts a sequence of entity labels y_{entity}.

$$L_E = L_{CRF}(\alpha, y_{entity}), \tag{2.1}$$

where $L_{CRF}(.)$ denotes negative log-likelihood for a CRF.

Intent classification

The $\_\_CLS\_\_$ token after passing through the transformer layers as well as the intent labels y_{intent} are embedded into a single semantic vector space $h_{CLS} = E(\alpha_{CLS}), h_{intent} = E(y_{intent})$, where $h \in R^{20}$. The dot-product loss is used to minimize the similarity $S_I^+ = h_{CLS}^T h_{intent}^+$ with the target label y_{intent}^+ and minimize similarities $S_I^- = h_{CLS}^T h_{intent}^-$ with negative samples y_{intent}^-

$$L_I = -\left\langle S_I^+ - log\left(e^{s_I^+} + \sum_{\Omega_I^-} e^{s_I^-}\right)\right\rangle, \tag{2.2}$$

where the sum is taken over the set of negative samples Ω_I^- and the average $\langle . \rangle$ is taken over all examples.

The dot product similarity is used as a ranker when trying to infer the correct intent label.

Masking

Apart from predicting the intent and entities of the input sentence, another training objective of the model is to predict randomly selected masked input tokens. 15% of the input tokens of the sentence are randomly masked. In 70% of the cases the masked token is substituted with the vector which corresponds to the mask token $\_\_MASK\_\_$, in 10% of the cases it is substituted with the vector corresponding to a random token and for the remaining cases the original input is maintained. This passes through the transformer layers and the output a-mask for each selected token y_{token} is fed through a dot-product loss.

$$L_M = -\left\langle S_M^+ - log\left(e^{s_M^+} + \sum_{\Omega_M^-} e^{s_M^-}\right)\right\rangle, \tag{2.3}$$

where $S_M^+ = h_{MASK}^T h_{token}^+$ is the similarity with the target label y_{token}^+ and $S_M^- = h_{MASk}^T h_{token}^-$ are the similarities with negative samples y_{token}^-. h_{MASK} = $E(\alpha_{MASK})$ and $h_{token} = E(y_{token})$ are the corresponding vectors $h \in R^{20}$. The sum is computed over the set of negative samples Ω_M^- and the average $\langle . \rangle$ is taken over all examples.

Masking is used as a regularization technique as well as in order to train a model that learns a better language representation of the custom domain.

Total loss

The total loss is computed as follows:

$$L_{total} = L_I + L_E + L_M \tag{2.4}$$

The model is trained in a multi-task manner trying to minimize the total loss L_{total}. The architecture is configurable according to the desired task and the user can turn off any of the above losses that contribute to the sum.

Batching

In order to tackle any potential class imbalance, a balance batching strategy is used. Another regularization technique utilized is increasing batch size throughout training.

2.7.2 DIET performance

DIET manages to achieve robust results both for intent classification and entity recognition. It manages to outperform other state-of-the-art systems, such as Joint BERT [30], a joint intent classification and slot filling model that utilizes Bidirectional Encoder Representations from Transformers (BERT), and HERMIT architecture [31], a hierarchy of self-attention mechanisms and Bidirectional Long Short-Term Memory (BiLSTM) encoders followed by CRF tagging layers. DIET achieves competitive scores even when trained only on sparse features, without pre-trained embeddings, which require significantly less training time [5].

2.8 Background Summary

So far we have presented the background relevant to our work, by examining the architectures proposed for dialogue systems, as well as their main components. We have also introduced Rasa, the framework utilized for building the SousChef assistant. Finally, we have discussed in detail the DIET architecture that lies at the core our system's implementation. In the following chapter, we thoroughly discuss the methodology followed for building our assistant and present the key components of its architecture.

3. SOUS CHEF DIALOGUE SYSTEM

In this chapter, we mainly aim to describe the methodology followed to build the SousChef dialogue system. We try to provide a clear overview of the system architecture as well as a comprehensive understanding of some key architectural decisions.

3.1 Methodology

As briefly discussed in Chapter 1, the main goal of our master thesis is the development of a voice assistant that can handle procedural text and answer user questions based on it. Procedural text typically contains a set of instructions and steps needed to be followed in order to achieve an ultimate goal. Some examples of procedural text include recipes, science experiments and manuals. In our case, SousChef is a dialogue system that can parse cooking recipes and answer relevant user questions. The assistant can be used through a web application, where the user can provide the URL of the recipe they want to execute. After the recipe is parsed, the user can ask recipe-related questions using voice and get the relevant answers from the system in the same way.

For the implementation of the dialogue system we utilize Rasa framework and the majority of supporting scripts are written in Python [1]. For converting our dialogue system to a voice assistant, we have connected it to a set of microservices, implemented by Athena RC, which contain the ASR and TTS components. The whole pipeline of services, containing our system, runs in a Docker container.

As seen in Figure 3.1., at the beginning of the conversation the user is asked to provide a URL with their preferred recipe. After the recipe is provided the text that contains the recipe is extracted from the HTML code and some basic pre-processing is performed. The cleaned text is used as input to the model that has been trained for entity recognition for the custom recipe-related entities. When the entities have been extracted, an ontology is filled with the instances of the selected recipe. Each time the user asks a question, the natural language question is converted to a SPARQL query with the help of Rasa's actions and slots. After the ontology is queried, the answer is converted to natural language with the help of Rasa's actions and returned back to the user.

3.1.1 Recipe Ontology

Since linked data became an integral part of modern computer applications, the use of ontologies has become extensive. An ontology is a data model that is able to formally represent a domain by specifying a set of concepts and the relationships between them [32]. However, the need for formal languages responsible for querying these knowledge

[1] The relevant scripts can be found at GitHub at the following link: https://github.com/Eleni-Ts/rasa_project

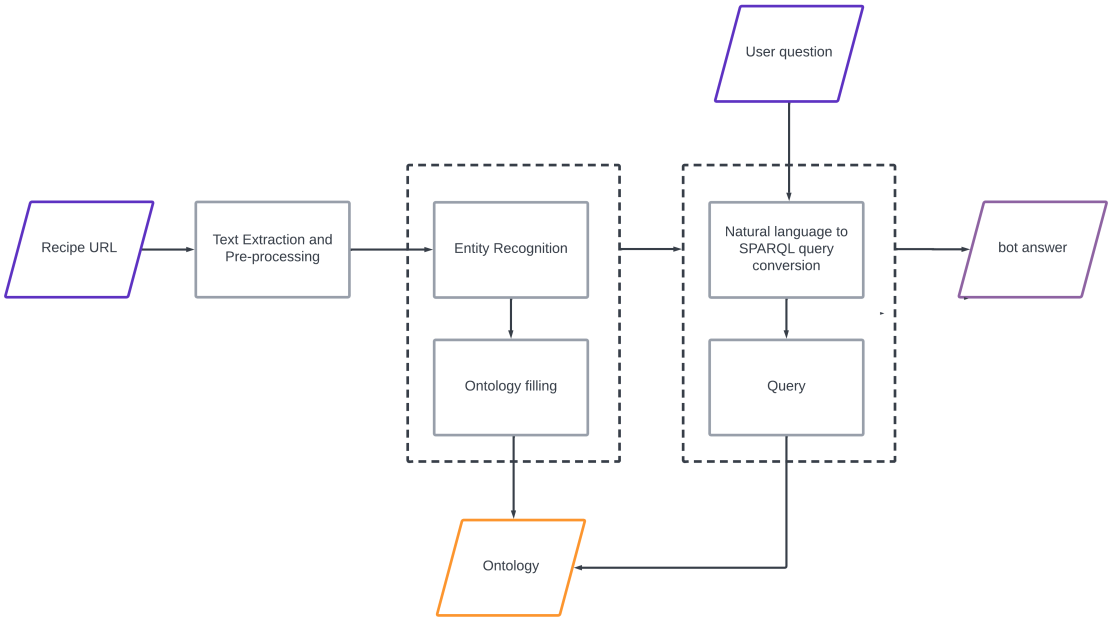

Figure 3.1: End-to-end pipeline of SousChef assistant. After the user provides a recipe URL, useful text is extracted from the HTML code and fed to the NER model. The extracted entities fill the recipe ontology, which is now ready to be queried. Each time the user makes a query, this is converted to SPARQL and is made on the ontology. The query answer is formatted in natural language and returned to the user.

bases emerged as well. SPARQL is such a popular formal query language that is used to query data stored in Resource Description Framework (RDF) format. An example of a SPARQL query can be seen in Figure 3.2

In our implementation, data are organized in RDF triples in order to build the recipe ontology. Triples are in the form of subject-relationship-object where subject and object represent ontology concepts. Protégé, an open-source ontology editor tool, was used in order to build the ontology.

The proposed ontology has been designed based on the common structure of online cooking recipes and in view of the type of expected questions to the assistant. A visualization of the recipe ontology is shown in Figure 3.3 .

The recipe ontology consists of the following Classes:

- $Recipe$: Class $Recipe$ has properties related to the recipe's execution time ($hasExecutionTime$), the recipe's preparation time ($hasPreparationTime$), the recipe's portions ($hasPortions$), the steps which the recipe consists of ($hasStep$) and the recipe instructions ($hasInstruction$).
- $Instruction$: Class $Instruction$ has properties corresponding to the order of the instruction in the recipe ($hasOrder$), the utensils needed for the instruction ($hasUtensil$) and the units found in the instruction ($hasUnit$).
- $Step$: Class $Step$ corresponds to the steps in which the recipe is divided and has a property related to the ingredients needed for a recipe step ($hasIngredient$).

```
PREFIX recipe: <http://www.semanticweb.org/eleni_tsili/recipe_ontology#>

SELECT ?mode_text
WHERE
{ ?ingr rdf:type recipe:Ingredient.
  ?ingr recipe:hasText ??.
  ?ingr recipe:hasMode ?mode.
  ?mode recipe:hasText ?mode_text
  }
```

Figure 3.2: Example of a SPARQL query that tries to find the mode of an ingredient from the information in the ontology. There are two main parts in the query: 1. the SELECT clause which identifies the variables to appear in the query results and 2. the WHERE clause which has the triple patterns.

- $Ingredient$: Class $Ingredient$ has properties related to the quantity of the ingredient needed ($hasQuantity$) and the details related to an ingredient ($hasMode$).
- $Utensil$: Class $Utensil$ corresponds to the utensils needed for the recipe.
- $Unit$: Class $Unit$ corresponds to the units of measure usually found in the ingredients list of a recipe.
- $Mode$: Class $Mode$ corresponds to the types of entities that provide details usually about the form of an ingredient.

Since users formulate questions in natural language, there is the need for converting natural language queries to SPARQL queries. To this end, when a user provides a recipe to the system, the ontology is filled with relevant instances extracted from it. Therefore, each time there is a recipe-related question, the ontology should be queried and the appropriate answer should be returned to the user. OWlread2 library is used for loading and manipulating the ontology. The detailed pipeline for SPARQL query formulation is discussed later in Section 3.2.

3.1.2 Named Entity Recognition model

After building the recipe ontology, a Named Entity Recognition (NER) model was trained to recognize the custom entities that could parse a recipe and fill the recipe ontology based on it. For this purpose, Rasa's NLU pipeline was utilized, the most important component of which is DIET classifier, presented in detail in Chapter 2.5.1. The training set for the NER model consisted of recipes retrieved from the internet and the model was trained on a number of custom entities that can be seen in Figure 3.4. Later on, the NER model was fine-tuned on additional dialogue-related training examples in order to form the basis of the system's NLU, by leveraging Rasa's incremental training. This process is described in section 3.2.

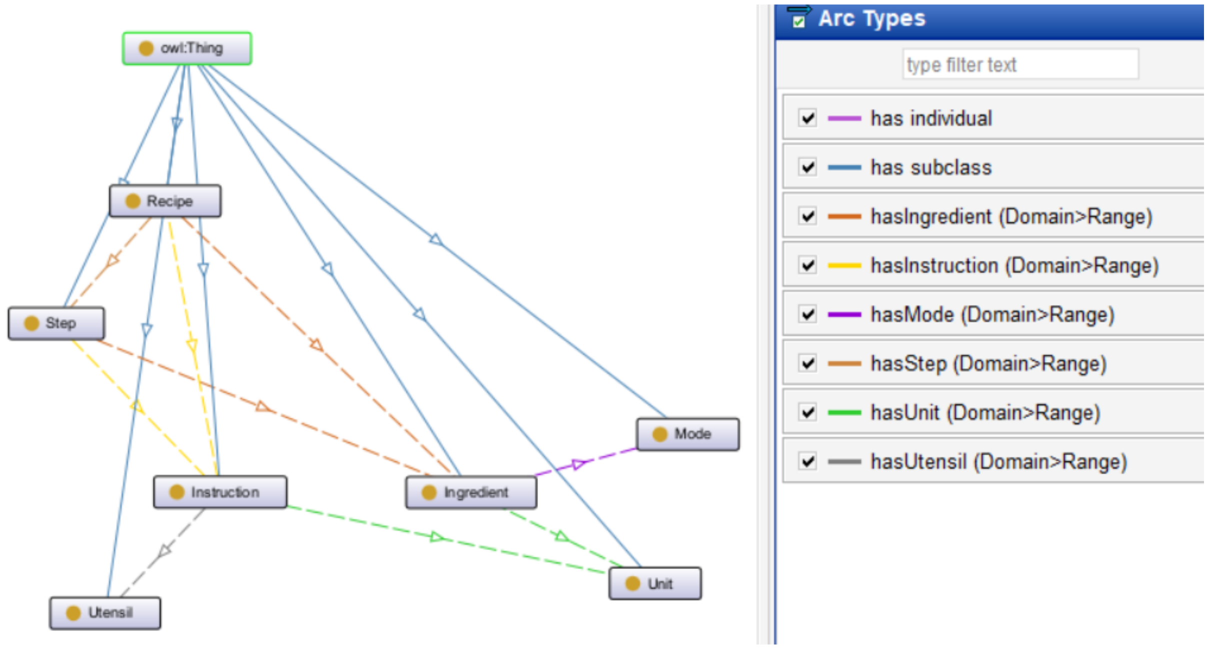

Figure 3.3: The recipe ontology. On the left side, there are the ontology classes (nodes) and how they are interconnected and on the right side there are the properties (edges) that connect the classes.

```yaml
entities:
  - ingredient
  - utensil
  - mode
  - step
  - quantifier
  - unit:
     groups:
     - time:
         roles:
         - sec
         - min
         - hour
     - amount
```

Figure 3.4: The entities the NER model was trained on.

Configuration	Sparse features	Dense features	Masking
Config_1	yes	yes (SpaCy)	-
Config_1_mask	yes	yes (SpaCy)	yes
Config_2	-	yes (Greek Bert)	-
Config_2_mask	-	yes (Greek Bert)	yes
Config_3	yes	-	-
Config_3_mask	yes	-	yes

Table 3.1: The different configurations the DIET classifier was trained on, for the task of entity recognition.

Dataset

For the creation of the dataset, a set of 30 recipes in Greek was retrieved from https://www.giorgostsoulis.com/. The recipe-related text was extracted from the HTML code and some pre-processing was performed, including tokenization at sentence level, expansion of unit abbreviations (e.g., *gr* expands to *γραμμάριο* | *γραμμάρια*) and removal of duplicate training instances. The resulting dataset consists of 450 sentences and was manually annotated with the custom entities. Finally, the dataset was split into a training set consisting of 360 sentences and a test set consisting of 90 sentences, with an 80:20 ratio.

Configuration

For the entity recognition task, we decided to utilize Rasa's NLU pipeline as it is highly flexible and customizable. It consists of various components for pre-processing, intent classification, entity extraction and response selection which can be used in a plug-and-play manner depending on the preferred task. One of the most important components that Rasa has to offer is DIET classifier (Dual Intent and Entity Transformer) as it is a multi-task architecture that can perform both intent recognition and entity classification, it offers a modularized architecture by combining various dense and sparse features, and it is easily configurable from a single .yml file. The architecture's adaptability makes it a suitable candidate for various applications, as in our case to train a NER model for the recipe-related entities of our ontology. At this stage of the system's development, we focus on the entity recognition part of DIET classifier. The intent classification part was utilized later on when fine-tuning the initial model on the whole dialogue-related dataset for the NLU part of the dialogue system.

For the entity recognition model, various configurations for the Rasa NLU pipeline, both lighter and more complex ones, were used and evaluated in order to conclude to which is the one that best suits our case. Streamlit was used in order to easily compare each configuration results on the test set, based on F1-score, precision and recall. We have tried various configurations in the featurization step, by using both sparse and dense features as input to the DIET classifier.

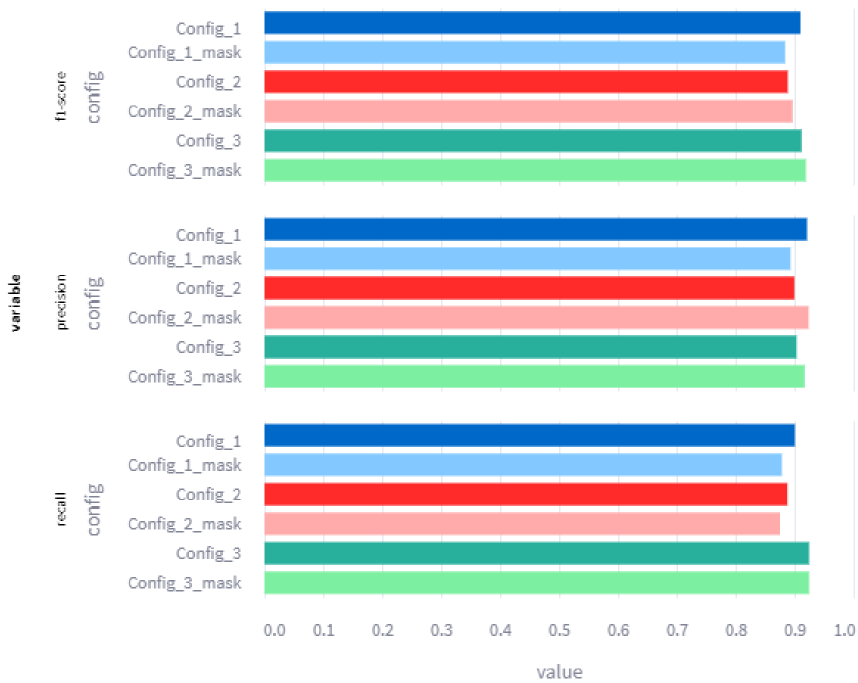

Figure 3.5: Results from the various configurations the NER model was trained on, in terms of F1-score, precision and recall. The best performing model is the one trained only on sparse features with masking activated.

In Config_1 we have selected to use sparse features together with SpaCy's pretrained embeddings for Greek as input to DIET, by utilizing SpaCy's medium model whose size is 40MB and it has been trained on news/media texts. For Config_2 DIET's input we utilized embeddings from Greek Bert, an edition of Google's BERT pre-trained language model for Greek. Finally, in Config_3 only sparse features, making it the lightest model to train. Additionally, the above models were tested with masking activated during DIET's training. All models were trained for 50 epochs. A diagrammatic representation of the aforementioned configurations can be seen in Table 3.1.

Results and Discussion

For the evaluation of the model we measure precision, recall and F1-score for each NER model. As seen in Figure 3.5, the DIET model trained only on sparse features with masking activated is the best performing model in terms of F1-score and recall, while the model trained on Greek Bert pre-trained embeddings with masking activated was the one that outperformed the rest in terms of precision. Since F1-score, is the weighted average of precision and recall, Config_3_mask was selected as the best performing model. We also favor F1-score over accuracy as it is a more suitable metric in cases where there is an imbalance in the distribution of classes, as in our case.

It is worth mentioning that the model that outperformed them all, was the one entailing the least degree of complexity as it was trained only on sparse features. The rest of the models were trained on pre-trained embeddings and thus required larger computational power to train as well as slower inference time, showcasing that DIET is indeed a suitable architecture for our application. Another important aspect to focus on, is that Config_3_mask outperformed the respective model that did not use the masking technique. That was indeed expected as masking is utilized mainly in order for the model to learn a better representation of the custom domain. A more in-depth analysis of the features used in the best-performing configuration is provided in Section 3.2.1.

3.2 System architecture

At the basis of our system's architecture lies the conversion of natural language to SPARQL in order to query the recipe ontology, with the help of Rasa Stack. The following subsections discuss the Rasa configurations utilized both for Rasa NLU and Rasa Core, as well as the pipeline for converting a user question to SPARQL.

3.2.1 Rasa NLU

The NLU part of our system is responsible for processing the user input in order to extract the intent of the user utterance, as well as to identify any entities included in it. Since we have already trained a model based on Rasa's NLU pipeline, that can pick up recipe-related entities, we decided to take advantage of Rasa's incremental training, which offers the option to fine-tune an existing model. The already trained DIET classifier was fine-tuned on some additional data related to the dialogue itself, this time by focusing both on intent recognition and entity extraction.

Dataset

The dataset for the NLU part consists of example user utterances which are categorized by intent and are stored in $nlu.yml$ file. Apart from the intents that are used when the conversation starts or ends, the rest of the intents have been manually generated based on what is expected from such an assistant to provide help with during the execution of a recipe. Each intent is assigned with multiple sentences that can vary in structure but all serve the same purpose. For instance, the intent $ask\_first\_instruction$ includes sentences like: ”Τι λέει να κάνω πρώτα;”, ”Πώς ξεκινάω τη συνταγή;” and ”Πες μου την πρώτη οδηγία”. A list of the intents our model was trained on can be found in Figure 3.6.

Apart from the intent categorization of user utterances, we also had to manually annotate any entities found in these utterances, as the DIET classifier would also be trained on entity extraction. This will play a significant role in the query formulation as it will be discussed in the following section.

```
intents:
  - recipe
  - greet
  - ask_ingredients
  - ask_utensils
  - ask_ingredient_amount
  - ask_first_instruction
  - ask_next_instruction
  - ask_previous_instruction
  - repeat_instruction
  - recipe_link
  - set_reminder
  - cancel_reminders
  - ask_time_of_recipe
  - ask_recipe_portions
  - ask_ingredient_mode
  - end_of_cooking
  - my_reminder
```

Figure 3.6: The intents the Rasa NLU was trained on.

Configuration

Since we fine-tuned the already trained classifier, we have used the configuration selected for the NER model. In our NLU pipeline, the user utterances first pass through the WhiteSpaceTokenizer, which as the name suggests, uses white space as the tokenization separator. Next is the featurization step where as described in the previous section we have chosen to include only sparse features. The featurizers selected for this step are the LexicalSyntacticFeaturizer and CountVectorsFeaturizer. The former moves with a sliding window over the tokens of a user utterance and extracts features for the previous, current, and next token. These include information about if the tokens are lower or upper case, if they start with an uppercase character, if they are at the beginning or end of the sentence, and if they contain just digits. The latter creates bag-of-words representations of user utterances by utilizing Sklearn's CountVectorizer and depending on the configuration it can extract word or character n-grams. In our case, we have selected to extract both as features. Finally, all extracted features are fed to the DIET classifier in order to fine-tune the model on the new instances.

3.2.2 Rasa Core

Rasa Core is mainly responsible for the conversation flow, the complete session management and the system's responses. The main components of Rasa Core are policies, which manage the selection of the next action to be executed, and stories, that serve as training data for the dialogue management part. Stories are representations of possible conversations between the user and the assistant, stored in $stories.yml$ file. User responses are declared as intents and bot responses as actions. Stories are designed to train a model that can also handle unseen conversation paths.

We have selected to use the default configuration for Rasa Core which consists of four different policies, both rule-based and machine-learning based. These are used in combination and the policy that makes the prediction with the highest score is selected. First, our configuration consists of MemoizationPolicy, which utilizes the stories provided during training in order to find one that matches the current conversation. If it manages to find a match, it predicts the next action, otherwise it returns None. The next policy in our configuration is RulePolicy, which predicts the next action based on rules, a type of training data that provide fixed conversation paths. Another configured policy is TEDPolicy, a multi-task architecture responsible both for predicting the next action and performing entity recognition, by taking into consideration any preceding context. Finally, our configuration includes UnexpecTEDIntentPolicy, an auxiliary policy which shares the same model architecture as TEDPolicy, and is responsible for handling unlikely user input.

3.2.3 Natural language to SPARQL conversion pipeline

This section includes a description of the main components of the pipeline that converts a user question to a SPARQL query. An example of such a transformation can be viewed in Figure 3.7.

Text extraction and pre-processing

At the beginning of the conversation flow, the bot asks the user to provide the URL of the recipe they want to execute. The website supported is https://www.giorgostsoulis.com/. After the user provides the recipe link, Rasa triggers an action that retrieves the recipe-related text from the HTML code and performs some basic pre-processing before this is inserted in the entity recognition model for annotation. Text cleaning includes sentence level tokenization, expansion of units, removal of some extra punctuation and removal of extra white space and tab characters. The name of the recipe is also stored. Finally, the part of the text that contains the list of ingredients is separated from the part that contains the instructions.

Entity recognition and Ontology filling

Following the recipe pre-processing, the NER model that has been trained for the custom recipe-related entities is imported. The recipe text, both the ingredients' and the instructions' part, is used as input to the model. After the model annotates the text with the respective entities, the output is exported to a .json file. Finally, the tagged entities are used to fill the recipe ontology with the respective instances. The ontology is now complete and ready to be queried.

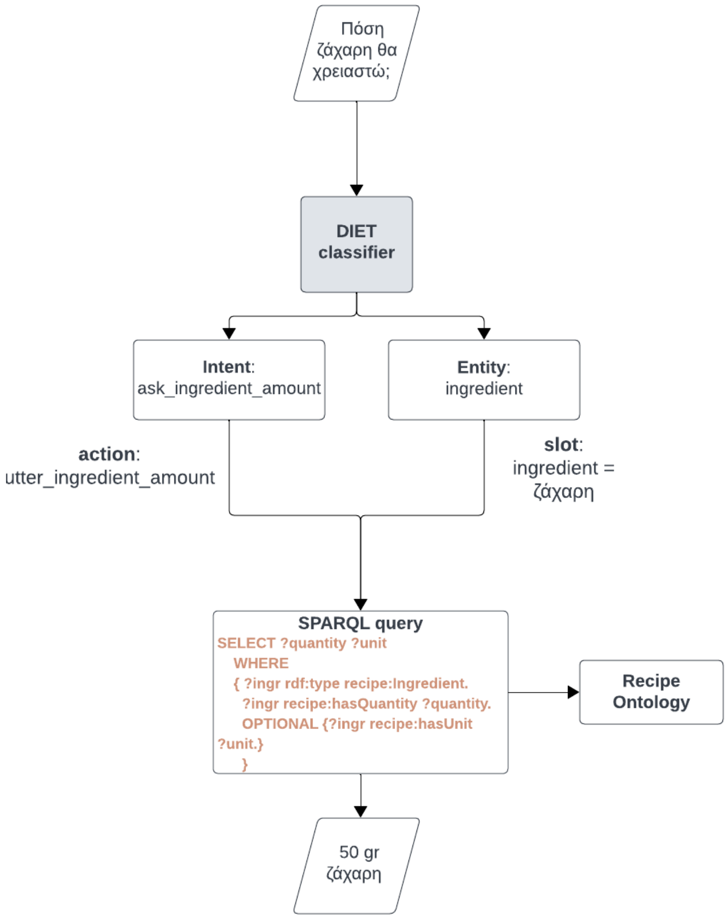

Figure 3.7: Example of the transformation of a natural language query to a SPARQL query, by utilizing intents and entities extracted by the DIET classifier in order to trigger a SPARQL query with the relevant parameters. This pipeline is described in detail in Section 3.2.

Slots

Rasa framework offers the possibility to store information in slots, which are key-value stores of information that either the user provides or that is retrieved from external source (e.g., the result of a database query). They can be viewed as the system's long memory in the conversation. Slots in our case are very useful as they can store entities picked up by the DIET model from user questions and store them for later use in the query formulation. For example, the slot $ingredient$ stores the respective entity when this is found in a user utterance. However, slots can also store any other type of information the system will need later during the conversation. For instance, the slot $current\_instruction$ serves as a tracker of the current step as it stores the index of the instruction the user is at.

Actions

Each time the user utters a message, Rasa Core decides what the next system action should be. In their simplest form, actions are pre-defined responses that will be sent back to the user depending on the input. However, Rasa offers the opportunity to create custom actions, which are functions that can contain any code the developer needs, such as to make API calls or to query a database. In our approach, we use both types of actions. Simple response actions contain those that are used to greet the user at the beginning and end of the conversation, such as $utter\_greeting$, or to ask what the user would like to cook. Some examples of more complex custom actions are those that query the recipe ontology, such as $action\_utter\_ingredients$ or $action\_utter\_time\_of\_recipe$, which query the recipe ontology according the user's question and return an answer. Another complex custom action is $action\_prepare\_recipe$, which dynamically parses the the HTML code of the recipe the user provides a link for, extracts the useful text and loads it to the NLU model for entity prediction. Finally, based on the results it fills the recipe ontology with the extracted instances.

SPARQL query conversion

Each time the user asks a question, the DIET classifier that has been fine-tuned to perform both entity recognition and intent classification, annotates the question with an intent as well as it annotates any entities found in this intent, in order to fill the respective Rasa slots. Based on the recognized intent, the respective Rasa action is triggered which in turn corresponds to a SPARQL query. The query parameters are retrieved from the slots filled. For example, as seen in Figure 3.7 , the user question ”Πόση ζάχαρη θα χρειαστώ;” is annotated by DIET classifier with the intent $ask\_ingredient\_amount$ and the entity $ingredient$. The aforementioned intent triggers the action $utter\_ingredient\_amount$ which is responsible for executing a query that looks for the amount of a specific ingredient in the recipe ontology. The parameter of the respective query is retrieved from the slot $ingredient$ which has been filled with the extracted entity. Before the query is executed the string of the extracted entity is compared with all the ingredients found in the filled recipe ontology, using

Figure 3.8: The web application through which users can interact with the SousChef assistant.

FuzzyWuzzy library[2] that utilizes Levenshtein distance to find the closest match. If a match is indeed found, the SPARQL query is executed. After we retrieve its result, a response is formatted and returned to the user in natural language once again with the help of Rasa actions.

3.3 Speech Recognition and Speech Synthesis

After the dialogue system was trained, it was integrated to a pipeline of Docker-based microservices that were able to convert it to a voice assistant and connected to a web client that can be seen in Figure. The aforementioned framework made it possible for users to use the assistant through an easy-to-use web application. The UI of the application can be seen in Figure 3.8. For the Speech Recognition and Speech Synthesis part, Google's Speech-to-Text and Text-to-Speech APIs were utilized accordingly. Both of these services are supplied by Google Cloud and make use of robust neural network models that have been trained on large amounts of data.

Each time the user utters a phrase, this is recorded and sent from the client to the Speech-to-Text API that converts it to a text transcription. The transcription of the user input is then forwarded to Rasa which is responsible for returning the appropriate answer as a string. The selected answer is dispatched to the Text-to-Speech API which transforms it to synthetic human speech and returns the answer to an MP3 file. Finally, the audio file is sent back to the client and the user receives an answer in natural language.

3.4 Methodology Summary

In the above chapter, we discussed the methodology followed for the design and implementation of SousChef, a voice assistant for cooking. We presented the recipe ontology that we have built and concepts that it covers. Later on, we provided a detailed description

[2]https://pypi.org/project/fuzzywuzzy/

of the processed followed to train an NER model based on Rasa's DIET classifier and how we selected the most suitable configuration for our case. Furthermore, we extensively described the architecture of our dialogue system, at the basis of which lies a pipeline that converts natural language queries to SPARQL, and the components utilized to converted it to a voice assistant. In the following chapter, we conduct a thorough evaluation of the system and provide the results and relevant discussion

4. EVALUATION OF THE DIALOGUE SYSTEM

In this chapter, we discuss the process we have followed in order to evaluate our dialogue system by examining: a. the system's ability to recognize intents and entities in user utterances, b. its efficacy and efficiency and c. the quality of dialogue. We also present the results of the evaluation in detail and try to analyze and discuss various points of interest that arise from them.

Our system falls into the category of task-oriented dialogue systems, since its main goal is to assist users in the execution of a task, namely to provide the necessary recipe-related information that a user needs during cooking. The evaluation of such systems is an intricate task that needs to take into consideration various aspects. The first issues we want to explore throughout the evaluation process is the assistant's performance on intent classification and entity recognition in user utterance, as well as how effective and efficient our assistant is in the completion of tasks. Furthermore, we attempt to measure our system's dialogue quality. For this purpose, we have relied on two evaluation approaches, automatic and human evaluation. During the first type of evaluation, we deploy various metrics to test the system, since these can offer an objective assessment of our assistant. However, since this is designed to converse with humans, we consider automatic evaluation alone insufficient to capture the more complex and refined elements of natural language that only real users can offer insights on. Thus, we have also selected to have human evaluators to provide feedback on the system.

4.1 Automatic evaluation

4.1.1 Intent classification and entity recognition

For the first part of the automatic evaluation, we focus on assessing the system's ability to correctly recognize intents and entities. We have already evaluated the NLU model that was trained on a number of annotated recipes, however at this point we proceed with the evaluation of the NLU model that was fine-tuned on the dialogue-related data. Since we need to deal with a rather limited data size, we have decided to use a 5-fold cross-validation approach to test our model, instead of using a single train-test split. Therefore, our data is split in 5 groups, meaning that at each fold the model was trained on 4 of them and tested on the remaining group. At each iteration, precision, recall and F1-score are calculated. The average of those metrics are the final scores achieved by the model. By using cross-validation we manage to conduct a fair and less biased evaluation of our model, as it is tested on the whole available dataset, instead of a small, single test set.

As seen it can be seen in Table 4.1, the NLU model achieves very high scores in all three metrics both in identifying intents and entities, so we can conclude that is has a very satisfactory performance on unseen data. However, to have a more thorough understanding of how the model achieves on these tasks, we identified the most common

Automatic evaluation		
	Metrics	**Scores**
Phase 1	Intent precision	0.983
	Intent recall	0.981
	Intent F1-score	0.984
	Entity precision	0.952
	Entity recall	0.918
	Entity F1-score	0.921
Phase 2	Task Completion Rate	82.07%
	Task Completion Cost	1.3
	Edit distance	0.18

Table 4.1: Results from automatic evaluation. In phase 1, we tested the system's performance on intent recognition and entity extraction. In phase 2, we evaluated the system's ability to complete tasks (Task Completion Rate), the average dialogue turns it needs to complete them (Task Completion Cost) and we measured how similar system dialogues are to the respective ground truth dialogues (Edit distance).

errors of the model from Rasa logs. In terms of intent recognition, we notice that the intents that confuses the system the most are $ask\_first\_instruction$, $repeat\_instruction$ and $ask\_first\_instruction$, probably because they are similar in structure and content. With regard to entity recognition, the most problematic case is that it misses to recognize words that belong to entity $ingredient$ and $step$.

4.1.2 Efficacy and efficiency

During the second phase of the automatic evaluation, we attempt to evaluate how well and how quickly our assistant can complete a task. For this purpose, we use Task Completion Rate and Task Completion Cost metrics. The former measures the system's ability to perform a task and it is computed as the rate of successful events in all task completion attempts. The latter measures how efficiently the system performs by computing the number of dialogue turns required until the completion of a task.

The end goal in our assistant's case is for the user to have completed a recipe. However, in a simulated evaluation setting, it is not easily measurable if the user has completed the task or how quickly this was achieved. The turns of the dialogue are subject to how each user chooses to use the system. For instance, a user might be familiar with a recipe and ask only a few questions while another might ask many questions until its completion. In order to have a clearer overview of our system's performance we considered that a dialogue can consist of many tasks, which include:

- providing the ingredients for the correct step, this can either be the whole recipe or a specific step of the recipe
- providing the utensils needed for the recipe

B: 15 λεπτά για την προετοιμασία και 2 ώρες 30 λεπτά για την εκτέλεση. Task 4: Failed

U: πόσο σκόρδο

B: Θα χρειαστείς:150 ml κονιάκ. Task 3: Successful (1 turn)

U: και πόση ποσότητα κονιάκ θέλω

Figure 4.1: Part of a user dialogue (1). The dialogue flow is from bottom to top, as the user (U) asks a question and the bot (B) provides the answer. This example is from the annotation process during the automatic evaluation.

- providing the quantity of an ingredient
- providing the correct step of the recipe
- providing the time needed for the recipe
- providing the portions of the recipe
- providing how a specific ingredient should be

The dialogues used for the calculation of the aforementioned metrics were retrieved by our own interaction with the system, as well as from the interaction of other users. At each dialogue, we tried to vary the questions both in terms of structure and intent, so that we have a full overview of the assistant's performance. We have collected a number of 30 dialogues and we have manually annotated the responses of the system as correct or not, based on the recipe provided. The average number of tasks each dialogue contained is 9. Some examples of annotated dialogues can be seen in Figures 4.1 and 4.2. The results from the second phase of the automatic evaluation can be viewed in Table 4.1.

Task Completion Rate

We calculate Task Completion Rate as the ratio of completed tasks over the total number of tasks. Our system achieves a score of 82.07% in the aforementioned metric, which means that it executes the majority of tasks. An important observation we have made during the annotation process was that the ASR part was accountable for many of the failed tasks. The NLU part of the assistant managed to handle some mistaken outputs of the ASR part and still returned the correct answer. However, there are a number of cases where the ASR misinterpretations lead to task failure. Another common case of task failure seems to be when a user asks for the utensils of the recipe, since there have been observed a few cases were the system either returns only a part of recipe utensils or returns false positives. This is probably due to the sparsity of training examples for this entity that the NLU model was trained on.

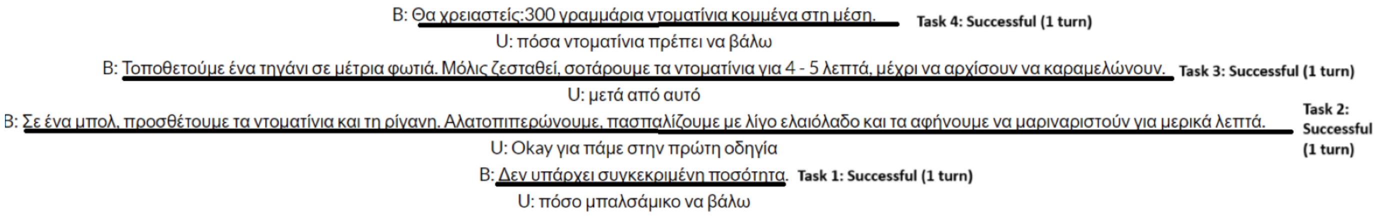

Figure 4.2: Part of a user dialogue (2). The dialogue flow is from bottom to top, as the user (U) asks a question and the bot (B) provides the answer. This example is from the annotation process during the automatic evaluation.

Task Completion Cost

We calculate Task Completion Cost as the average number of dialogue turns the system needs until the completion of a task. Since the ideal case of a dialogue for our system would be to provide an answer at a single turn, without repetition of questions from the user, the ideal Task Completion Cost is considered 1. Our system managed to score 1.3 showing that it is very efficient. Of course, the results for Task Completion Cost should be viewed in parallel to the results from Task Completion Rate, as the former was calculated based only on successfully completed tasks.

Edit distance

Finally, we decided that it would also be useful to evaluate the dialogue as a whole in order to check whether the system managed to successfully complete the end goal which is the recipe execution. To this end, we asked from 5 users to interact with the system until they reach the end of the recipe. We then represented each dialogue as sequence based on the number of turns. For instance, the representation of a ground truth dialogue with 4 turns would be $< t_1 t_2 t_3 t_4 >$. In cases where the system asks the user to repeat the question or fails to execute a task, the turn is represented as x. Therefore, if a dialogue contains a repetition it will be represented as $< t_1 t_2 x t_3 t_4 >$. Each dialogue was then compared to its ground truth dialogue, by calculating the edit distance according to the following metric:

$$Error = (subsitutions + deletions + 0.4 * insertions)/ground\_truth\_turns) \tag{4.1}$$

where substitutions are cases when the system fails to complete a task, deletions are the cases where it does not return any answer and insertions are cases where it asks the user to repeat. In order to favor cases where the system asks the user to repeat compared to the other two cases, we multiply it by a correction factor 0.4.

The average error from the above calculation for all dialogues was 0.18, indicating that the actual dialogue users had with the system was quite efficient as most of the dialogue turns were successful.

4.2 Human evaluation

During human-based evaluation we try to measure our assistant's dialogue quality based on scores provided by 10 human evaluators that interacted with the system. The questions provided to the users try to evaluate the dialogue in terms of relevance, appropriateness, informativeness and fluency.

Users were asked to interact with the assistant by using the web UI[1] and subsequently fill in a questionnaire with some questions about their experience. They were asked to select a recipe of their choice from the supported website and they were advised to ask questions until they complete the whole recipe, in an attempt to simulate real-world conditions. Following their communication with the assistant, they completed the questionnaire where they had to rate how relevant the assistant's answers were to their questions, how appropriate they find the system's answers for this type of dialogue, if they have received the expected answers from the system and to rate the level of naturalness of the dialogue. The possible answers were the following: 'very', 'good enough', 'a little', 'not at all'.

4.3 Evaluation Insights and Discussion

The user evaluation provided a number of useful insights not only on the dialogue quality, but also on the overall user experience. The appropriateness of the dialogue seemed to be on quite satisfactory levels, as 6 users provided the highest score and the rest answered that the answers were appropriate enough for this type of dialogue. In terms of fluency, users seemed to have the most unanimous opinion, with 9 out of 10 stating that the system's answers sounded natural enough. The categories of relevance and informativeness seem to be the most debatable ones. However, for both categories 6 users rated the questions they got from the system as 'good enough', with regard to how relevant to their question they were and to whether they got the expected information. For both categories, three users provided the rating 'very' and one user provided the rating 'a little'. Overall, the above scores showcase that users were quite content with the quality of dialogue they conducted with the assistant.

Furthermore, regarding the users' overall interaction with the system we made several useful observations. An initial observation was that some of the users seemed to have a difficulty in using the interface provided and follow the steps on how to use the assistant. Although they were provided with instructions beforehand, they were confused on how to start the dialogue or when to provide the link. This led us to the conclusion that maybe the steps of greeting the assistant and providing the link should be removed from the course of the dialogue. An easier alternative for the user would be to provide the link when opening the web application and then start asking questions at their own pace.

Another valuable insight was that sometimes users asked follow-up questions on one of the already mentioned entities, but without stating the entity directly. This is the linguistic

[1] https://apps.ilsp.gr/basic-va/voice/

B: κομμένα κάθετα σε λεπτές φέτες.
U: Πώς είπες ότι πρέπει να είναι κομμένο το κρεμμύδι
B: Δεν κατάλαβα ποιο υλικό εννοείς. Μπορείς να επαναλάβεις;
U: Πώς είπες ότι πρέπει να είναι κομμένο
B: Θα χρειαστείς: 1 κιλό κρεμμύδια κομμένα κάθετα σε λεπτές φέτες.
U: πόσα κρεμμύδια να βάλω

Figure 4.3: Part of a user's interaction with the assistant, where we notice that the assistant fails at the execution of one of the tasks as the user doesn't refer directly to a concept but uses anaphora.

phenomenon of anaphora, where a word or a phrase refers to a previously stated concept. An example of such a case, can be seen in Figure 4.3 where a user, after receiving the required amount of the ingredient 'onions', wants to know how these should be cut, without specifying the ingredient. The system was not able to correctly identify the intents of those questions and was asking the user to repeat.

Other useful aspects of the user evaluation were to notice the different variations in terms of structure and vocabulary that each human evaluator used in order to express the same intent. The assistant managed to effectively handle the majority of those variations, although it had difficulty understanding the most descriptive ones. Some of the questions were also expressed as yes or no questions which the system could not support.

Overall, although evaluating dialogue systems' performance is a complex task, we have tried to perform a comprehensive evaluation of our assistant, by including both an automatic and a human evaluation. Since our system could also be viewed as an information retrieval system, BLEU score would be another valuable evaluation metric. However, this would also require an additional annotation process in order to compare the human answers to the assistant' s. Furthermore, an in-field evaluation in order to test the systems performance in the conditions it was designed to be used would provide valuable insights. This is of course, very difficult to be achieved as it would require a full deployment of the system and some necessary alterations so it could be used in such a noisy environment as a kitchen. It would also be very difficult to recruit a number of users to participate in such a time-consuming evaluation.

5. CONCLUSIONS AND FUTURE WORK

Over the course of our master thesis, we managed to implement SousChef, an end-to-end voice assistant that can help a user during cooking by answering recipe-related questions. The main point of interest regarding the architecture of our system lies in the natural language to SPARQL conversion pipeline that forms its basis. In order to support this query conversion, we have designed a Recipe ontology and trained an NER model for custom recipe-related entities on a dataset of Greek recipes, using Rasa's NLU pipeline. Then, we fine-tuned this initial model on dialogue-related data and with the help of Rasa Stack we achieved to implement our assistant. In order, to support interactions through voice, we connected the system to a set of microservices that use Google's Speech-to-Text and Text-to-Speech APIs as well as a web client.

After SousChef was set up, we tried to conduct a comprehensive evaluation, both automatic and human-based. During the automatic evaluation we first tested the system's performance in intent recognition and entity extraction by calculating F1-score, precision and recall. We also evaluated the assistant's ability to perform an efficient and effective dialogue, using Task Completion Rate and Task Completion Cost metrics, and we compared user dialogues with ground truth dialogues using edit distance metric. Finally, we conducted a human evaluation in order to assess the dialogue's quality in terms of relevance, appropriateness, informativeness and fluency. The results from the overall evaluation showed that SousChef is capable of conducting effective and efficient dialogue while also achieving satisfactory scores in dialogue quality.

As an initial step for future work, we consider useful to fine-tune the ASR model on task-related speech in order to achieve better results in the speech recognition part, which widely influences the rest of the dialogue system pipeline.

Another aspect for future development would be the extension of the ontology. It could be expanded with more entities and more complex relations between entities. In this way, the system would be able to support additional questions.

Finally, an alternative approach that we consider useful is to view the overall task as a question-answering problem. In this case, a QA model could be fine-tuned on a number of recipes. This would allow a wider variety of recipe-related questions, especially for the instructions part. However, this would add a significant workload in terms of data gathering as there is no available annotated dataset for Greek recipes.

ABBREVIATIONS - ACRONYMS

AI	Artificial Intelligence
AIML	Artificial Intelligence Markup Language
API	API Application Programming Interface
ASR	Automatic Speech recognition
BERT	Bidirectional Encoder Representations from Transformers
BiLSTM	Bidirectional Long Short-Term Memory
CRF	Conditional Random Field
DIET	Dual Intent and Entity Transformer
DST	Dialogue State tracking
HTML	HyperText Markup Language
NER	NER Named Entity Recognition
NLG	NLG Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
TTS	TTS Text-to-Speech
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

REFERENCES

[1] M. T. Zemčík, “A brief history of chatbots,” *DEStech Transactions on Computer Science and Engineering*, vol. 10, 2019.

[2] J. Ni, T. Young, V. Pandelea, F. Xue, V. V. Adiga, and E. Cambria, “Recent advances in deep learning based dialogue systems: a systematic survey,” *Artificial Intelligence Review*, vol. 56, pp. 3055–3155, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:234342691

[3] S. Mehri, T. Srinivasan, and M. Eskenazi, “Structured fusion networks for dialog,” in *Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue*, S. Nakamura, M. Gasic, I. Zuckerman, G. Skantze, M. Nakano, A. Papangelis, S. Ultes, and K. Yoshino, Eds. Stockholm, Sweden: Association for Computational Linguistics, Sep. 2019, pp. 165–177. [Online]. Available: https://aclanthology.org/W19-5921

[4] R. Sharma, “An analytical study and review of open source chatbot framework, rasa,” *International Journal of Engineering Research and*, vol. V9, 06 2020.

[5] T. Bunk, D. Varshneya, V. Vlasov, and A. Nichol, “Diet: Lightweight language understanding for dialogue systems,” 04 2020.

[6] I. Turing, “Computing machinery and intelligence-am turing,” *Mind*, vol. 59, no. 236, p. 433, 2007.

[7] K. Ramesh, S. Ravishankaran, A. Joshi, and K. Chandrasekaran, “A survey of design techniques for conversational agents,” in *International conference on information, communication and computing technology*. Springer, 2017, pp. 336–350.

[8] H. Chen, X. Liu, D. Yin, and J. Tang, “A survey on dialogue systems: Recent advances and new frontiers,” *SIGKDD Explor. Newsl.*, vol. 19, no. 2, p. 25–35, nov 2017. [Online]. Available: https://doi.org/10.1145/3166054.3166058

[9] J. Weizenbaum, “Eliza—a computer program for the study of natural language communication between man and machine,” *Commun. ACM*, vol. 9, no. 1, p. 36–45, jan 1966. [Online]. Available: https://doi.org/10.1145/365153.365168

[10] H.-Y. Shum, X. He, and D. Li, “From eliza to xiaoice: Challenges and opportunities with social chatbots,” 2018.

[11] D. Jurafsky and J. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 02 2008, vol. 2.

[12] K. Colby, S. Weber, and F. Hilf, “Artificial paranoia,” *Artificial Intelligence*, vol. 2, pp. 1–25, 03 1971.

[13] G. Neff, “Talking to bots: Symbiotic agency and the case of tay,” *International Journal of Communication*, 2016.

[14] R. S. Wallace, *The anatomy of ALICE*. Springer, 2009.

[15] B. Abushawar and E. Atwell, “Alice chatbot: Trials and outputs,” *Computación y Sistemas*, vol. 19, 12 2015.

[16] M. Hoy, “Alexa, siri, cortana, and more: An introduction to voice assistants,” *Medical Reference Services Quarterly*, vol. 37, pp. 81–88, 01 2018.

[17] “80+ industry specific voice search statistics for 2023,” 2023. [Online]. Available: https://www.synup.com/voice-search-statistics

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

[19] F. Filippidou and L. Moussiades, “A benchmarking of ibm, google and wit automatic speech recognition systems,” in *Artificial Intelligence Applications and Innovations: 16th IFIP WG 12.5 International Conference, AIAI 2020, Neos Marmaras, Greece, June 5–7, 2020, Proceedings, Part I 16*. Springer, 2020, pp. 73–82.

[20] M. Saxon, S. Choudhary, J. P. McKenna, and A. Mouchtaris, “End-to-end spoken language understanding for generalized voice assistants,” *arXiv preprint arXiv:2106.09009*, 2021.

[21] Y. Ning, S. He, Z. Wu, C. Xing, and L.-J. Zhang, “A review of deep learning based speech synthesis,” *Applied Sciences*, vol. 9, no. 19, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/19/4050

[22] A. Quamar, C. Lei, D. Miller, F. Ozcan, J. Kreulen, R. J. Moore, and V. Efthymiou, “An ontology-based conversation system for knowledge bases,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 361–376.

[23] A. Sattar, E. S. M. Surin, M. N. Ahmad, M. Ahmad, and A. K. Mahmood, “Comparative analysis of methodologies for domain ontology development: A systematic review,” *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 5, 2020.

[24] D. S. Mishra, A. Agarwal, B. Swathi, and K. C. Akshay, “Natural language query formalization to sparql for querying knowledge bases using rasa,” *Progress in Artificial Intelligence*, vol. 11, no. 3, pp. 193–206, 2022.

[25] D. Altinok, “An ontology-based dialogue management system for banking and finance dialogue systems,” *arXiv preprint arXiv:1804.04838*, 2018.

[26] J. Deriu, A. Rodrigo, A. Otegi, G. Echegoyen, S. Rosset, E. Agirre, and M. Cieliebak, “Survey on evaluation methods for dialogue systems,” *Artificial Intelligence Review*, vol. 54, pp. 755–810, 2021.

[27] M. Henderson, I. Casanueva, N. Mrkšić, P.-H. Su, T.-H. Wen, and I. Vulić, “Convert: Efficient and accurate conversational representations from transformers,” *arXiv preprint arXiv:1911.03688*, 2019.

[28] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” in *North American Chapter of the Association for Computational Linguistics*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:52967399

[29] J. Pennington, R. Socher, and C. Manning, “GloVe: Global vectors for word representation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, A. Moschitti, B. Pang, and W. Daelemans, Eds. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. [Online]. Available: https://aclanthology.org/D14-1162

[30] Q. Chen, Z. Zhuo, and W. Wang, “Bert for joint intent classification and slot filling,” *arXiv preprint arXiv:1902.10909*, 2019.

[31] A. Vanzo, E. Bastianelli, and O. Lemon, “Hierarchical multi-task natural language understanding for cross-domain conversational ai: Hermit nlu,” *arXiv preprint arXiv:1910.00912*, 2019.

[32] A. H. Asiaee, T. Minning, P. Doshi, and R. L. Tarleton, “A framework for ontology-based question answering with application to parasite immunology,” *Journal of biomedical semantics*, vol. 6, pp. 1–25, 2015.