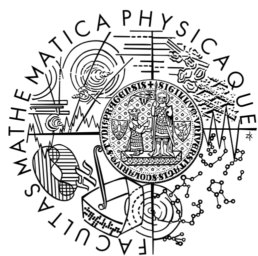

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Jaroslav Šafář

Practical neural dialogue management using pretrained language models

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: Mgr. et Mgr. Ondřej Dušek, Ph.D.
Study programme: Computer Science (N1801)
Study branch: IUI (1801T036)

Prague 2023

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In .... date ....

Author's signature

I would like to express my gratitude to my supervisor, Mgr. et Mgr. Ondřej Dušek, Ph.D., for his invaluable guidance, unwavering support, and constructive feedback throughout the development of this thesis. Furthermore, I would like to extend my thanks to my family for their continuous support and unwavering encouragement during my years of studies. Their belief in my abilities has been a constant source of motivation. I am deeply appreciative of their support and understanding, as without them, this thesis would not have been possible.

Title: Practical neural dialogue management using pretrained language models

Author: Bc. Jaroslav Šafář

Institute: Institute of Formal and Applied Linguistics

Supervisor: Mgr. et Mgr. Ondřej Dušek, Ph.D., Institute of Formal and Applied Linguistics

Abstract: Task-oriented dialogue systems pose a significant challenge due to their complexity and the need to handle components such as language understanding, state tracking, action selection, and language generation. In this work, we explore the improvements in dialogue management using pretrained language models. We propose three models that incorporate pretrained language models, aiming to provide a practical approach to designing dialogue systems capable of effectively addressing the language understanding, state tracking, and action selection tasks. Our dialogue state tracking model achieves a joint goal accuracy of 74%. We also identify limitations in handling complex or multi-step user requests in the action selection task. This research underscores the potential of pretrained language models in dialogue management while highlighting areas for further improvement.

Keywords: dialogue systems, pretrained language models, natural language processing, dialogue management

Název práce: Praktický neuronový dialogový manažer s použitím předtrénovaných jazykových modelů

Autor: Bc. Jaroslav Šafář

Ústav: Ústav formální a aplikované lingvistiky

Vedoucí diplomové práce: Mgr. et Mgr. Ondřej Dušek, Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt: Dialogové systémy zaměřené na úkoly představují výzvu vzhledem ke své složitosti a potřebě zvládnout komponenty, jako porozumění jazyku, sledování stavu, výběr akcí a generování jazyka. V této práci zkoumáme zlepšení řízení dialogu pomocí předtrénovaných jazykových modelů. Představujeme tři modely, postevané na předtrénovaných jazykových modelech, jejichž cílem je poskytnout praktický přístup k návrhu dialogových systémů schopných efektivně řešit porozumění jazyku, sledování stavu a úlohu výběru akcí. Náš model pro sledování stavu dialogu dosahuje přesnosti 74%. V úloze pro výběr akcí identifikujeme problémy ve zpracování složitých nebo vícekrokových uživatelských požadavků. Tento výzkum podtrhuje potenciál předtrénovaných jazykových modelů v dialogovém managementu a zároveň ukazuje na oblasti pro další zlepšení.

Klíčová slova: dialogové systémy, předtrénované jazykové modely, zpracování přirozeného jazyka, dialogový manažer

Contents

Introduction **4**

1 Theoretical Background **6**
- 1.1 Dialogue Systems . . . 6
  - 1.1.1 Chatbots and Task-oriented Dialogue Systems . . . 6
  - 1.1.2 Task-oriented Dialogue Systems and Their Traditional Architecture . . . 7
- 1.2 Language Modeling . . . 9
  - 1.2.1 Language Models . . . 10
  - 1.2.2 N-grams and n-gram Language Models . . . 10
  - 1.2.3 Estimating n-grams Probabilities . . . 11
  - 1.2.4 Text Generation Using Language Models . . . 11
- 1.3 Deep Neural Networks . . . 12
  - 1.3.1 Understanding Deep Neural Networks . . . 12
  - 1.3.2 Building Blocks of DNNs . . . 13
  - 1.3.3 Training Deep Neural Networks . . . 15
  - 1.3.4 Feed-forward and Convolutional Neural Networks . . . 16
  - 1.3.5 Recurrent Neural Networks . . . 17
  - 1.3.6 Sequence-to-Sequence Architecture . . . 18
- 1.4 Transformers . . . 19
  - 1.4.1 Transformer Encoder-Decoder Structure . . . 19
  - 1.4.2 Input, Output, and the Embeddings . . . 19
  - 1.4.3 Positional Encoding . . . 20
  - 1.4.4 Multi-Head Attention . . . 21
  - 1.4.5 Feed-Forward Networks, Residual Connections, and Layer Normalization . . . 22
- 1.5 Pretrained Language Models . . . 23
  - 1.5.1 Transfer Learning in NLP . . . 23
  - 1.5.2 Pretrained Language Models and Pretraining Methods . . . 23
  - 1.5.3 Fine-tuning Large Language Models . . . 24
  - 1.5.4 Influential Large Language Models . . . 24

2 Dialogue Management and Related Work **26**
- 2.1 Early Dialogue Management . . . 26
- 2.2 Modern Approaches to Dialogue Management . . . 27
  - 2.2.1 Dialogue State Tracking . . . 27
  - 2.2.2 Action Selection / Dialogue Policy . . . 30
  - 2.2.3 End-to-End Dialogue Systems . . . 32

3 Practical Dialogue Management **34**
- 3.1 Dialogue State Tracking . . . 34
  - 3.1.1 Theoretical Description of Generative DST . . . 35
  - 3.1.2 Dialogue State and the String Representation . . . 35
  - 3.1.3 The Input and Output Strings . . . 38
- 3.2 Action Selection . . . 39

3.2.1 Theoretical Description of Generative Action Selection . . 39
3.2.2 Theoretical Description of Classification-Based Action Selection 40
3.2.3 Database and the String Representation 41
3.2.4 Action and String Representation 42
3.2.5 The Input and Output for Action Selection 45

4 Experiments 46
4.1 The MultiWOZ Dataset 46
4.1.1 MutliWOZ 46
4.1.2 MultiWOZ 2.1 46
4.1.3 MutliWOZ 2.2 47
4.1.4 Train Dataset and its Subsets 47
4.1.5 Dialogue State Ontology 48
4.1.6 Supported Actions 49
4.2 Dialogue State Tracking Metrics 49
4.2.1 Domain Level Metrics 49
4.2.2 Slot Level Metrics 50
4.2.3 Global Slot Level Metrics 50
4.2.4 Joint Goal Accuracy 51
4.3 Action Selection Metrics 51
4.3.1 Action Level Metrics 51
4.3.2 Turn Level Accuracy 51
4.3.3 Macro Averaged Metrics 52
4.3.4 Weighted Averaged Metrics 52
4.4 Model Training Details 52

5 Results and Discussion 54
5.1 Dialogue State Tracking Results 54
5.2 Action Selection Results 54
5.3 Manual Analysis 56

6 Conclusion 58

Bibliography 60

List of Figures 69

List of Tables 70

List of Abbreviations 71

List of Glossaries 72

A Attachments 75
A.1 Action support 75
A.2 Dialogue Analysis Example 80
A.3 Source Code Description and Usage 83
A.3.1 evaluate_pipeline.py 84
A.3.2 main_action_classification.py 85

A.3.3 main_action_generation.py 85
A.3.4 main_state_update.py 85

Introduction

Dialogue systems have evolved as one of the most significant advancements in natural language processing (NLP), offering the capacity for humans to communicate with machines in a natural, conversational manner. Particularly, task-oriented dialogue systems, designed to assist users in completing specific tasks, are being integrated into various fields such as customer service, personal assistance, and e-commerce (McTear, 2021).

Modeling dialogue is complex and typically divided into several components, such as language understanding, state tracking, action selection, and response generation. These components work together to ensure that the system understands user input, maintains the conversation context, decides on its next action, and generates an appropriate response. One of the main challenges of practical dialogue systems is managing these tasks effectively, especially in resource-limited scenarios.

Recently, architectures such as Hybrid Code Networks (Williams et al., 2017) have demonstrated promising results in practical settings but do not leverage the benefits of pretrained language models (Radford et al., 2018, 2019; Devlin et al., 2019; Liu et al., 2019; Raffel et al., 2020), which have dramatically transformed the landscape of NLP by showing impressive performance across a range of tasks. On the other hand, utilizing pretrained language models in the end-to-end dialogue system approaches such as Sequicity (Lei et al., 2018), MintL (Lin et al., 2020), Soloist (Peng et al., 2021), and AuGPT (Kulhánek et al., 2021) gives great results but often requires vast amounts of annotated data, which can be challenging to acquire. Moreover, these models tend to hallucinate and generate ungrounded natural language outputs, making them unreliable for practical applications (Ji et al., 2023).

Therefore, in practice, dialogue systems are still typically composed of individual modules: natural language understanding (NLU), dialogue management (DM), and natural language generation (NLG). Motivated by this, our thesis explores potential improvements in DM methods using pretrained language models. The focus is to provide a practical approach for designing new dialogue systems which can effectively address language understanding, state tracking, and action selection tasks, even under limited data conditions. We propose three models that handle NLU combined with dialogue state tracking and action selection tasks, demonstrating the potential of using pretrained language models in dialogue management. All models were evaluated in limited data setting. Dialogue state tracking model achieves a joint goal accuracy of 74%. Action selection scores are not high but the predicted actions are usually reasonable. We also note the limitations of the action selection model in their ability to handle complex or multi-step user requests.

The thesis is organized as follows. In Chapter 1, we provide a theoretical background on dialogue systems, language modeling, deep neural networks, transformers, and pretrained transformer-based language models. This knowledge is used in the subsequent chapters. Chapter 2 delves into the early development and modern approaches of dialogue management, concentrating on dialogue state tracking, action selection, and end-to-end dialogue systems. Chapter 3, our main

contribution, focuses on our proposed architecture for practical dialogue management, explaining in detail how each component works. Chapter 4 describes the experiments to evaluate our models, detailing the dataset used, the evaluation metrics, and training specifics. In Chapter 5, we discuss the results and implications of our models. Lastly, we summarize our findings, highlight the limitations of our approach, and suggest possible directions for future work in Chapter 6.

1. Theoretical Background

This chapter provides a theoretical foundation for understanding dialogue systems, language modeling, and deep neural networks. It begins with an introduction to dialogue systems (Section 1.1), distinguishing between chatbots and task-oriented dialogue systems and describing their traditional architecture, before moving to the foundation of language modeling (Section 1.2), including language models, n-grams, and their application in text generation. The following chapter dives deep into deep neural networks (Section 1.3), exploring their structure, training methods, and different architectures.

The chapter further describes in detail the Transformer architecture (Section 1.4) which revolutionized the field of natural language processing, concluding with a description of pretrained Transformer-based language models (Section 1.5) with emphasis on their pretraining methods, fine-tuning, and the influential models.

1.1 Dialogue Systems

A *dialogue system* is a computer program that can communicate with users in a natural language (McTear, 2021, p. 11). Communication is usually text-based, spoken, or multimodal. Even though other modes, such as graphics, haptics, or gestures, can be a part of communication, especially in humans, most dialogue systems use text or speech. Moreover, speech recognition (Yu and Deng, 2014) and speech synthesis (Wang et al., 2017) can be used to translate speech-to-text and text-to-speech, which allows us to work primarily with conversations in the form of text, which will also be the focus of this thesis.

1.1.1 Chatbots and Task-oriented Dialogue Systems

Dialogue systems are usually categorized into task-oriented dialogue systems and non-task-oriented dialogue systems. *Task-oriented dialogue systems* are purpose-built to perform a specific function. In these systems, the user and the system communicate to accomplish tasks, such as assisting with scheduling appointments, booking flights, or making restaurant reservations (Chen et al., 2017). Conversations with these task-oriented systems are generally more structured and primarily focused on obtaining the information needed to complete the task for which the given system is made (Rudnicky et al., 1999). For instance, a task-oriented dialogue system designed for booking flights might ask for the destination, date, and number of passengers. These systems are well known because digital assistants such as Siri, Google Home, and Alexa are examples of multi-purpose task-oriented dialogue systems (Williams et al., 2016).

On the other hand, *non-task-oriented dialogue systems* (chatbots) primarily engage in general conversation with users. Usually, their objective is to simulate human conversation as best as possible and can lead conversations on various topics to keep users' attention and provide entertainment (Shawar and Atwell, 2007; Brandtzaeg and Følstad, 2017).

1.1.2 Task-oriented Dialogue Systems and Their Traditional Architecture

In task-oriented dialogue systems, the user and the system traditionally take turns interacting with each other (McTear, 2021, p. 44). Each turn in the conversation is a continuous block text (or speech) called utterance. An utterance can be a question, command, or just a statement. These turns come together to form a dialogue driven toward a specific goal. Each utterance is produced based on all the previous utterances from the beginning of the dialogue up to the previous one. We call these utterances a dialogue context. To illustrate these concepts, consider the following dialogue with a system designed to book flights:

1. The user's initial utterance: "I'd like to book a flight to Paris" initiates the dialogue, and also the dialogue context:

   ["I'd like to book a flight to Paris"]

2. The system's response might be: "Sure, I can help with that. When would you like to fly", which extends the context to include utterances from both turns:

   ["I'd like to book a flight to Paris",
   "Sure, I can help with that. When would you like to fly"]

3. The user's next utterance: "Next Wednesday" adds one more utterance to the dialogue context:

   ["I'd like to book a flight to Paris.",
   "Sure, I can help with that. When would you like to fly?",
   "Next Wednesday."]

4. The system uses the dialogue context to understand that the user wishes to fly to Paris next Wednesday.

5. This back-and-forth exchange of turns continues until the user's task is accomplished or the dialogue is otherwise concluded.

The traditional task-oriented dialogue system can be broken down into several modules.

Automatic Speech Recognition (ASR)

For spoken dialogue systems, the process begins with automatic speech recognition (ASR). ASR technology analyzes acoustic waves and converts these signals into written text (Yu and Deng, 2014).

Natural Language Understanding (NLU)

The next step is natural language understanding (NLU), which is responsible for parsing and interpreting the natural language text to determine its meaning. NLU accomplishes this by converting the input text into a structured semantic representation, typically in the form of dialogue act (McTear, 2021, p. 46). A dialogue act contains only the necessary information for a dialogue system by

encapsulating an utterance's semantic *intent* or purpose and its *slots*, which are specific information related to the intent. Slots usually have assigned *values*. For example, in the dialogue act `inform(food=Chinese, price=cheap)`, `inform` is the intent, and `food` and `price` are slots with values `Chinese` and `cheap` respectively.

Dialogue Management (DM)

Another essential part of the system is dialogue management (DM), which accepts the representation of the user utterance generated by NLU, interacts with the external knowledge base, such as the database, and decides what to do next (Brabra et al., 2022). Dialogue management consists of two components:

- dialogue state tracking (DST), also *belief state tracking*,
- action selection, also known as *dialogue policy*

DST keeps track of the conversation context, information, and user preferences collected during the dialogue. It achieves this by managing the so-called *dialogue state*, a dynamic representation of the dialogue history containing various elements such as slots and their associated values (extracted by the NLU), user preferences, and past system actions. As the conversation continues, the DST updates the dialogue state with the latest information extracted from user utterances allowing the system to accurately represent the user's needs, even across multiple turns.

After updating the dialogue state, the next critical task is action selection. This process chooses the system's response to the user's latest utterance, using the updated dialogue state's information. The selected actions should be the ones that best move the dialogue towards its goal, reflecting the user's needs and preferences as represented in the dialogue state. The actions are usually structured similarly to dialogue act, with an *intent* and potential *slots* with assigned *values*. The intent signifies the system's intended action, such as informing, requesting, or confirming something, while the slots capture specific information relevant to that action. For instance, if the intent is to `inform`, slots could represent categories of information the system aims to inform about, with values containing the actual information. The values are typically fetched from a backend database.

Natural Language Generation (NLG)

The output of the dialogue management is then passed to the natural language generation (NLG) module (McTear, 2021). This process transforms the system's actions into a natural-sounding text. Some systems might use simple templating methods, where predefined text templates are filled with information from the system action; others might employ advanced machine-learning techniques to generate more diverse and natural-sounding responses.

Speech synthesis (TTS)

Finally, in spoken dialogue systems, the generated text is converted back into speech through a process called speech synthesis (TTS). This process is especially crucial for dialogue systems that interact with users via voice, such as virtual

assistants. The goal of speech synthesis is to generate speech that sounds as natural and human-like as possible (Wang et al., 2017).

Example Let usconsider an example of a conversation with a system designed to book restaurants:

1. The user's initial utterance: "I want to find a cheap Chinese restaurant." initiates the dialogue.

2. If the case of a spoken dialogue system, the system's ASR module transcribes this utterance to text:

   "I want to find a cheap Chinese restaurant."

3. The system's NLU module recognizes the intent and extracts the slots and values, creating a dialogue act:

   ```
   inform(food=Chinese, price=cheap)
   ```

4. The dialogue state tracking updates the empty dialogue state with the information from the dialogue act. We represent the dialogue state symbolically as an assignment of values into slots for each domain. In this case:

   ```
   restaurant{food=Chinese, price=cheap}
   ```

5. In the action selection stage, the system uses its dialogue state and decides to search through the database for a suitable restaurant. Let's assume it found a restaurant called `Lucky Dragon` that matches the user's criteria, creating an action

   ```
   restaurant-inform(name=Lucky Dragon, food=Chinese,
   price=cheap)
   ```

6. From the generated action, the system constructs a response using the NLG module: "I found a cheap Chinese restaurant called Lucky Dragon."

7. Finally, in the case of the spoken dialogue system, theTTS generates the response: "I found a cheap Chinese restaurant called Lucky Dragon."

This dialogue continues until the user's task is accomplished or the dialogue is otherwise concluded.

1.2 Language Modeling

To understand a language, it is not enough to understand the meanings of individual words; we also need to understand the syntactic and semantic relationships between the words that form a sentence. One way to model these relationships is through *language modeling*, a crucial area of natural language processing (NLP) and dialogue systems (as we will see in Chapter 2) that develops probabilistic

models to predict the next word in a sentence given the preceding words. In this section, we will describe what language model is (Sections 1.2.1, 1.2.2, 1.2.3) and how are they used to generate language (Section 1.2.4)

1.2.1 Language Models

A language model (LM) is a type of probabilistic model that assigns the probability to a sequence of words, or generally tokens. In language modeling, tokens can be individual words, parts of words, or even single characters, representing the smallest processing units in the model. Formally, given a sequence of t tokens $x_{1:t} = (x_1, x_2, \ldots, x_t)$, a language model calculates the following probability:

$$P(x_{1:t}) = P(x_1, x_2, \ldots, x_t). \tag{1.1}$$

A goal of the language model is to assign a higher probability to correct or more likely sequences of tokens.

1.2.2 N-grams and n-gram Language Models

An n-gram is a contiguous sequence of n tokens. In the case of language models, n-grams play a crucial role in computing the conditional probability of a token given the preceding tokens.

An *n-gram model* computes the probability of the last token in an n-gram given the previous tokens. Formally, these models compute the conditional probability of the last token x_n in an n-gram $x_{1:n} = (x_1, x_2, \ldots, x_n)$ given the preceding $n-1$ tokens:

$$P(x_n|x_{1:n-1}) = P(x_n|x_1, \ldots, x_{n-1}), \tag{1.2}$$

where n is the order of the n-gram.

For example, in a bigram model ($n = 2$), the probability of a word would only depend on the previous word, i.e., $P(x_n|x_{n-1}) = P(x_2|x_1)$.

When computing a probability of the entire sequence of t tokens $x_{1:t}$ as in Equation 1.1, we can also use n-grams. Using the chain rule of probability, we can decompose this probability as follows:

$$P(x_{1:t}) = P(x_1)P(x_2|x_1)P(x_3|x_{1:2}) \ldots P(x_t|x_{1:t-1}) = \prod_{k=1}^{t} P(x_k|x_{1:k-1}), \tag{1.3}$$

However, this computation can become computationally infeasible with increasing t due to the massive number of potential token sequences. An approximation is commonly used based on the Markov assumption stating that a token's probability only depends on a limited number of previous tokens instead of the whole history. This is where n-grams come into play: in an n-gram model, each conditional probability of a k-th token given its preceding $k-1$ tokens is approximated to only depend on the previous $n-1$ tokens (where we assume that $k \geq n$, otherwise k is used), significantly reducing the computational complexity, formally:

$$P(x_k|x_{1:k-1}) \approx P(x_k|x_{k-n+1:k-1}) \tag{1.4}$$

where n is the order of the Markov model. The approximation becomes exact when $k = n$, which connects back to our initial definition of n-gram models. The

Markov assumption substantially reduces the computational complexity, making n-gram models a practical and widely used approach in language modeling

We can now apply the n-gram model approximation to the computation of a probability of the entire sequence of t tokens $x_{1:t}$ in Equation 1.3, where we first decompose the joint probability $P(x_{1:t})$ of the sequence using the chain rule and then applying n-gram approximation from Equation 1.4, obtaining:

$$P(x_{1:t}) = \prod_{i=1}^{t} P(x_i|x_{1:i-1}) \approx \prod_{k=1}^{t} P(x_k|x_{k-n+1:k-1}). \tag{1.5}$$

1.2.3 Estimating n-grams Probabilities

To estimate the model probabilities, we can use maximum likelihood estimation (MLE) by counting the occurrences of n-grams in a corpus. For example, the probability $P(x_k|x_{k-1})$ can be estimated as the count of the bigram (x_{k-1}, x_k) and divided by the count of the unigram x_{k-1}:

$$P(x_k|x_{k-1}) = \frac{C(x_{k-1}, x_k)}{C(x_{k-1})} \tag{1.6}$$

where $C(x_{k-1}, x_k)$ is the count of the bigram (x_{k-1}, x_k), and $C(x_{k-1})$ is the count of the unigram x_{k-1}.

This estimation, however, can be problematic when dealing with bigrams (or higher n-grams) that did not occur in the training data. To overcome this problem, several smoothing techniques, such as *Laplace smoothing* (Manning and Schütze, 1999), which assigns a small probability to unseen n-grams, or advanced *Kneser-Ney smoothing* (Kneser and Ney, 1995), can be applied.

1.2.4 Text Generation Using Language Models

One of the interesting applications of language models (LM) is their capability to generate text. Given a starting token or tokens, a language model can generate subsequent tokens most likely to follow. In the simplest case the text generation process involves iteratively selecting the most probable next token and appending it to the sequence, known as *greedy decoding*.

Consider an n-gram model. Given a starting sequence of k tokens $x_{1:k} = (x_1 \ldots, x_k)$, the model generates the next token x_{k+1} that maximizes the following conditional probability:

$$x_{k+1} = \arg\max_{x} P(x|x_{1:k}) \approx \arg\max_{x} P(x|x_{k-n+2:k}). \tag{1.7}$$

Once the token x_{k+1} is selected and appended to the sequence, the updated sequence serves as the starting sequence for generating the next token. This process is repeated until a specific condition is met, such as reaching a predefined length or generating special *end-of-text* token.

While this greedy approach is computationally efficient, it is deterministic and can lead to repetitive and predictable text. We can improve text generation by sampling from the probability distribution of the next token instead of taking the most probable one.

The quality of the generated text heavily depends on the quality of the language model itself. For n-gram models, the quality of generated text cannot capture long-range dependencies between words due to the Markov assumption. As such, other types of language models based on deep neural networks, such as recurrent neural networks (RNNs)(Section 1.3.5) or Transformers (Section 1.4), are often used for text generation, as they can capture longer dependencies.

1.3 Deep Neural Networks

In recent years, deep neural networks (DNNs) have gained significant popularity and have proven to be an effective method for a wide range of tasks in numerous fields, from image recognition in computer vision to speech recognition and language modeling in natural language processing (Goodfellow et al., 2016).

The rise of deep neural networks in recent years is closely tied to the advancement of computational power and the availability of large-scale datasets. However, the concept of neural networks dates back to 1943 with the McCulloch-Pitts neuron model (McCulloch and Pitts, 1943), the first-ever mathematical model of a biological neuron. However, due to the limited computing resources of their time and the need for more sufficient data, the applicability of their neuron model was initially limited. The rebirth of neural networks started in the late 1980s and early 1990s with the introduction of the backpropagation algorithm (Rumelhart et al., 1986). However,"deep learning" began to be associated with neural networks only around the mid-2010s, when techniques to train deeper networks effectively were developed.

1.3.1 Understanding Deep Neural Networks

Deep neural networks (DNNs) are essentially function approximators (Goodfellow et al., 2016, p. 168), which can learn to predict the corresponding outputs given a set of inputs. They aim to approximate some unknown function f^* , which maps an input $\boldsymbol{x}$ into an output $\boldsymbol{y}$:

$$\boldsymbol{y} = f^*(\boldsymbol{x}). \tag{1.8}$$

Then, a DNN constructs a mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ and learns the optimal values for the parameters $\boldsymbol{\theta}$ to approximate the function f^* as best as possible. The mapping is constructed by composing together many different inner functions f_i, with each function associated with a specific *layer* L_i of the network. These layers can be viewed as layers of artificial neurons, hence the name "neural network". Each i-th layer (or i-th function) in a DNN takes the output from the previous layers (functions), performs some transformation on them, and passes the result onto the following layers (functions).

This design of inputs and outputs is often represented as a directed acyclic graph (DAG), which describes how various functions are composed to create the full network function. Using this graph representation of the DNN, each inner function f_i (or layer of neurons L_i) is represented by a graph *node* v_i. Therefore, a DAG is the another representation[1] of the same concept of a DNN.

[1] In the context of deep neural networks, the terms inner *function*, *layer*, and graph *node* can

This specific design of a deep neural network as a directed acyclic graph offers several advantages, most notably due to the properties that come along with the topological ordering of the nodes in the graph, which means that all computations on which a particular node depends are guaranteed to be completed before the computation at the node itself is performed.

Nodes with no incoming edges are the *input* nodes, similarly, nodes without any outgoing edges are the *output* nodes. The length of the longest path from an input node to an output node in the DAG, which corresponds to the number of layers in the network, gives the depth to the model, from which the term "deep learning" arises (Goodfellow et al., 2016).

1.3.2 Building Blocks of DNNs

The main building blocks of a deep neural network's layers are its artificial *neurons*. Each neuron is a simple computational unit that takes several inputs, applies a function, and produces an output.

Single Neuron

Consider a neuron in a particular layer L_i. We denote the inputs to this neuron as a row vector $\boldsymbol{x}^T = [x_1, x_2, \ldots, x_n]$ and the corresponding weights as a column vector (or matrix of shape $(n \times 1)$) as

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

Each neuron computes the linear combination (weighted sum) of its inputs, adds a *bias* term b, and passes the result through an *activation function* φ to produce its output:

$$y = \varphi\left(\sum_{j=1}^{n} w_j x_j + b\right) = \varphi\left(\boldsymbol{x}^T \boldsymbol{w} + b\right). \tag{1.9}$$

Layer of neurons

A layer in a DNN is a collection of k neurons that each receive the same input $\boldsymbol{x}^T$ and perform the same operation described above but with different weights. If we represent the weights of all neurons in the layer as a matrix $\boldsymbol{W}$ of shape $(n \times k)$ and the biases as a row vector $\boldsymbol{b}^T = [b_1, b_2, \ldots, b_k]$ then the output of the whole layer L_i is a row vector $\boldsymbol{y}^T = [y_1, y_2, \ldots, y_k]$, calculated as follows:

$$\boldsymbol{y}^T = \varphi(\boldsymbol{x}^T \boldsymbol{W} + \boldsymbol{b}^T), \tag{1.10}$$

where the activation function φ is applied element-wise.

often be used interchangeably because a layer in the network can be considered as a function that transforms its input into output and a node in DAG can represent that function. Hence, a layer of a neural network, its inner function, and its graphical representation as a node essentially depict the same concept from different perspectives.

Activation functions

The choice of activation function φ in a layer depends on the layer's position in the network and the problem being solved. The activation function introduces non-linearity into the model, enabling the network to learn more complex functions.

- Commonly used activation functions for hidden layers include:
  - *ReLU* (Rectified Linear Unit) is the most widely used activation function in hidden layers. It squashes all negative inputs to zero and leaves positive inputs unchanged, formally:
  $$\mathrm{ReLU}(x) = \max(0, x). \tag{1.11}$$
  - The *tanh* (hyperbolic tangent) function is another activation function used in hidden layers, especially in recurrent neural networks. It squashes the output into the range between -1 and 1:
  $$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{1.12}$$
  This function is similar to the sigmoid function (see below), but its output is zero-centered.
- Activation functions for output layers are typically chosen based on the nature of the task being solved. The common choices are:
  - For regression tasks (predicting the real-valued output), typically, no activation function is used in the output layer, meaning it is an identity function:
  $$\mathrm{Id}(x) = x \tag{1.13}$$
  - For binary classification tasks, the *sigmoid* function maps input values into the range between 0 and 1, providing an output that can be interpreted as a probability:
  $$\mathrm{sigmoid}(x) = \frac{1}{1 + e^{-x}}. \tag{1.14}$$
  - For multi-class classification tasks, the *softmax* function is used, which generalizes the sigmoid function for multi-class problems. Given an input vector $\boldsymbol{x}$ of K elements, the softmax function is defined as:
  $$\mathrm{softmax}(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{1.15}$$
  for $i = 1, \ldots, K$. The softmax function outputs a vector representing a probability distribution over a list of K potential classes, i.e., the sum of the vector values is 1.

1.3.3 Training Deep Neural Networks

The goal of training a DNN $f(\boldsymbol{x}; \boldsymbol{\theta})$ is to find the parameters $\boldsymbol{\theta}$ that minimize a *loss function* $\mathcal{L}$, which measures the difference between the network's predictions and the true labels (Goodfellow et al., 2016). We often assume that our training data is a set of N samples $\mathcal{D} = \left\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, \boldsymbol{y}^{(N)})\right\}$ that are independently drawn from some unknown but fixed data generating probability distribution p_{data}, which corresponds to the unknown function f^*so that $\boldsymbol{y}^{(i)} = f^*(\boldsymbol{x}^{(i)})$.

The loss function is the expected value of the per-example loss function L taken across the p_{data}. In practice, this expectation is estimated by averaging over all training samples $\mathcal{D}$:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p_{data}}[L(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{\theta})] \approx \frac{1}{N}\sum_{i=1}^{N} L(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}|\boldsymbol{\theta}), \tag{1.16}$$

where $L(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}|\boldsymbol{\theta})$ is the loss of the i-th sample. The exact form of the per-example loss function L depends on the task:

- For regression tasks, the mean squared error (MSE) is usually used:

  $$L(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}|\boldsymbol{\theta}) = (\boldsymbol{y}^{(i)} - f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}))^2, \tag{1.17}$$

  The overall MSE loss across the dataset is:

  $$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{y}^{(i)} - f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}))^2, \tag{1.18}$$

- For binary classification, the binary cross-entropy loss:

  $$L(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}|\boldsymbol{\theta}) = -\boldsymbol{y}^{(i)}\log(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})) - (1 - \boldsymbol{y}^{(i)})\log(1 - f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})), \tag{1.19}$$

  The overall binary cross-entropy loss across the dataset is:

  $$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{y}^{(i)}\log(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})) + (1 - \boldsymbol{y}^{(i)})\log(1 - f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})), \tag{1.20}$$

- Finally, for multi-class classification tasks, the categorical cross-entropy loss is used. We denote the number of classes as K, and $y_k^{(i)}$ and $f_k(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$ as the true and predicted probability of the i-th sample belonging to class k, respectively. Then:

  $$L(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}|\boldsymbol{\theta}) = -\sum_{k=1}^{K} y_k^{(i)}\log(f_k(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})). \tag{1.21}$$

  $\boldsymbol{y}^{(i)}$ is a one-hot encoded vector where only one element is 1 (corresponding to the probability of the true class) and the rest are 0. Therefore, the sum over the K classes simplifies to a logarithm of the predicted probability for the true class:

  $$L(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}|\boldsymbol{\theta}) = -\log(f_{k^*}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})), \tag{1.22}$$

  where k^* is the true class for the i-th sample. Therefore, the overall Categorical Cross-Entropy loss across the dataset is:

  $$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}\log(f_{k^*}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})), \tag{1.23}$$

To find the parameters $\boldsymbol{\theta}$ that minimize $\mathcal{L}(\boldsymbol{\theta})$, *gradient descent* (Ruder, 2017), an iterative optimization algorithm for finding the minimum of a function, is used. To implement gradient descent, we first need to compute the gradient of the loss function $\nabla_\theta \mathcal{L}(\boldsymbol{\theta})$, i.e., the vector of its partial derivatives with respect to the parameters $\boldsymbol{\theta}$. The update rule in gradient descent is then given by:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_\theta \mathcal{L}(\boldsymbol{\theta}), \tag{1.24}$$

where α_i is the *learning rate*, a hyperparameter that determines the size of our steps. The learning rate can generally change across iterations or stay constant.

In practice, $\boldsymbol{\theta}$ is a high-dimensional vector, and calculating the exact gradient $\nabla_\theta \mathcal{L}(\boldsymbol{\theta})$ using the whole training dataset can be computationally expensive. Thus, we often use *stochastic gradient descent* (SGD) or its variant *mini-batch gradient descent*, which computes the gradient and updates the parameters based on a random subset of B examples of the training set called a *batch*. Moreover, improvements upon SGD, such as Adam (Kingma and Ba, 2017), and its extension AdamW (Loshchilov and Hutter, 2017) have been proposed to adaptively adjust the learning rate based on a historical gradient information.

We compute the gradients using the backpropagation algorithm (Rumelhart et al., 1986). Given the batch of training examples $\left\{\left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right)\right\}_{i=1}^{B}$, we perform a forward pass through the network to compute the outputs and the loss. We then perform a backward pass to compute the gradients of the loss with respect to the weights, using the chain rule of differentiation.

This iterative process of forward and backward passes continues until the loss function stops decreasing (convergence) or after a fixed number of iterations.

1.3.4 Feed-forward and Convolutional Neural Networks

The architecture of a deep neural network plays a significant role in determining the model's performance. Different architectures have been designed depending on the task and input data structure.

The most commonly used architecture in deep learning is the feed-forward neural network (FNN) (Goodfellow et al., 2016). As the name suggests, in this architecture, the computations are performed through the network only in a forward direction: from the input layer to the output layer, passing through any hidden layers in between without loops, following the DAG architecture discussed so far, which can be formally represented by:

$$\boldsymbol{h}_i = \varphi_i(\boldsymbol{W}_i \boldsymbol{h}_{i-1} + \boldsymbol{b}_i), \qquad \forall i \in \{1, 2, ..., L\} \tag{1.25}$$

where $\boldsymbol{h}_i$ is the output of the i-th layer, $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$ are the weight matrix and bias vector of the i-th layer, respectively, and φ_i is the activation function of the i-th layer.

In the computer vision field, another type of neural networks architecture, called convolutional neural networks (CNNs) (LeCun et al., 1998), have been used extensively. CNNs are designed to learn spatial hierarchies of features from the input images automatically. The core building block of a CNN is the *convolutional layer* performing a convolution operation on the input data, which can be formally

represented by:

$$\boldsymbol{h}_{i,j} = \varphi\left(\sum_{m,n} \boldsymbol{W}_{m,n} \cdot \boldsymbol{x}_{i+m,j+n} + b\right) \tag{1.26}$$

where $\boldsymbol{h}_{i,j}$ is the output of the convolution operation at location (i, j), $\boldsymbol{W}_{m,n}$ is the weight of the filter at location (m, n), and $\boldsymbol{x}_{i+m,j+n}$ is the input at location $(i + m, j + n)$. For instance, AlexNet (Krizhevsky et al., 2012) is an example of CNN designed for image classification tasks. It has greatly influenced the design of subsequent deep learning models for computer vision.

1.3.5 Recurrent Neural Networks

Recurent neural networks (RNNs) (Rumelhart et al., 1986; Elman, 1990) are a class of neural networks that are especially well-suited to prediction problems involving sequential data, such as natural language text, sound, or any time series data.

Unlike FNNs, which are represented by a DAG, RNNs introduce cycles in their network structure, essentially making the graph directed but cyclic. This fundamental difference allows RNNs to maintain an internal *hidden state*, or *memory*, enabling them to process sequences of inputs.

The computations in RNNs involve cycles, which can be formally represented by:

$$\boldsymbol{h}_t = \varphi(\boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}) \tag{1.27}$$

where $\boldsymbol{h}_t$ is the hidden state at time step t, $\boldsymbol{x}_t$ is the input at time step t, and $\boldsymbol{U}$ and $\boldsymbol{W}$ are the weight matrices for the hidden state and input, respectively. From this equation, we can see that the output of an RNN at any given time step depends not only on the input at that time step but also on the previous hidden state, therefore, on all the previous inputs processed by the network. The computation is visualized in Figure 1.1.

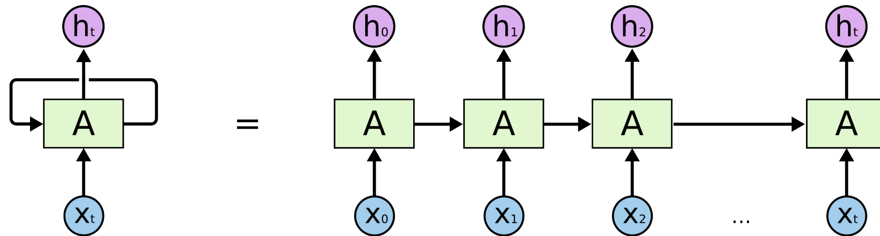

Figure 1.1: An unrolled recurrent neural network. Image source: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-unrolled.png`

However, standard RNNs suffer from a significant problem known as the vanishing gradient problem (Bengio et al., 1994), causing the gradients to drop to zero, which makes it hard for them to learn and tune the model parameters during the training process, especially when dealing with long sequences. This problem led to the development of more sophisticated types of RNNs such as the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and the Gated Recurrent Unit (GRU)(Cho et al., 2014), which are designed to capture longer dependencies in the input data. GRU is visualized in Figure 1.2.

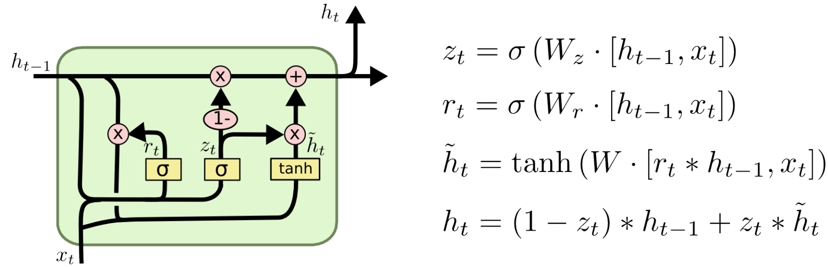

Figure 1.2: The Gated Recurrent Unit (GRU). It consists of multiple operations and gates as visualized in the image and described by equations. Image source: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-var-GRU.png`

1.3.6 Sequence-to-Sequence Architecture

The Sequence-to-sequence (Seq2Seq) or encoder-decoder architecture is designed for tasks where the input and output are both sequences, and their lengths may differ, such as machine translation, speech recognition, or text summarization. It was first introduced by Sutskever et al. (2014) and then improved by incorporating attention mechanisms (Bahdanau et al., 2015; Vaswani et al., 2017).

A Seq2Seq model consists of two RNNs: an encoder and a decoder. The encoder processes the input sequence and uses its last hidden state to represent the input as a context vector. The decoder uses this context vector to generate the output sequence. Mathematically, given an input sequence $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N)$ and an output sequence $\boldsymbol{y} = (y_1, y_2, ..., y_M)$, the model learns the conditional probability $P(\boldsymbol{y}|\boldsymbol{x})$ by using the encoder to map $\boldsymbol{x}$ into a fixed-length context vector $\boldsymbol{z}$ and the decoder to generate $\boldsymbol{y}$ token by token from the conditional probability $P(y_k|\boldsymbol{y_{1:k-1}}, \boldsymbol{z})$, where $\boldsymbol{y_{1:k-1}}$ are the previously predicted outputs. The first output token y_1 is a special *beginning-of-sequence* token $< \texttt{bos} >$, which is used to initialize decoding and to generate a second output token from $P(y_2| < \texttt{bos} >, \boldsymbol{z})$. This generated token is then appended to the output sequence used to generate another token. Tokens are generated until another special token, *end-of-sequence* $< \texttt{eos} >$, is generated. This generative process is called *autoregressive*.

During training, Seq2Seq models usually employ a technique known as *teacher forcing*. When generating k-th output token y_k, we use the actual output from the training dataset as the previously generated tokens $\boldsymbol{y_{1:k-1}}$ instead of using the model's predictions. However, at inference, the decoder cannot access the actual output and must generate it step-by-step in an autoregressive manner.

A critical limitation of the original Seq2Seq architecture using RNNs is that it encodes the entire input sequence into a single fixed-length context vector, which can lead to information loss, especially for longer sequences. An *attention mechanism* was introduced by Bahdanau et al. (2015) to overcome this limitation by allowing the decoder to refer (attend) back to any part of the input sequence when generating the output.

1.4 Transformers

Transformer models, introduced by Vaswani et al. (2017) offer an innovative approach to sequence processing. In contrast to the recurrent and convolutional architectures typically used for sequence processing, which rely on recurrent and convolutional operations to process data sequentially or to capture local dependencies, Transformers use different techniques. Instead, they rely entirely on attention mechanisms and feed-forward networks, allowing them to directly model dependencies between elements in the input sequence, regardless of their distance. This design enables Transformers to overcome the limitations of RNNs and CNNs, such as the difficulty capturing long-range dependencies and the necessity for sequential computation.

1.4.1 Transformer Encoder-Decoder Structure

The Transformer model, as described by (Vaswani et al., 2017), consists of an encoder and a decoder, each comprising multiple identical layers, as illustrated in Figure 1.3. The encoder and decoder share a similar high-level architecture, but their internal components differ slightly.

The *encoder* contains two main sub-layers: a multi-head self-attention and a position-wise fully connected feed-forward network. The input to each encoder layer first passes through the self-attention layer, utilizing dependencies between the input elements. Each element in the result is then fed into a feed-forward layer, independent of the rest. Each of these two sub-layers has a residual connection followed by layer normalization.

The *decoder* also contains two similar sub-layers with an additional encoder-decoder attention layer that uses the output of the encoder. The first sub-layer is a masked self-attention layer. This modified self-attention layer allows each element to attend only to the previous elements and themselves, preserving the auto-regressive property. The self-attention result is then combined with the encoder information in the encoder-decoder attention layer. Finally, the result is fed into a fully connected feed-forward network. Like in the encoder, each sub-layer in the decoder is also surrounded by a residual connection followed by layer normalization. In the following subsections, we will describe the individual parts of the Transformer architecture in more detail.

1.4.2 Input, Output, and the Embeddings

In Transformers, the input and output are sequences of tokens. We use a predefined vocabulary to represent each token by a unique integer identifier. These integer identifiers are then mapped to continuous vectors in a high-dimensional space from a trainable embedding matrix, resulting in the input and output embeddings. These embeddings capture the semantic properties of the tokens and are learned jointly with the rest of the model during training (Mikolov et al., 2013; Pennington et al., 2014).

Formally, given a vocabulary V of size $|V|$, the embeddings are represented by a matrix $\boldsymbol{E} \in \mathbb{R}^{|V| \times d}$, where d is the dimension of the embedding space. The rows of the matrix are the individual embeddings. To transform a token with identifier

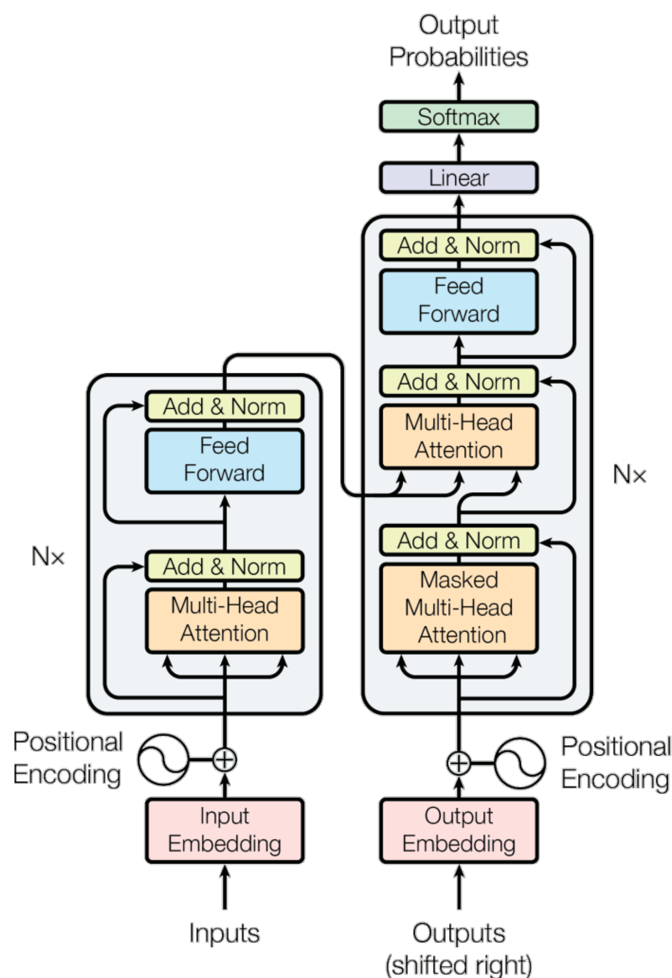

Figure 1.3: The Transformer architecture. The left part of the figure depicts the encoder. The right part is the decoder. Image source: Figure 1 of Vaswani et al. (2017).

id into its vector representation, we take the id-th row of the embedding matrix $\boldsymbol{E}$. We use the same embedding matrix for the encoder and decoder input tokens.

The output tokens of the Transformer decoder are usually produced by passing the outputs of the last decoder layer through another fully-connected linear layer to produce logits as vectors of the size $|V|$. The softmax activation function then transforms the logits into a probability distribution over the vocabulary. The output token is then selected from this distribution. The token with the highest probability is taken as the output during the training phase. During the inference phase, different strategies such as greedy decoding or nucleus sampling might be used (Holtzman et al., 2020).

1.4.3 Positional Encoding

While the self-attention mechanism in the Transformer model allows it to consider all tokens in the input sequence simultaneously, it does not consider the tokens' positions. To address this, Vaswani et al. (2017) used a positional encoding scheme to enable the model to consider the order of the tokens in the input sequence. The positional embeddings have the same dimension d as the token

embeddings, allowing them to be summed. For token position p and index of the positional embedding i, the positional encodings are defined as follows:

$$PE(p, 2i) = \sin\left(p \cdot 10000^{-2i/d}\right) \tag{1.28}$$

$$PE(p, 2i+1) = \cos\left(p \cdot 10000^{-2i/d}\right) \tag{1.29}$$

1.4.4 Multi-Head Attention

The *multi-head attention* layer allows the model to focus on different positions in the input sequence. It applies the following mechanism in parallel multiple times with different learned linear transformations, resulting in multiple attention *heads*. Formally, given an input matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, where each row represents one of the n input vectors of dimension d, we first transform $\boldsymbol{X}$ into matrices of keys $\boldsymbol{K}$, values $\boldsymbol{V}$, and queries $\boldsymbol{Q}$ for each head i using separate learned linear transformations:

$$\begin{aligned} \boldsymbol{Q}_i &= \boldsymbol{X}\boldsymbol{W}_i^Q, \\ \boldsymbol{K}_i &= \boldsymbol{X}\boldsymbol{W}_i^K, \\ \boldsymbol{V}_i &= \boldsymbol{X}\boldsymbol{W}_i^V, \end{aligned}$$

for $i = 1, ..., h$, where $\boldsymbol{W}_i^Q$, $\boldsymbol{W}_i^K$, and $\boldsymbol{W}_i^V$ are weight matrices for the i-th head of dimensions $d \times d_k$, $d \times d_k$, and $d \times d_v$, respectively. The attention for each head i (*Scaled Dot-Product Attention*), computes the query's dot product with all keys, divides them by $\sqrt{d_k}$, and applies a row-wise softmax function to obtain the weights for multiplying values in $\boldsymbol{V}_i$:

$$\boldsymbol{Z}_i = \text{softmax}\left(\frac{\boldsymbol{Q}_i \boldsymbol{K}_i^\top}{\sqrt{d_k}}\right)\boldsymbol{V}_i. \tag{1.30}$$

The scaling factor $\sqrt{d_k}$ is used to prevent the dot products from growing too large, which could cause the softmax function to have very small gradients and hamper learning (Vaswani et al., 2017). The resulting matrix $\boldsymbol{Z}_i$ for each head i has dimensions $n \times d_v$.

After computing the attention for all h heads, the results are concatenated into a matrix with shape $n \times h \cdot d_v$ and linearly transformed into the output of the attention layer:

$$\boldsymbol{Z} = \text{Concat}\left[\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_h\right]\boldsymbol{W}^O, \tag{1.31}$$

where $\boldsymbol{W}^O$ is a learned linear transformation of dimensions $h \cdot d_v \times d$. The resulting output $\boldsymbol{Z}$ has the same shape $n \times d$ as the input $\boldsymbol{X}$, enabling it to be used in subsequent layers. Figure 1.4 visualizes the whole attention mechanism.

Variants of the Attention in the Decoder

The Transformer decoder has two variants of the multi-head attention mechanism. One of them is the *masked multi-head attention* which is almost identical to the regular multi-head attention mechanism, with one key difference: it applies a mask to the input of the softmax function. This mask ensures that the future

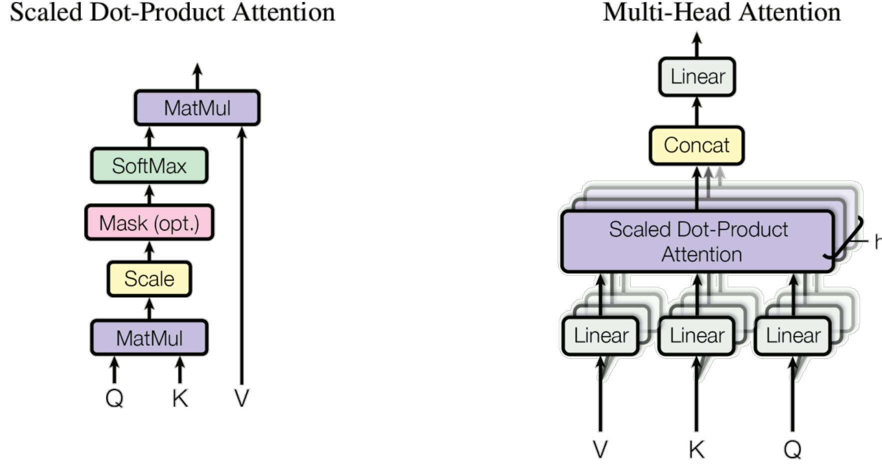

Figure 1.4: The Transformer multi-head self-attention. Image source: Figure 2 of Vaswani et al. (2017).

positions (i.e., positions to the right of the current position in the sequence) have a value of $-\infty$ before applying the softmax. This results in zero probability for these positions, ensuring that the attention mechanism does not consider future tokens when predicting the next token in the sequence.

The other variant is the *encoder-decoder multi-head attention*. Here, the queries come from the previous decoder layer, and the keys and values come from the output of the encoder allowing every position in the decoder to attend to all positions in the input sequence.

1.4.5 Feed-Forward Networks, Residual Connections, and Layer Normalization

Following the multi-head attention sub-layer, the Transformer model also includes a fully connected feed-forward network in each layer of the encoder and decoder. The same feed-forward network is applied independently to each position in the sequence.

Formally, given an input matrix $\boldsymbol{Z} \in \mathbb{R}^{n \times d}$, where each row represents one of the n input vectors of dimension d, the operation of the feed-forward network is defined as:

$$\text{FFN}(\boldsymbol{Z}) = \text{ReLU}(\boldsymbol{Z}\boldsymbol{W}^1 + \boldsymbol{b}^1)\boldsymbol{W}^2 + \boldsymbol{b}^2, \tag{1.32}$$

where $\boldsymbol{W}^1 \in \mathbb{R}^{d \times d_f}$, $\boldsymbol{b}^1 \in \mathbb{R}^{d_f}$, $\boldsymbol{W}^2 \in \mathbb{R}^{d_f \times d}$, and $\boldsymbol{b}^2 \in \mathbb{R}^{d}$ are the weights and biases of two affine transformations, respectively, and d_f is the dimensionality of the intermediate layer, usually bigger than d.

To enhance training stability and enable deeper model training, Vaswani et al. (2017) used *residual connections* together with *layer normalization* (Ba et al., 2016) after each sub-layer. Formally, the output of each sub-layer is added to its input and then normalized:

$$\boldsymbol{Y} = \text{LayerNorm}(\boldsymbol{X} + \text{SubLayer}(\boldsymbol{X})), \tag{1.33}$$

where LayerNorm refers to layer normalization, $\boldsymbol{X}$ is the input matrix to

the sub-layer, SubLayer represents the operation performed by the concrete sub-layer (attention or FFN), and $\boldsymbol{Y}$ is the output matrix. Layer normalization helps combat the problem of changing distributions in deep neural networks, known as internal covariate shift, first described by Ioffe and Szegedy (2015).

1.5 Pretrained Language Models

After the introduction of Transformer architecture, there has been an explosion of pretrained Transformer-based language models, such as BERT (Devlin et al., 2019), GPT (Radford et al., 2018) and its subsequent iterations GPT-2 (Radford et al., 2019) and GPT-3 (Brown et al., 2020), RoBERTa (Liu et al., 2019), T5 (Raffel et al., 2020), and others. These models are first pretrained on a large corpus of text and then fine-tuned for specific tasks. The underlying principle behind these models is transfer learning (Section 1.5.1), which has significantly improved the state-of-the-art across various NLP tasks.

1.5.1 Transfer Learning in NLP

Transfer learning is a machine learning technique that uses a pretrained model on a second related task. In natural language processing, models are usually pretrained on a large text corpus and then fine-tuned on a smaller task-specific dataset (Radford et al., 2018; Devlin et al., 2019). The motivation behind transfer learning is that it allows models to use the knowledge learned from the larger dataset, often leading to better performance than training on the smaller task-specific dataset alone.

1.5.2 Pretrained Language Models and Pretraining Methods

Pretrained language models are a specific class of Transformer-based language models pretrained on a large text corpus. The pretraining phase of transfer learning means training the Transformer model to learn representations from the given text corpus. The pretraining is accomplished unsupervised or self-supervised, using techniques like standard (autoregressive) language modeling (Radford et al., 2018), or its variants such as masked language modeling (MLM) (Devlin et al., 2019).

Autoregressive Language Modeling

Given all the preceding tokens, the *autoregressive language modeling* objective is to predict the next token in a sequence, as introduced in section 1.2. Formally, given a sequence of tokens $x_{1:t} = (x_1, x_2, \ldots, x_t)$, the language model objective is to maximize the likelihood of each token given its preceding tokens. The following per-example loss function is used to minimize the negative log-likelihood of the token sequence:

$$L_{\text{LM}} = -\log P(x_i | x_{1:i-1}), \tag{1.34}$$

for all $i \in \{1, 2, \ldots, t\}$. Here, $P(x_i|x_{1:i-1})$ represents the probability that the model picks the actual token x_i at the i-th position, given the preceding tokens $x_{1:i-1}$. The per-example loss L_{LM} is a cross-entropy loss defined in Equation 1.22.

Masked Language Modeling

The *masked language model* (MLM) randomly masks a portion of the input tokens by replacing them with a special `<MASK>` token. To goal is to predict those masked tokens. Formally, given a sequence of tokens $x_{1:t} = (x_1, x_2, \ldots, x_t)$, the MLM objective is to minimize the following per-example loss:

$$L_{\text{MLM}} = -\log P(x_i|x_{1:i-1}, x_{i+1:t}), \tag{1.35}$$

where i is the index of the masked token. L_{MLM} is also a cross-entropy loss defined in Equation 1.22.

While the MLM objective enables learning from both the left and right context by masking tokens in the sequence, the standard language model objective learns only from the left context due to its autoregressive nature.

1.5.3 Fine-tuning Large Language Models

After pretraining the large language models (LLMs) on a large corpus, these models are fine-tuned on a specific task using a smaller, possibly labeled, dataset. Fine-tuning adjusts the pretrained model's weights to make it more suitable for the specific task.

Fine-tuning assumes initializing the model's weights with the values from the pretraining phase. Adding some task-specific layers to the pretrained model may also be required depending on the task. These layers are usually added on top of the pre-existing architecture with weights initialized randomly. This pretrained model is then trained on the task-specific dataset.

For example, a fully-connected layer with an output size corresponding to the number of classes is added to the pretrained model for classification tasks. Suppose we want to classify the whole input sequence. In that case, the input to the classification layer is usually the transformer output vector corresponding to a special token (like the `[CLS]` token in BERT by Devlin et al. (2019)) that carries the complete sequence information. Then, a softmax activation function is applied to this output to produce probabilities for each class. On the other hand, if the task involves classifying individual tokens within the sequence, each transformer output vector is passed through the classification fully-connected layer, followed by the softmax activation function to produce class probabilities for each token.

1.5.4 Influential Large Language Models

In this section, we will introduce some of these pretrained language models.

BERT

BERT (Devlin et al., 2019) is the Transformer encoder-based model, employing multiple encoder layers to capture bidirectional contextual relations between

words in a text. The model is pretrained using two objectives: masked language modeling MLM described in 1.5.2, and the next-sentence prediction, where the model is trained to predict whether two sentences follow each other in the original text.

RoBERTa

RoBERTa is a variant of BERT proposed by Liu et al. (2019). It uses the same architecture as BERT. The differences between RoBERTa and BERT are in the pretraining process. RoBERTa drops the next sentence prediction task, which Liu et al. (2019) claim does not contribute significantly to the model's performance. Instead, RoBERTa increases the amount of pretraining data and employs larger batch sizes for more robust training. The masked language model task is also slightly modified to have dynamic masking rather than static masking used in BERT, allowing the model to see each sentence with different masks multiple times.

GPT

GPT, was introduced by Radford et al. (2018). Unlike BERT, GPT uses only the Transformer decoder. GPT is pretrained on a large text corpus using a standard autoregressive language modeling task (1.5.2) by predicting the next word in the sequence. The original GPT was significantly improved with GPT-2 (Radford et al., 2019) and GPT-3 (Brown et al., 2020). GPT-2, with 1.5 billion parameters, was a much larger model than the original GPT (117 million parameters), allowing it to generate more coherent and contextually rich text. GPT-3 took the scale approach even furthe. With 175 billion parameters, GPT-3 was able to generate text of unprecedented quality and to solve a wide array of tasks without task-specific training data, simply through "few-shot learning", where the model generates answers based on a few example inputs and outputs given at inference time.

T5

T5 was introduced by Raffel et al. (2020). It uses the full Transformer model, including an encoder and a decoder. T5 represents every NLP task as a text generation task, including tasks usually framed as classification or regression, making the model versatile and capable of handling various tasks without significant modifications to its architecture. T5 pretraining uses a *denoising autoencoder* objective, where a unique noise token randomly replaces some contiguous sequences of tokens in the input text. The model is trained to generate the original text from this corrupted version, which helps the model to understand the text's context and semantic meaning. In fine-tuning to a specific task, T5 incorporates task-specific prefixes. For example, during a translation task from English to German, the input text might be prepended with "translate English to German:".

2. Dialogue Management and Related Work

Dialogue management (DM) is a crucial component of any dialogue system, particularly in task-oriented systems, as we described in Subsection 1.1.2. The module keeps track of the conversation and determines the system's next action. It plays a pivotal role in driving the conversation forward and ensuring the successful completion of the task. Dialogue management traditionally involves two sub-components: dialogue state tracking (DST) and action selection. DST is responsible for maintaining and updating the dialogue state, representing the dialogue history containing various elements such as slots and their assigned values. The action selection decides the system's next action based on the updated dialogue state.

Dialogue management approaches have significantly evolved over the past few decades. The evolution of dialogue systems and dialogue management has been driven by the increasing complexity of dialogue tasks, the availability of larger and more diverse dialogue datasets, and advancements in machine learning and natural language processing (NLP) technologies.

This chapter will delve into the various methods used for dialogue management and the related work in the field. In the first part, we briefly describe the evolution of dialogue management, starting with rule-based systems and their improvement using states and probabilistic representations (Sections 2.1). Then we look at modern data-driven methods using machine learning (Section 2.2), covering dialogue state tracking and action selection approaches using supervised learning, reinforcement learning, and end-to-end methods.

2.1 Early Dialogue Management

In the earliest stages, dialogue systems and dialogue management were primarily rule-based, such as ELIZA (Weizenbaum, 1966), a computer program developed in the 1960s at MIT and one of the earliest attempts to create a conversational agent (chatbot). It used a set of predefined rules to analyze the input text by searching for keywords and determining the system's response. However, these systems could only handle cases covered by the predefined rules.

To overcome the limitations of rule-based systems, state-based dialogue management approaches were introduced that treated the dialogue as a sequence of states. The system would transition from one state to another based on the user's input and a predefined state-transition function. Another improvement came with the introduction of the belief state, a probabilistic representation of the system's belief about the state of the dialogue, which allowed to handle uncertainty in dialogue systems (Williams and Young, 2007).

2.2 Modern Approaches to Dialogue Management

With the rise of machine learning, data-driven methods for dialogue management began to emerge. These approaches aim to learn strategies directly from data, bypassing the need for predefined rules or state transitions.

The introduction of belief states led to the use of Markov Decision Processes (MDPs) (Young, 2000) and Partially Observable Markov Decision Processes (POMDPs) (Young et al., 2007, 2010; Gašić and Young, 2011) for dialogue management, followed by approaches based on reinforcement learning (Jurčíček et al., 2011; Gasic et al., 2011; Su et al., 2017), and more recently, end-to-end learning using deep neural networks (Serban et al., 2016; Bordes et al., 2017; Rajendran et al., 2018).

2.2.1 Dialogue State Tracking

The introduction of the belief state was an important step in dialogue management. Recently, several methods have been proposed to enhance the dialogue state tracking process.

For instance, Henderson et al. (2013) introduced a fully connected deep neural network approach for dialogue state tracking which predicted probability distribution over all possible values for each slot. Following their work, Mrkšič et al. (2015) used basic recurent neural networks (RNNs) to capture contextual information for modeling and labeling complex dynamic sequences. Their model improved previous approaches by efficiently handling multiple domains in a single conversation. Žilka and Jurčíček (2015) then used Long Short-Term Memory (LSTM) networks for incremental dialogue state tracking. Their model, LecTrack, used LSTM to process the whole dialogue history incrementally, eliminating the need for (spoken) natural language understanding (NLU).

The scalability of unbounded (e.g., date, time, or location) or dynamic (e.g., movies or usernames) slots and their values was addressed by Rastogi et al. (2017). Their work did not rely on an exhaustive enumeration of possible slot values. Instead, they estimate possible values at each turn by a size-bounded candidate set of slot values from the NLU or system actions. The bidirectional GRU then represents the dialogue state as a distribution over these candidate sets. Goel et al. (2018, 2019) improved this solution, enhancing the robustness of the scalable approach to handle diverse scenarios better.

Classification-based DST Using Pretrained Language Models

The introduction of large language models (LLMs) has brought a shift in DST due to the ability of these models to learn high-quality contextual embeddings that can capture complex relations in the text. In particular, BERT has been adopted by Chao and Lane (2019) who proposed BERT-DST, a dialogue state tracking model, which directly predicts slot values from the dialogue context, as illustrated in Figure 2.1. The model uses BERT to encode the previous system and current user utterances into their contextual embeddings. Then, it uses the first token (`[CLS]`) embedding, which represents the whole input, to predict for each slot

whether the value for this slot should be updated from the input by looking for its span or not. If the value should be updated, the contextual embeddings of the tokens are used to predict the start and end token of the span.

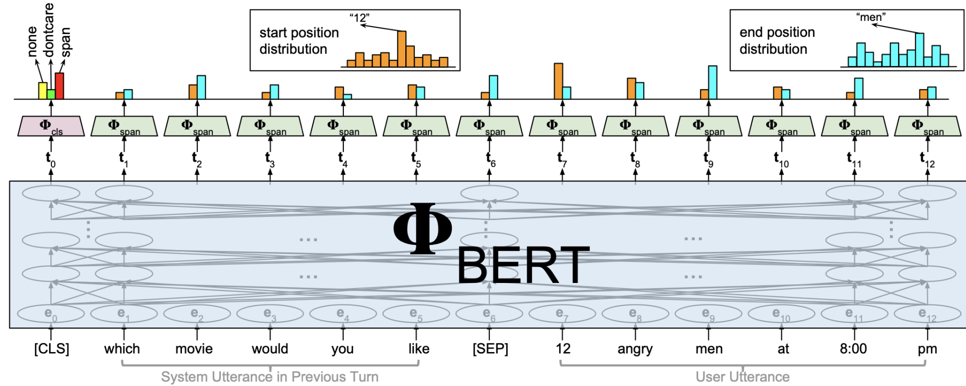

Figure 2.1: The architecture of the BERT-DST model. Image source: Figure 1 of Chao and Lane (2019).

Heck et al. (2020) further improved upon the BERT-DST by introducing their TripPy model, which uses three different sources to fill slots with values. It improves upon BERT-DST by extracting values not only from the current user and previous system utterances but also a system inform memory that keeps track of the values offered or recommended by the system, and a dialogue state memory, that allows values to be copied over from a different slot that is already contained in the dialog state as shown in Figure 2.2. TripPy achieved state-of-the-art performance on several benchmark datasets, including a joint goal accuracy of over 55% on the Multiwoz 2.1 dataset (Eric et al., 2020).

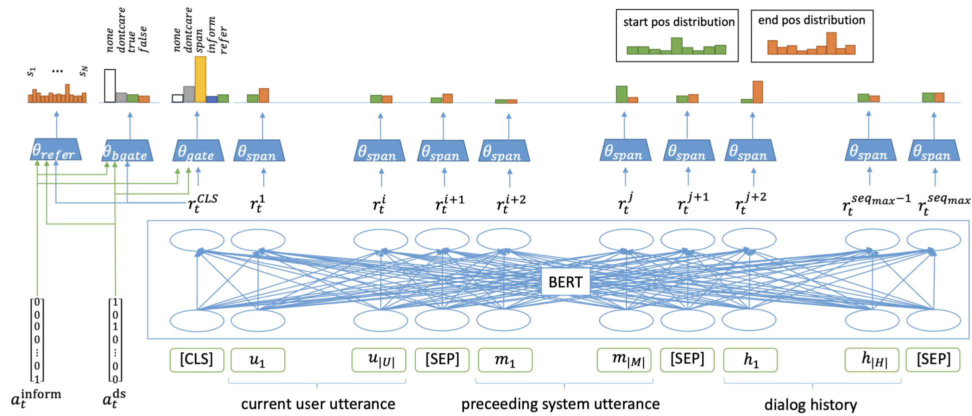

Figure 2.2: The architecture of the TripPy model. Input is the previous system utterance, current user utterance, and dialog history; output is the dialogue state. Image source: Figure 2 of Heck et al. (2020).

Generation-based Dialogue State Tracking

Another approach to dialogue state tracking focuses on directly generating the dialogue state as a text using encoder-decoder model architecture. Wu et al. (2019)

propose a Transferable Dialogue State Generator (TRADE) that generates dialogue states from utterances using a copy mechanism. The model comprises an utterance encoder, a slot gate, and a state generator, which are shared across domains, allowing it to handle domains, slots, and values unseen during the training. This is achieved by jointly training the model on several domains simultaneously. The ability to handle unseen domains, slots, and values makes this approach applicable in a real-world scenario where a complete ontology is hard to obtain and, even if it exists, the number of possible slot values can be intractable or unbounded (e.g., date, time, location or name). They used bi-directional GRU to encode dialogue utterances into a sequence of fixed-length vectors. The state generator then independently predicts the value for each `(domain, slot)` pair, using the sum of their embeddings as the first input to the GRU decoder. The slot gate then decides whether the generated value for the `(domain, slot)` pair should be used. The model is shown in Figure 2.3.

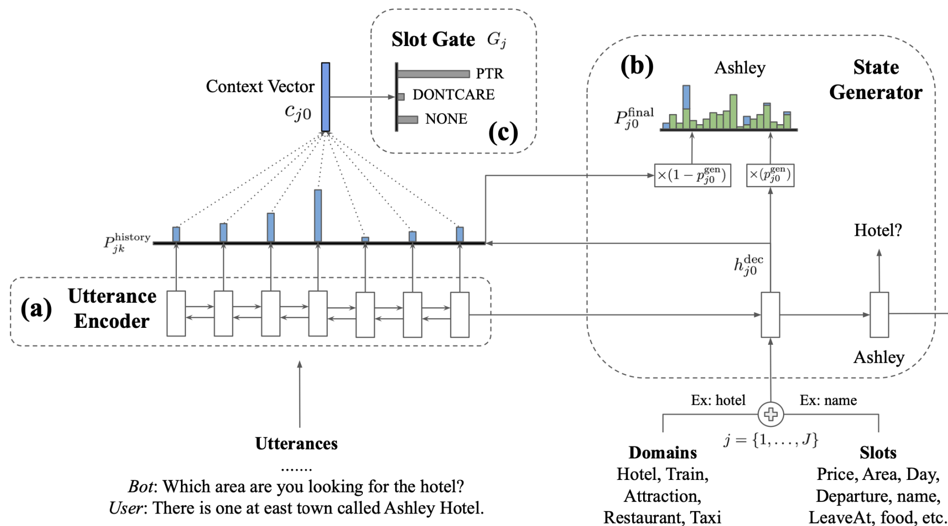

Figure 2.3: The architecture of the TRADE model, which includes (a) an utterance encoder, (b) a state generator, and (c) a slot gate, all of which are shared among domains. Image source: Figure 2 of Wu et al. (2019).

Recently, Lee et al. (2021) improved generation-based dialogue state tracking by using a pre-trained T5 language model. They use two decoding strategies for generation-based DST at a particular turn: sequential (a) and independent (b)(c), both of which are shown in Figure 2.4. In the first case (top system (a) in Figure 2.4), only the dialogue history, as a concatenation of last L user and system utterances, is taken as input to the encoder, and all domain-slot-value triplets are generated sequentially. In the second case (two systems (b)(c) in Figure 2.4), the values for each domain-slot pair are generated independently. The input consists of a dialogue history followed by domain and slot names (case (b)), optionally with the description of the slot that can also include a list of all possible values (case (c)), all in textual form. The corresponding output is a generated value for the domain-slot pairs in the input with a possible value of "none". A more detailed example is shown in Figure 2.5.

Using a pre-trained language model, such as T5, allows the system to generate any value as a text for any domain and slot, showing greater generalization

than previous models. Furthermore, the use of task-specific prompts aids in contextualizing the dialogue, enabling the model to generate more accurate and appropriate slot values.

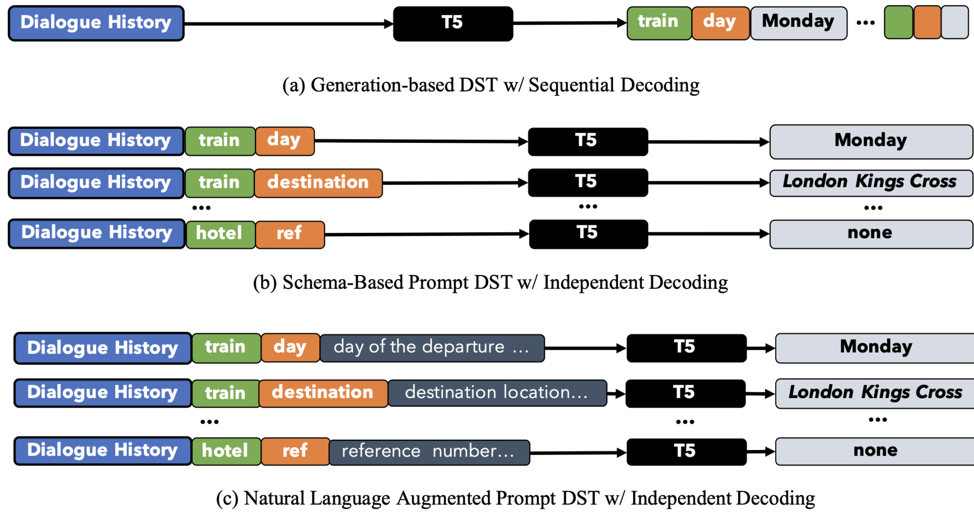

Figure 2.4: The overview of generative DST approaches for the multi-domain scenario using the T5 model illustrates three different generative approaches. Image source: Figure 1 of Lee et al. (2021).

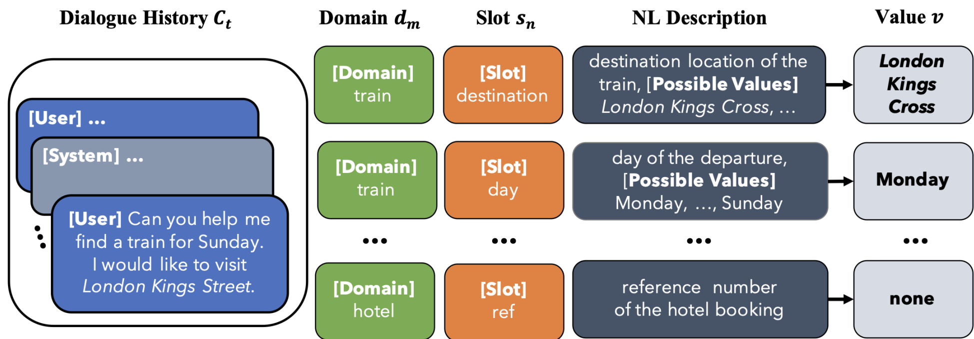

Figure 2.5: Example of generative DST for multi-domain scenario using T5 model with concrete examples for dialogue history, domain names, slot names, and natural language descriptions (types, set of valid values) for slots. Image source: Figure 1 of Lee et al. (2021).

2.2.2 Action Selection / Dialogue Policy

The DM must also choose the next system action to generate an appropriate system response. Recently, machine learning approaches based on supervised and reinforcement learning have been used (Brabra et al., 2022). The supervised approach learns the strategies for action selection from a set of labeled data. In contrast, the reinforcement approach focuses on optimizing the strategies by a trial-and-error process using reinforcements that represent rewards or punishments.

Supervised Learning in Action Selection

The first standard approach for action selection is to use supervised learning to train a classification model to predict the next system action based on the dialogue state (Brabra et al., 2022). McLeod et al. (2019) introduced an innovative approach to dialogue policy training, where they leverage multi-task learning for system action selection, using supervised learning and features from related tasks such as slot-filling and user-intent classification. Su et al. (2016) represent dialogue policy as a neural network with one hidden layer. The input to the model is the belief state. The output is the dialogue action, produced using a combination of softmax and sigmoid output layers to predict system dialogue acts and associated slots.

The second approach is implementing an action selection model that produces a system action directly from a user utterance, effectively combining NLU and DM modules into one model. A common approach is based on the encoder-decoder architecture, which takes a sequence consisting of the user utterance and some optional information, such as dialogue history, as input and generates the target sequence with a corresponding action. Zhao et al. (2017) implemented such encoder-decoder model using CNNs and LSTMs.

Reinforcement Learning in Action Selection

While supervised learning usually serves as an initial step in developing an action selection module, it has a substantial drawback. This approach typically trains the module to choose the optimal action for the current turn without considering the potential long-term implications of the action on the entire dialogue. Reinforcement learning can be used to develop an action selection module that can strategically plan ahead (McTear, 2021).

In the context of reinforcement learning (RL), action selection is often represented as a policy that maps the dialogue state into system actions. Unlike supervised learning approaches, RL aims to optimize the policy for long-term rewards, which is well-aligned with the nature of dialogues, where the quality of an action is often not immediately evident and can be influenced by the course of the entire conversation. Frameworks such as Markov Decision Processes (MDPs) (Young, 2000) and Partially Observable Markov Decision Processes (POMDPs) (Young et al., 2007, 2010; Gašić and Young, 2011) are usually utilized to model the dialogue system response generation as a stochastic policy.

There are many approaches to finding the optimal policy in RL. One approach is Deep Q-Networks (DQNs) (Mnih et al., 2015), a tool for approximating the Q-function in RL. This concept was applied to dialogue systems by Li et al. (2017) and Lipton et al. (2017), who leveraged DQNs in end-to-end task completion and efficient exploration in dialogue systems.

Policy Gradient methods and the REINFORCE algorithm directly approximate the policy and have also been used for policy optimization in dialogue management, as demonstrated by Su et al. (2017).

Even though RL provides an elegant solution for long-term planning in dialogues, it also requires substantial data for effective training, and the commonly available datasets often do not suffice (Li et al., 2017). Simulations of user interactions or feedback from actual users can help to gather the necessary training data.

Moreover, in dialogue systems, RL is typically applied after an initial supervised learning phase, or they can regularly alternate (Xiong et al., 2018).

2.2.3 End-to-End Dialogue Systems

End-to-end dialogue systems is an approach where a single neural network model represents the entire dialogue system. These systems attempt to capture the full complexity of the dialogue without needing separately trained components. Typically, the input to end-to-end models is the user's utterance, and the output is the system's response. This paradigm differs from traditional pipeline-based approaches, where each dialogue system component is individually designed and trained.

However, not all end-to-end models encompass the whole dialogue system. Some models merge multiple, but not all, components into a single model, such as Hybrid Code Networks (HCN) introduced by Williams et al. (2017). It consists of four components: RNN, domain-specific knowledge encoded as software, domain-specific system action templates, and conventional entity extraction module for identifying entity mentions in text, as shown in Figure 2.6. The HCN cycle starts with the user providing an utterance (step 1 in Figure 2.6) that is processed and turned into features in several ways: a bag of words vector, utterance embedding, entity extraction and its tracking (steps 2-5 in Figure 2.6). These feature components are concatenated to form a feature vector, which is processed by an RNN (steps 6-7 in Figure 2.6). The RNN calculates a hidden state vector and passes it through a fully connected layer with a softmax activation, outputting a probability distribution over possible system action templates that are masked to remove non-permitted actions and normalized back into a probability distribution (steps 9-10 in Figure 2.6). Once an action is selected from this distribution, developer code optionally fills in entities with values to produce a fully-formed action (steps 11-13 in Figure 2.6), which can be either an API call or text that is communicated to the user (steps 14, 15, 17 in Figure 2.6). This cycle repeats for every user input. The taken action is also provided as a feature to the RNN in the next timestep (step 16 in Figure 2.6). The HCNs offer flexibility and can be trained using supervised and reinforcement learning.

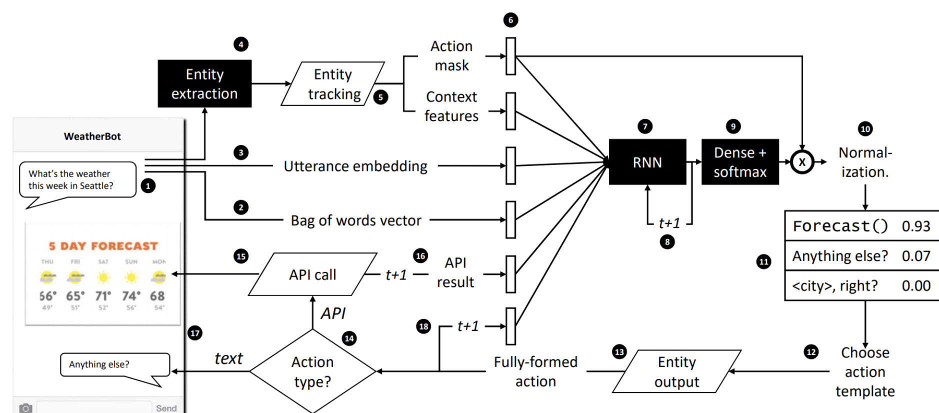

Figure 2.6: A Hybrid Code Network architecture displaying the operational loop. Image source: Figure 1 of Williams et al. (2017).

The seq2seq (or encoder-decoder) architecture is usually a way to implement the complete end-to-end dialogue system. These models directly map user utterances to system responses, abstracting away the details of the intermediate dialogue management tasks, as shown by Lei et al. (2018); Lin et al. (2020); Peng et al. (2021); Kulhánek et al. (2021).

3. Practical Dialogue Management

This chapter will describe our own approach to dialogue management. The research in dialogue systems has recently gravitated towards end-to-end neural models that directly generate a system response given the user utterance. However, this approach is notorious for needing large amounts of annotated data, which is hard to obtain. Moreover, end-to-end models that represent the whole dialogue system, including NLG, are generally unsafe to use in practical applications due to their tendency to hallucinate and generate ungrounded outputs (Ji et al., 2023), which makes such models generally unsafe for practical applications.

Therefore, in practice, dialogue systems often remain composed of multiple separate modules: natural language understanding (NLU), dialogue management (DM), and natural language generation (NLG). They can also join NLU and DM into a single module. With this in mind, our goal is to explore potential improvements in dialogue management (DM) methods, which could serve as a practical starting point when designing new dialogue systems. Our approach leverages pre-trained language models and focuses on supervised learning methods. Our focus on supervised learning methods originates from the challenges of using reinforcement learning, which requires substantial data and a fully functioning dialogue system for effective training, as the system needs to be capable of interacting with users or simulations. Therefore, supervised learning, with its lower data requirements and more straightforward training process, offers a more practical approach for training DM models.

This chapter delves into two components of dialogue management: dialogue state tracking (Section 3.1) and action selection (Section 3.2). The former describes our approach to generate updated dialogue states. The latter proposes two methodologies for action selection: a generative method and a classification-based one, both utilizing distinct mechanisms to choose actions from a predefined set. These proposed models aim to address practical challenges in designing efficient dialogue systems.

3.1 Dialogue State Tracking

The section will introduce our approach to DST, which, similarly to Lee et al. (2021), utilizes a pre-trained transformer T5 language model to generate dialogue state. In their work, Lee et al. (2021) created a T5 model that uses dialogue history, domain name, and slot name as input to produce the corresponding slot value as a text output (see Section 2.2.1).

In contrast to their method, we use the T5 language model differently: We use a custom text-based representation of the dialogue state. Our model receives as input the text representation of the previous dialogue state, along with the previous system response (providing a context of length one) and the current user utterance, inspired by the end-to-end systems such as Kulhánek et al. (2021). The output is an updated text representation of the dialogue state.

3.1.1 Theoretical Description of Generative DST

The proposed model for dialogue state tracking is based on the sequence-to-sequence or encoder-decoder Transformer architecture (Section 1.4.1). At each turn t, the model takes as input a sequence that consists of the text representation of the previous dialogue state $\text{str}_S(S_{t-1})$, the context with the previous system response C_{t-1}, and the current user utterance U_t, and generates a text representation of the updated dialogue state $\text{str}_S(S_t)$. We can express this functionally as a parametrized mapping g_{DST} from the tuple of $(\text{str}_S(S_{t-1}), C_{t-1}, U_t)$ to $\text{str}_S(S_t)$. Formally,

$$\text{str}_S(S_t) = g_{\text{DST}}\left(\text{str}_S(S_{t-1}), C_{t-1}, U_t; \boldsymbol{\theta}\right) \tag{3.1}$$

where, $\boldsymbol{\theta}$ represents the parameters of the model. The model can also be interpreted as a probabilistic model representing the conditional probability distribution $P(\text{str}_S(S_t)|\text{str}_S(S_{t-1}), C_{t-1}, U_t; \boldsymbol{\theta})$.

At the start of the dialogue, $\text{str}_S(S_0)$ represents a text representation of the initial empty dialogue state. Similarly, C_0 is an empty string representing the initial empty context. The function g_{DST} can be described as the encoding-decoding process of the model:

$$\boldsymbol{H}_t = \text{encoder}(\text{str}_S(S_{t-1}), C_{t-1}, U_t; \boldsymbol{\theta}) \tag{3.2}$$

$$\text{str}_S(S_t) = \text{decoder}(\boldsymbol{H}_t; \boldsymbol{\theta}) \tag{3.3}$$

where $\boldsymbol{H}_t \in \mathbb{R}^{L \times h}$ denotes the matrix of hidden states obtained after passing the input sequence $(\text{str}_S(S_{t-1}), C_{t-1}, U_t)$ of length L through the encoder, and h is the hidden size. The hidden states are fed into the decoder to generate the updated dialogue state $\text{str}_S(S_t)$.

The learning objective of the model is to find the parameters $\boldsymbol{\theta}$ that minimize the average negative log-likelihood (or equivalently maximize the log-likelihood) of the dataset, also known as the cross-entropy loss (Equation 1.23). Formally, suppose we denote the entire dataset as $\mathcal{D}$, where each element $(S_{t-1}, C_{t-1}, U_t, S_t)$ in $\mathcal{D}$ represents one turn in a dialogue. In that case, the objective function to minimize is given by:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} -\frac{1}{|\mathcal{D}|} \sum_{(S_{t-1}, C_{t-1}, U_t, S_t) \in \mathcal{D}} \log P(\text{str}_S(S_t)|\text{str}_S(S_{t-1}), C_{t-1}, U_t; \boldsymbol{\theta}). \tag{3.4}$$

Here, $|\mathcal{D}|$ denotes the size of the dataset. The learning process aims to find the model parameters that minimize this loss, as described in Section 1.3.3.

3.1.2 Dialogue State and the String Representation

Multi-domain Dialogue State

In a multi-domain dialogue system, the dialogue state denoted as S is a set of ordered pairs (d^k, S^k), where d^k denotes the domain name, and S^k denotes the domain state. Each domain state S^k can be represented as a set of ordered pairs (s_i^k, v_i^k), where s_i^k represents a slot name and v_i^k is its corresponding value. We can formalize it as follows:

$$S = \left\{ (d^k, S^k) \right\}_{k=1}^{n}, \tag{3.5}$$

where for each $k \in 1, 2, ..., n$,

$$S^k = \left\{ (s_i^k, v_i^k) \right\}_{i=1}^{m_k}. \tag{3.6}$$

Here, n represents the number of domains in the dialogue system, and m_k is the number of slots in the k-th domain. The set of domains $\left\{ d^k \right\}_{k=1}^{n}$ and the set of its corresponding slots $\left\{ s_i^k \right\}_{i=1}^{m_k}$ are finite and defined by the system's capabilities. For each slot s_i^k, the set of all possible values $\left\{ v_{i_j}^k \right\}_{j=1}^{\alpha}$ can be infinite.

Dialogue State String Representation

The function $\text{str}_S()$ converts the dialogue state S into its string representation. It iterates over each domain d^k and corresponding domain state S^k in S, converts each domain state S^k into a string by concatenating the sorted slot-value pairs (separated by commas), and then concatenates the domain name d^k with the string representation of its domain state S^k. The string representations for all the domains are then concatenated together (separated by semicolons). Formally:

1. For each ordered pair (d^k, S^k) in S, we start by alphabetically sorting the pairs in ascending order of the domain names. This is done to maintain consistency and readability, as the sorted string will always present information about the same domain in the same relative position.

2. For each domain d^k, we construct a string representation of its domain state S^k by iterating over all slot-value pairs (s_i^k, v_i^k) in S^k. To maintain consistency, we sort the pairs in ascending order of slot names s_i^k. The string representation of the domain state, denoted as $\text{str}_S(S^k)$, is then formed as follows:

$$\text{str}_S(S^k) = s_1^k + \text{`` : ''} + v_1^k + \sum_{i=2}^{m_k} (\text{``, ''} + s_i^k + \text{`` : ''} + v_i^k) \quad \text{if } v_i^k \neq \text{``None'', else ``''} \tag{3.7}$$

   where s_i^k is the slot name, v_i^k is the slot value, and the sum operation indicates string concatenation. Note that we ignore any slot-value pair with a value of "None", as these do not contribute meaningful information to the dialogue state and would only make the string representation longer.

3. The overall string representation of the dialogue state $\text{str}_S(S)$ is then constructed by combining the domain strings $\text{str}_S(S^k)$, each prefixed by their respective domain name d^k and a hyphen for separation. A semicolon and a space separate the individual domain strings:

$$\begin{aligned} \text{str}_S(S) = & d_1 + \text{`` - ''} + \text{str}_S(S^1) + \sum_{k=2}^{n} (\text{``; ''} + d^k + \text{`` - ''} + \text{str}_S(S^k)) \\ & \text{if } \text{str}_S(S^k) \neq \text{``''}, \text{ else ``''} \end{aligned} \tag{3.8}$$

   where an empty string for $\text{str}_S(S^k)$ signifies that the domain state S^k does not contain any slot-value pairs where the value is not "None". In such a case, we ignore this domain.

Inverse String Representation Function

The inverse function of $\text{str}_S()$, which we denote as $\text{str}_S^{-1}()$, converts the string representation of the dialogue state back into its structured representation. This process can be useful, for example, when we want to inspect the model's output in a structured form or when we need to compare the output to a ground-truth dialogue state. The function operates as follows:

1. Initialize an empty dialogue state $S = \{\}$, represented as an empty set of domain-state pairs.

2. Split the string representation of the dialogue state, $\text{str}_S(S)$, into domain sections. The domain sections are separated by a semicolon and a space in the string representation. Thus, we can split the string representation using this separator. Formally:

$$\text{DomainSections} = \text{Split}(\text{str}_S(S), \text{``; ''}) \tag{3.9}$$

3. Iterate over each domain section in DomainSections. Each domain section is a string representing a domain and its state. The domain name and the string representation of its state are separated by " - ". Split the domain section into the domain name d^k and the string representation of its state $\text{str}_S(S^k)$. Formally:

$$d^k, \text{str}_S(S^k) = \text{Split}(\text{DomainSection}, \text{`` - ''}) \tag{3.10}$$

4. Check if the domain name d^k is recognized by the system. If it is not, skip this domain section. If the domain name is recognized, initialize an empty domain state $S^k = \{\}$.

5. Split the string representation of the domain state, $\text{str}_S(S^k)$, into slot-value pairs. The slot-value pairs are separated by ", ". Each is a string with the slot name and the value separated by " : ". Formally:

$$\text{SlotValuePairs} = \text{Split}(\text{str}_S(S^k), \text{``, ''}) \tag{3.11}$$

6. Iterate over each slot-value pair in SlotValuePairs. Split the slot-value pair into the slot name s_i^k and the value v_i^k. Formally:

$$s_i^k, v_i^k = \text{Split}(\text{SlotValuePair}, \text{`` : ''}) \tag{3.12}$$

7. Check if the slot name s_i^k is recognized for the domain d^k. If it is not, skip this slot-value pair. If the slot name is recognized and the value v_i^k is not equal to "None" or if the system is configured to include "None" values, add the slot-value pair to the domain state S^k. Formally:

$$S^k = S^k \cup \{(s_i^k, v_i^k)\} \tag{3.13}$$

8. If the domain state S^k is not empty, add the domain-state pair to the dialogue state S. Formally:

$$S = S \cup \{(d^k, S^k)\} \tag{3.14}$$

9. Finally, return the constructed dialogue state S.

Example

1. Consider the dialogue state S which is defined as follows:

$$\begin{aligned} S = \{&(\text{hotel}, \{(\text{area}, \text{north}), (\text{bookday}, \text{sunday}), \\ &(\text{bookpeople}, 5), (\text{bookstay}, 3), (\text{intent}, \text{book_hotel}), \\ &(\text{name}, \text{avalon}), (\text{stars}, 4), (\text{type}, \text{guesthouse})\}), \\ &(\text{train}, \{(\text{departure}, \text{cambridge}), (\text{intent}, \text{find_train})\})\} \end{aligned}$$

2. When applying the $\text{str}_S()$ function to the dialogue state S, we obtain the following string representation:

$$\begin{aligned} &\text{str}_S(S) = \\ &\text{hotel - area : north, bookday : sunday, bookpeople : 5, bookstay : 3,} \\ &\text{intent : book_hotel, name : avalon, stars : 4, type : guesthouse;} \\ &\text{train - departure : cambridge, intent : find_train} \end{aligned}$$

3. Finally, if we apply the inverse function $\text{str}_S^{-1}()$ to the string representation of the dialogue state S, we recover the original dialogue state S.

3.1.3 The Input and Output Strings

As explained in Section 3.1.1, at turn t, the model takes the string representation of the previous dialogue state $\text{str}_S(S_{t-1})$, the context with the previous system response C_{t-1}, and the current user utterance U_t as inputs. These three components are concatenated into a single string using unique segment tokens to denote the beginning of each component.

We prepend a task description, denoted as T, to the input string, a directive for the T5 model that defines the task to be performed: "Update state". For each segment of the input, we use special tokens:

- $S =$ `[state]` signifies the start of the dialogue state.
- $C =$ `[context]` marks the beginning of the context.
- $U =$ `[user]` signals the start of the user utterance.

These tokens help the model identify the corresponding segments within the input string X_t at turn t, which can formally be expressed as follows:

$$X_t = T + \text{" "} + S + \text{" "} + \text{str}_S(S_{t-1}) + \text{". "} + C + \text{" "} + C_{t-1} + \text{". "} + U + \text{" "} + U_t \quad (3.15)$$

The spaces separating the tokens and their corresponding values ensure proper tokenization, while the periods at the end of each segment enhance readability and set clear segment boundaries. This approach ensures that the model receives the necessary information for DST in a well-structured string format that resembles the natural language that the T5 model was trained on and allows it to build on its pre-existing language understanding capabilities to perform the task of dialogue state tracking more effectively.

The output string Y_t of our DST model at turn t represents the updated dialogue state, denoted as $\text{str}_S(S_t)$. Formally:

$$Y_t = \text{str}_S(S_t). \tag{3.16}$$

As with the input string, this representation is beneficial as it closely resembles the standard text that the T5 model encountered during pre-training. It consists of alphabetically ordered domain names, slot names, and slot values, making it human-readable and easy to interpret. Moreover, this structured representation assists in evaluating the performance of our DST model by enabling easy comparison of the predicted dialogue state with the ground truth.

3.2 Action Selection

This section introduces our approach to action selection, which is similar in structure to our DST model. The goal of action selection is to determine the system's next response given the current dialogue state, the context of the previous system response, the current user utterance, and the optional string representation of the database counts, which represents the external knowledge base for the DM. We propose two distinct methods for action selection: a generative method and a classification-based method. In both cases, the task is to select some subset of actions from the predefined set of actions $\mathcal{A}$ of size $|\mathcal{A}|$.

3.2.1 Theoretical Description of Generative Action Selection

In the generative method, the model uses an encoder-decoder Transformer architecture, similar to the DST model (Section 1.4.1). However, instead of generating a sequence of tokens to represent an updated dialogue state, this model outputs a sequence representing the system's action.

At each turn t, the task is to select a subset of actions $A_t \subset \mathcal{A}$. Given the input sequence that consists of the string representation of the updated dialogue state $\text{str}_S(S_t)$, the context with the previous system response C_{t-1}, the current user utterance U_t, and the string representation of the database counts $\text{str}_D(D_t)$, the model generates the string representation of the selected actions $\text{str}_A(A_t)$. Formally, we can express this as a mapping g_{AS} from the tuple of $(\text{str}_S(S_t), C_{t-1}, U_t, \text{str}_D(D_t))$ to $\text{str}_A(A_t)$:

$$\text{str}_A(A_t) = g_{\text{AS}}\left(\text{str}_S(S_t), C_{t-1}, U_t, \text{str}_D(D_t); \boldsymbol{\theta}\right), \tag{3.17}$$

where $\boldsymbol{\theta}$ represents the parameters of the action selection model. The model can also be interpreted as a probabilistic model representing the conditional probability distribution $P(\text{str}_A(A_t)|\text{str}_S(S_t), C_{t-1}, U_t, \text{str}_D(D_t); \boldsymbol{\theta})$.

The function g_{AS} can also be described as the encoding-decoding process of the action selection model:

$$\boldsymbol{H}_t = \text{encoder}(\text{str}_S(S_t), C_{t-1}, U_t, \text{str}_D(D_t); \boldsymbol{\theta}) \tag{3.18}$$

$$\text{str}_A(A_t) = \text{decoder}(\boldsymbol{H}_t; \boldsymbol{\theta}) \tag{3.19}$$

where $\boldsymbol{H}_t \in \mathbb{R}^{L \times h}$ denotes the matrix of hidden states obtained after passing the input sequence $(\text{str}_S(S_t), C_{t-1}, U_t, \text{str}_D(D_t))$ of length L through the encoder, and h is the hidden size. The hidden states are fed into the decoder to generate the system's action $\text{str}_A(A_t)$.

The learning objective of the action selection model is to find the parameters $\boldsymbol{\theta}$ that minimize the dataset's average negative log-likelihood. Suppose we denote the entire dataset as $\mathcal{D}$, where each element $(S_t, C_{t-1}, U_t, D_t, A_t)$ in $\mathcal{D}$ represents one turn in a dialogue. Then, the objective function to minimize is given by:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} -\frac{1}{|\mathcal{D}|} \sum_{(S_t, C_{t-1}, U_t, D_t, A_t) \in \mathcal{D}} \log P(\text{str}(A_t) | \text{str}(S_t), C_{t-1}, U_t, \text{str}(D_t); \boldsymbol{\theta}), \tag{3.20}$$

where $|\mathcal{D}|$ is the size of the dataset.

3.2.2 Theoretical Description of Classification-Based Action Selection

In the classification-based action selection method, the task is again to select a subset of actions $A_t \subset \mathcal{A}$ using the same input as the generative method. At a given turn t, the input consists of the string representation of the updated dialogue state $\text{str}_S(S_t)$, the context with the previous system response C_{t-1}, the current user utterance U_t, and the string representation of the database count $\text{str}_D(D_t)$. The output is a binary vector $\text{vec}_A(A_t)$ of length equal to the number of actions $|\mathcal{A}|$, with 1 indicating that the corresponding action is selected and 0 otherwise. Formally, we can express this as a mapping f_{AS} from the tuple of $(\text{str}_S(S_t), C_{t-1}, U_t, \text{str}_D(D_t))$ to $\text{vec}_A(A_t)$:

$$\text{vec}_A(A_t) = f_{\text{AS}}\left(\text{str}_S(S_t), C_{t-1}, U_t, \text{str}_D(D_t); \boldsymbol{\theta}\right), \tag{3.21}$$

where $\boldsymbol{\theta}$ represents the parameters of the classification model.

The model uses a pretrained language model, to process the input sequence and compute the contextual embedding vector of the first token (`[CLS]`) as the representation of the whole input. This embedding vector is then passed through a fully connected layer with a size equal to the number of actions $|\mathcal{A}|$ and a sigmoid activation function to obtain the probabilities for each action. Formally, this can be written as:

$$\boldsymbol{H}_t = \text{LM}(\text{str}_S(S_t), C_{t-1}, U_t, \text{str}_D(D_t); \boldsymbol{\theta}_{LM}), \tag{3.22}$$

$$P_t = \text{Sigmoid}(\text{FNN}(\boldsymbol{H}_t\,[\texttt{[CLS]}]\,; \boldsymbol{\theta}_{FNN})), \tag{3.23}$$

$$\text{vec}_A(A_t) = \text{Binary}(P_t), \tag{3.24}$$

where LM denotes the pretrained language model, Sigmoid is the sigmoid activation function (Equation 1.14), FNN represents the fully connected layer, `[CLS]` refers to the token in the sequence that represents the whole input (usually the first one), and Binary is a function that converts the probabilities into a binary vector based on a threshold (e.g., 0.5).

The task is a multi-label binary classification. The ground-truth set of actions at turn t is defined as a subset of $\mathcal{A}$ and denoted as G_t. Its corresponding binary

vector representation is a binary vector of length $|\mathcal{A}|$, denoted as $\text{vec}_A(G_t)$. The predicted vector of probabilities, the output from the Sigmoid function in our model, is denoted as P_t. For a given action a, the predicted probability and the ground-truth binary value are obtained by indexing.

The binary cross-entropy loss (Equation 1.19) is used to compute the loss between the predicted and ground-truth values. For a single action a it is defined as:

$$\text{BCE}_a(G_t[a], P_t[a]) = -\text{vec}_A(G_t)[a] \log(P_t[a]) - (1 - \text{vec}_A(G_t)[a]) \log(1 - P_t[a]) \tag{3.25}$$

and the total binary cross-entropy loss for a dialogue turn t is:

$$\text{BCE}(G_t, P_t) = \sum_{a \in \mathcal{A}} \text{BCE}_a(G_t[a], P_t[a]). \tag{3.26}$$

The learning objective of the classification model is to find the parameters $\boldsymbol{\theta} = \boldsymbol{\theta}_{LM} \cup \boldsymbol{\theta}_{FNN}$ that minimize the dataset's average binary cross-entropy loss. If we denote the entire dataset as $\mathcal{D}$, where each element $(S_t, C_{t-1}, U_t, D_t, G_t)$ in $\mathcal{D}$ represents one turn in a dialogue, then the objective function to minimize is given by:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} -\frac{1}{|\mathcal{D}|} \sum_{(S_t, C_{t-1}, U_t, D_t, G_t) \in \mathcal{D}} \text{BCE}(G_t, P_t), \tag{3.27}$$

where $P_t = \text{Sigmoid}\left(\text{FNN}(\text{LM}(\text{str}_S(S_t), C_{t-1}, U_t, \text{str}_D(D_t); \boldsymbol{\theta}_{LM}); \boldsymbol{\theta}_{FNN})\right)$, $|\mathcal{D}|$ is the size of the dataset, LM denotes the pretrained language model, Sigmoid is the sigmoid activation function, FNN represents the fully connected layer, and BCE denotes the binary cross-entropy loss.

3.2.3 Database and the String Representation

Database Structure

In a multi-domain dialogue system, the database result D can be represented as a set of ordered pairs (d^k, D^k), where d^k denotes the domain name, and D^k denotes the database results for that domain. Each database result D^k is an ordered list of database entries, where each entry e_j^k is a set of ordered pairs (s_i^k, v_{ij}^k), representing slot names and their corresponding values. This can be formalized as follows:

$$D = \left\{(d^k, D^k)\right\}_{k=1}^{n}, \tag{3.28}$$

where for each $k \in 1, 2, ..., n$,

$$D^k = \left[e_j^k\right]_{j=1}^{p_k}, \tag{3.29}$$

and

$$e_j^k = \left\{(s_i^k, v_{ij}^k)\right\}_{i=1}^{m_k}. \tag{3.30}$$

Here, n represents the number of domains in the dialogue system, p_k is the number of database entries in the k-th domain that match the current domain state S^k, and m_k is the number of slots for k-th domain.

The database results D^k are fetched using the corresponding domain state S^k. Specifically, we retrieve all database entries that match the current domain state, i.e., those where slots and values match those in the domain state. Each database entry, however, can have additional slots and values that are not present in the domain state. These slots can take on arbitrary values unconstrained by the domain state.

Database Count String Representation

The function $\text{str}_D()$ converts the database D into a string representation with information about the size of each domain database results list. It iterates over each domain d^k and corresponding database count $|D^k|$ in D and forms a string by concatenating the domain name d^k with the number of database results for this domain $|D^k|$. The string representations for all the domains are then concatenated together (separated by semicolons). Formally:

1. For each ordered pair (d^k, D^k) in D, we start by alphabetically sorting the pairs in ascending order of the domain names.

2. For each domain d^k, we construct a string representation of its database count $|D^k|$ by combining the domain name with the count. The string representation of the database count, denoted as $\text{str}_D(|D^k|)$, is then formed as follows:
$$\text{str}_D(|D^k|) = d^k + \text{`` - ''} + |D^k| \tag{3.31}$$

3. The overall string representation of the database count $\text{str}_D(D)$ is then constructed by combining the domain strings $\text{str}_D(|D^k|)$, each separated by a semicolon and a space:
$$\text{str}_D(D) = \text{str}_D(|D^1|) + \sum_{k=2}^{n}(\text{``; ''} + \text{str}_D(|D^k|)) \tag{3.32}$$

With this, we have a consistent string representation of the database count, which can be used as a part of the input to the action selection models. This string representation helps to condense the information from the database into a form that is easy to process by the model and can help to select the actions.

3.2.4 Action and String Representation

Delexicalized Action Representation

In our dialogue system, the subset of actions the model predicts at each turn, denoted as $A \subset \mathcal{A}$, consists of delexicalized action strings. Delexicalization (Henderson et al., 2014) refers to removing the specific slot values from the action strings, maintaining only the structural format of dialogue acts, i.e.,

`[domain]-[intent]([slot]).`

For instance, the dialogue act `restaurant-inform(name=The Big Belly)` in its delexicalized form would become `restaurant-inform(name)`. This delexicalization process enables generalization across similar actions and vastly reduces the space of possible actions to predict, allowing the system to handle unseen instances more robustly.

Action String Representation $\mathbf{str}_{A_0}$

To form the basic string representation of a set of actions A, denoted as $\text{str}_{A_0}(A)$, we begin by lexicographically sorting the delexicalized actions in A, which ensures consistency in the representation. These sorted actions are then concatenated into a string, separated by commas and spaces. Formally, the process can be described as:

1. Sort the actions in A lexicographically, obtaining the ordered list A'.
2. Form $\text{str}_{A_0}(A)$ by combining the sorted actions in A', each separated by a comma and a space:

$$\text{str}_{A_0}(A) = A'[1] + \sum_{k=2}^{n} (\text{``; ''} + A'[k]) \tag{3.33}$$

   where n is the size of the sorted action list A'.

Structured Action String Representation $\mathbf{str}_A$

A different approach to forming a string representation of actions A involves grouping the actions by their corresponding domains. This string representation, denoted as str_A, consists of the domain name followed by its associated actions. The process of forming $\text{str}_A(A)$ can be outlined as follows:

1. First, group the actions in A by their respective domains. We can achieve this by splitting each action into the domain and the action parts, then appending the action part to the list of actions for the corresponding domain. We denote this grouped action representation as G, a set of ordered pairs (d^k, A^k), where d^k is the domain name and A^k is the list of actions for the domain. Formally:

$$G = \left\{(d^k, A^k)\right\}_{k=1}^{n}, \tag{3.34}$$

   where for each $k \in 1, 2, ..., n$,

$$A^k = \left\{a_i^k\right\}_{i=1}^{n_k}. \tag{3.35}$$

   Here, n represents the number of domains in the dialogue system, and n_k is the number of actions in the k-th domain.

2. For each domain d^k in G, we construct a string representation of its action list A^k by iterating over all actions a_i^k in A^k. To maintain consistency, we sort the actions a_i^k in lexicographical order. The string representation of the domain's action list, denoted as $\text{str}_A(A^k)$, is then formed as follows:

$$\text{str}_A(A^k) = a_1^k + \sum_{i=2}^{m_k} (\text{``, ''} + a_i^k) \tag{3.36}$$

3. The overall string representation of the actions A, denoted as $\text{str}_A(A)$, is then constructed by combining the domain-action strings $\text{str}_A(A^k)$, each prefixed by their respective domain name d^k and a hyphen for separation. A semicolon and a space separate the individual domain-action strings:

$$\text{str}_A(A) = d_1 + \text{`` - ''} + \text{str}_A(A^1) + \sum_{k=2}^{n} (\text{``; ''} + d^k + \text{`` - ''} + \text{str}_A(A^k)) \tag{3.37}$$

Inverse Action String Representation Function $\mathbf{str}_A^{-1}$

The inverse function of str$A$, which we denote as $\text{str}_A^{-1}()$, converts the string representation of actions back into the set of actions. The function operates as follows:

1. Initialize an empty set of actions $A = \{\}$.
2. Split the string representation of actions, $\text{str}_A(A)$, into domain-action sections. A semicolon and a space in the string representation separate the domain-action sections. Thus, we can split the string representation using this separator. Formally:

$$\text{DomainActionSections} = \text{Split}(\text{str}_A(A), \text{``; ''}) \tag{3.38}$$

3. Iterate over each domain-action section in DomainActionSections. Each domain-action section is a string representing a domain and its associated actions. The domain name and the string representation of its action list are separated by " - ". Split the domain-action section into the domain name d^k and the string representation of its action list $\text{str}_A(A^k)$. Formally:

$$d^k, \text{str}_A(A^k) = \text{Split}(\text{DomainActionSection}, \text{`` - ''}) \tag{3.39}$$

4. Check if the system recognizes the domain name d^k. If not, skip this domain-action section. If the domain name is recognized, split the string representation of the domain's action list, $\text{str}_A(A^k)$, into individual actions. The actions are separated by ", ". Formally:

$$\text{Actions} = \text{Split}(\text{str}_A(A^k), \text{``, ''}) \tag{3.40}$$

5. Iterate over each action in Actions. For each action a_i^k, form the complete action (dialogue act) by combining the domain name d^k with the action string separated by a hyphen. Append the complete action string to the action set A. Formally:

$$A = A \cup (d^k + \text{``-''} + \text{action}) \tag{3.41}$$

6. Once all domain-action sections are processed, return the list of actions A.

Example

Consider the following set of actions A:

$$\begin{aligned} A = \{&\text{booking-request(bookday)}, \text{general-greet}, \\ &\text{restaurant-inform(area)}, \text{restaurant-inform(food)}, \\ &\text{restaurant-inform(name)}, \text{restaurant-inform(pricerange)}\} \end{aligned}$$

Applying the $\text{str}_{A_0}(A)$ function, we obtain the action string representation:

$$\mathrm{str}_{A_0}(A) = \text{booking-request(bookday); general-greet; restaurant-inform(area);}$$
$$\text{restaurant-inform(food); restaurant-inform(name);}$$
$$\text{restaurant-inform(pricerange)}$$

Similarly, applying the $\mathrm{str}_A(A)$ function, we get the structured action string representation:

$$\mathrm{str}_A(A) = \text{booking - request(bookday); general - greet;}$$
$$\text{restaurant - inform(area), inform(food), inform(name),}$$
$$\text{inform(pricerange)}$$

To retrieve the set of actions A from the string representation, we apply the str_A^{-1} function and we get A.

3.2.5 The Input and Output for Action Selection

As outlined in the Sections 3.2.1 and 3.2.2, at turn t the action selection models take the string representation of the updated dialogue state $\mathrm{str}_S(S_t)$, the context with the previous system response C_{t-1}, the current user utterance U_t, and the string representation of the database counts $\mathrm{str}_D(D_t)$ as inputs. We use unique segment tokens to denote the beginning of each component.

A task description, denoted as T, is also prepended to the input string for the generative method. The task description, in this case, is "Generate actions". The other special tokens we use are the same as those used for dialogue state tracking, with an additional token $R =$ `[database]` that marks the beginning of the database results counts. Therefore, the input string X_t at turn t can be expressed formally as follows:

$$X_t = T + \text{" "} + S + \text{" "} + \mathrm{str}_S(S_t) + \text{". "} + C + \text{" "} + C_{t-1} + \text{". "} + U + \text{" "} + U_t + \text{". "} + R + \text{" "} + \mathrm{str}_D(D_t). \tag{3.42}$$

In the case of the classification-based method, the input string is the same but without the task description T and the following space. The output Y_t of our action selection model at turn t represents the selected actions. For the generative method, this is the string representation of the selected actions, denoted as $\mathrm{str}_A(A_t)$, formally:

$$Y_t = \mathrm{str}_A(A_t). \tag{3.43}$$

The inverse function str_A^{-1} can be used to obtain the action list A_t, where

$$A_t = \mathrm{str}_A^{-1}(Y_t).$$

In the classification-based method, the output is a binary vector that indicates the selected actions:

$$Y_t = \mathrm{vec}_A(A_t). \tag{3.44}$$

This binary vector is then converted to action list A_t, using the indices of 1s in the vector to select corresponding actions from the 1-1 mapping between indices and actions.

4. Experiments

In this chapter, we will introduce the MultiWOZ dataset (Section 4.1) and evaluation metrics used for DST (Section 4.2) and action selection (Section 4.3). Section 4.4 describes details of model training.

4.1 The MultiWOZ Dataset

In dialogue systems, the availability of high-quality datasets plays a vital role in research and experimentation. One such dataset is the Multi-Domain Wizard-of-Oz dataset or MultiWOZ, which is a common benchmark dataset for task-oriented dialogue systems.

4.1.1 MutliWOZ

Budzianowski et al. (2018) introduced the Multi-Domain Wizard-of-Oz (MultiWOZ) dataset to address the limitation in the scale of available data, which has been a significant obstacle in dialogue research. The dataset is a fully-labeled collection of human-human written conversations covering several domains and topics, making it a rich source for dialogue modeling.

At a size of 10,438 dialogues, it offers extensive resources for both training and evaluating dialogue systems. The dialogues span 8 domains: `attraction`, `bus`, `hospital`, `police`, `hotel`, `restaurant`, `taxi`, and `train`, with 7,032 multi-domain dialogues. The corpus was randomly split into a train, test, and validation set, with test and validation sets containing 1000 dialogues each. Each dialogue consists of a goal, multiple user and system utterances, and a belief state and set of dialogue acts with slots per turn.

The data collection process for MultiWOZ is based on crowd-sourcing, eliminating the need for professional annotators. The procedure followed the Wizard-of-Oz set-up (Kelley, 1984), which allows the collection of annotated dialogues. The procedure involved workers who participated in dialogue creation and ensured a wide diversity in the dataset. One worker played the *user* role, while the other acted as the *system*. Each *user* was given a set of tasks and goals within the conversation, and the *system* was tasked with assisting the user, creating an authentic taks-oriented dialogue exchange.

4.1.2 MultiWOZ 2.1

In subsequent iterations, improvements were made to the MultiWOZ dataset, addressing issues identified in the original version, MultiWOZ 2.0. The first significant update, MultiWOZ 2.1, was introduced by Eric et al. (2020), who found the original version to contain substantial noise in the dialogue state annotations and the dialogue utterances.

To resolve this issue, MultiWOZ 2.1 involved a re-annotation to fix common errors found in MultiWOZ 2.0. Furthermore, spelling errors were corrected, and entity names were standardized to improve the dataset's consistency. As a result, changes were made to over 32% of state annotations across 40% of the dialogue

turns. 146 dialogue utterances were corrected by aligning slot values with the dataset ontology. Moreover, MultiWOZ 2.1 integrated additional annotations such as user dialogue act information and multiple slot descriptions for each dialogue state slot.

4.1.3 MutliWOZ 2.2

Following MultiWOZ 2.1, Zang et al. (2020) introduced MultiWOZ 2.2, which further enhanced the quality and usability of the dataset. This version aimed to resolve the remaining annotation errors and inconsistencies in MultiWOZ 2.1 and introduce new elements to enrich the dataset.

In MultiWOZ 2.2, the researchers found and corrected dialogue state annotation errors in approximately 17.3% of the utterances compared to version 2.1. They also changed the dataset's ontology by disallowing vocabularies for slots that could hold many values, such as `name` and `booking-time`. In addition, they introduce *slot span annotations* for user and system utterances that are beneficial for dialogue state tracking models that utilize span annotations to locate where slot values are mentioned in the utterances. Lastly, another new feature in MultiWOZ 2.2 included *active user intents* and *requested slots* for each user utterance. These additional annotations were designed to provide more in-depth insights into the user's objectives and requests.

By introducing a new schema, standardizing slot values, correcting annotation errors, and adding span annotations, active intents, and requested slots, MultiWOZ 2.2 provides a more reliable and comprehensive resource for dialogue system research. Therefore, we use this dataset in our experiments. The dataset is split into training, validation, and test sets, as shown in Table 4.1.

	Train	**Validation**	**Test**
Dialogues	8438	1000	1000
Turn pairs (user + system)	56776	7374	7372

Table 4.1: MultiWOZ 2.2 dataset split with sizes.

4.1.4 Train Dataset and its Subsets

In our experiments, we utilize the entire training dataset, denoted as $\mathcal{D}^{\text{train}}$, and specific subsets of this dataset to train models under conditions of data limitation. These subsets are derived from $\mathcal{D}^{\text{train}}$ using a ratio parameter r, which we chose to take on values from the set $\{0.3, 0.5, 1.0\}$, where value 1.0 signifies the entire dataset. The subsets are created as follows:

- For each domain d^k, a random subset of size r of dialogues with d^k as their primary domain is chosen. We use the primary domain with which a dialogue typically starts but often evolves to include other domains as the conversation progresses. These subsets are chosen reproducibly by fixing the seed for random selection.

- Therefore, each training subset $\mathcal{D}_r^{\text{train}}$ includes dialogues (and all associated turns) proportionate to the size r for each domain d^k within $\mathcal{D}^{\text{train}}$. This

approach helps to maintain the domain imbalances present in the original dataset, $\mathcal{D}^{\text{train}}$, where certain domains, such as `restaurant`, are more prevalent than others, like `taxi`.

4.1.5 Dialogue State Ontology

Our dialogue management leverages an ontology based on the MultiWOZ 2.2 dataset. This ontology comprises eight domains, each with slots that encapsulate the necessary details for carrying out a conversation within that domain. The slots are divided into two types - non-categorical and categorical.

Non-categorical slots have a large or dynamic set of possible values, and there is no pre-defined list for these slots. Instead, their values are extracted from the dialogue history. On the other hand, *categorical slots* naturally take a small finite set of values. In every domain, we also include common slots `intent` and `requested` that MultiWOZ 2.2 introduced as a new feature (Sebsection 4.1.3) to help us keep track of the user's intents and requested slots. A list of possible slots for each domain in the MultiWOZ 2.2 dataset is provided in Table 4.2.

Domain	**Categorical Slots**	**Non-Categorical Slots**	**Common Slots**
restaurant	pricerange, area, bookday, bookpeople	food, name, booktime	intent, requested
attraction	area, type	name	intent, requested
hotel	pricerange, parking, internet, stars, area, type, bookpeople, bookday, bookstay	name	intent, requested
taxi	-	departure, destination, arriveby, leaveat	intent, requested
train	destination, departure, day, bookpeople	arriveby, leaveat	intent, requested
bus	day	departure, destination, leaveat	intent, requested
hospital	-	department	intent, requested
police	-	name	intent, requested

Table 4.2: The slots for each domain in the MultiWOZ 2.2 dataset.

Since the `police` domain has very few dialogues in the training set (145 dialogues), the number of possible slot values in this domain does not reflect the proper attributes of the slots. We classify them by referring to similar slots in different domains instead, following the approach used by Zang et al. (2020). Therefore, our dialogue state does not contain a domain, domain state pair (d^k, S^k) for police.

4.1.6 Supported Actions

Our experiments focus on delexicalized actions for the action selection model represented as dialogue acts with the form `[domain]-[intent]([slot])` such as `restaurant-inform(name)`. All actions are lowercase and do not always involve a slot, like `general-reqmore`, `general-bye`, or `booking-inform`. We denote the initial set of all possible actions gathered from the $\mathcal{D}^{\text{train}}$ as $\mathcal{A}^{\text{train}}$. It consists of 248 actions, i.e., $|\mathcal{A}^{\text{train}}| = 248$. Appendix A.1 contains all actions and their support.

Our focus is on actions that occur frequently enough in the MultiWOZ 2.2 dataset, specifically, those appearing in at least K turns (with $K = 10$) within the training set. This approach filters out less frequent actions and leaves us with a set of supported actions, denoted as $\mathcal{A}$. In the context of training data subsets, if we refer to all possible and supported actions for a subset of size r, we use the notation $\mathcal{A}_r^{\text{train}}$ and $\mathcal{A}_r$, respectively.

After filtering out actions with less than K occurrences in $\mathcal{A}^{\text{train}}$, we have $|\mathcal{A}| = |\mathcal{A}_{1.0}| = 210$. Examples of filtered actions are `hotel-select(phone)` with support 1, `restaurant-select(address)` with support 5, or action with support 8, such as `train-offerbook(duration)`.

4.2 Dialogue State Tracking Metrics

Let's denote the ground-truth dialogue state as S and the predicted dialogue state as $\hat{S}$, where

$$S = \left\{(d^k, S^k)\right\}_{k=1}^{n} \quad \text{and} \quad \hat{S} = \left\{(\hat{d}^k, \hat{S}^k)\right\}_{k=1}^{n}, \tag{4.1}$$

where for each $k \in 1, 2, ..., n$,

$$S^k = \left\{(s_i^k, v_i^k)\right\}_{i=1}^{m_k} \quad \text{and} \quad \hat{S}^k = \left\{(\hat{s}_i^k, \hat{v}_i^k)\right\}_{i=1}^{m_k}. \tag{4.2}$$

In the above equations, n is the number of domains, m_k is the number of slots in the k-th domain, d^k and $\hat{d}^k$ are the ground-truth and predicted domain names, respectively, s_i^k and $\hat{s}_i^k$ are the ground-truth and predicted slot names, respectively, and v_i^k and $\hat{v}_i^k$ are the ground-truth and predicted values respectively.

Let's define the test dataset $\mathcal{D}^{\text{test}}$, which consists of L dialogue turns. Each turn l has a ground-truth dialogue state S_l and a predicted dialogue state $\hat{S}_l$. Based on this structure, we compute the following evaluation metrics for dialogue state tracking.

4.2.1 Domain Level Metrics

For each domain d^k, we compute the True Positives (TP), False Positives (FP), and False Negatives (FN) across the $\mathcal{D}^{\text{test}}$ as follows:

$$\begin{aligned} TP_{d^k} &= |\{l : d^k \in S_l \text{ and } d^k \in \hat{S}_l\}|, \\ FP_{d^k} &= |\{l : d^k \notin S_l \text{ and } d^k \in \hat{S}_l\}|, \\ FN_{d^k} &= |\{l : d^k \in S_l \text{ and } d^k \notin \hat{S}_l\}|. \end{aligned}$$

Then we can compute precision, recall, and F1-score for each domain as:

$$P_{d^k} = \frac{TP_{d^k}}{TP_{d^k} + FP_{d^k}}, \tag{4.3}$$

$$R_{d^k} = \frac{TP_{d^k}}{TP_{d^k} + FN_{d^k}}, \tag{4.4}$$

$$F1_{d^k} = 2 \cdot \frac{P_{d^k} \cdot R_{d^k}}{P_{d^k} + R_{d^k}}. \tag{4.5}$$

4.2.2 Slot Level Metrics

Similar to domain level metrics, for each slot s_i^k in each domain d^k, we first compute TP, FP, and FN across the dataset:

$$TP_{s_i^k} = |\{l : (s_i^k, v_i^k) \in S_l^k \text{ and } (s_i^k, v_i^k) \in \hat{S}_l^k\}|,$$
$$FP_{s_i^k} = |\{l : (s_i^k, v_i^k) \notin S_l^k \text{ and } (s_i^k, v_i^k) \in \hat{S}_l^k\}|,$$
$$FN_{s_i^k} = |\{l : (s_i^k, v_i^k) \in S_l^k \text{ and } (s_i^k, v_i^k) \notin \hat{S}_l^k\}|.$$

The precision, recall, and F1-score for each slot are calculated as follows:

$$P_{s_i^k} = \frac{TP_{s_i^k}}{TP_{s_i^k} + FP_{s_i^k}},$$
$$R_{s_i^k} = \frac{TP_{s_i^k}}{TP_{s_i^k} + FN_{s_i^k}},$$
$$F1_{s_i^k} = 2 \cdot \frac{P_{s_i^k} \cdot R_{s_i^k}}{P_{s_i^k} + R_{s_i^k}}.$$

4.2.3 Global Slot Level Metrics

In addition to individual domain and slot level metrics, we can calculate the global metrics by accumulating the results from all individual slots:

$$TP_{\text{global}} = \sum_{k=1}^{n} \sum_{i=1}^{m_k} TP_{s_i^k},$$
$$FP_{\text{global}} = \sum_{k=1}^{n} \sum_{i=1}^{m_k} FP_{s_i^k},$$
$$FN_{\text{global}} = \sum_{k=1}^{n} \sum_{i=1}^{m_k} FN_{s_i^k},$$

where n is the total number of domains, m_k is the number of slots in the k-th domain, and $TP_{s_i^k}$, $FP_{s_i^k}$, and $FN_{s_i^k}$ are the true positive, false positive, and false negative counts for the i-th slot in the k-th domain, respectively.

With these global counts, the overall precision, recall, and F1-score are calculated as follows:

$$
\begin{aligned}
P_{\text{global}} &= \frac{TP_{\text{global}}}{TP_{\text{global}} + FP_{\text{global}}}, \\
R_{\text{global}} &= \frac{TP_{\text{global}}}{TP_{\text{global}} + FN_{\text{global}}}, \\
F1_{\text{global}} &= 2 \cdot \frac{P_{\text{global}} \cdot R_{\text{global}}}{P_{\text{global}} + R_{\text{global}}}.
\end{aligned}
$$

4.2.4 Joint Goal Accuracy

Joint goal accuracy (JGA) is computed as the proportion of turns where the entire dialogue state is correctly predicted:

$$
JGA = \frac{|\{l : S_l = \hat{S}_l\}|}{L}. \tag{4.6}
$$

4.3 Action Selection Metrics

In the action selection task, we aim to predict a set of actions for each dialogue turn. We denoted the set of all possible actions as $\mathcal{A}$. Suppose we have a test dataset $\mathcal{D}^{\text{test}}$ of L dialogue turns, where each turn l consists of a predicted action set $\hat{A}_l \subset \mathcal{A}$ and the corresponding ground-truth action set $A_l \subset \mathcal{A}$.

4.3.1 Action Level Metrics

For each possible action a in the action set $\mathcal{A}$ we calculate True Positives (TP), False Positives (FP), and False Negatives (FN) across the $\mathcal{D}^{\text{test}}$ as follows:

$$
\begin{aligned}
TP_a &= |l : a \in A_l \text{ and } a \in \hat{A}_l|, \\
FP_a &= |l : a \notin A_l \text{ and } a \in \hat{A}_l|, \\
FN_a &= |l : a \in A_l \text{ and } a \notin \hat{A}_l|.
\end{aligned}
$$

We can then calculate precision, recall, and F1-score for each action as:

$$
\begin{aligned}
P_a &= \frac{TP_a}{TP_a + FP_a}, \\
R_a &= \frac{TP_a}{TP_a + FN_a}, \\
F1_a &= 2 \cdot \frac{P_a \cdot R_a}{P_a + R_a}.
\end{aligned}
$$

4.3.2 Turn Level Accuracy

We can also define an accuracy metric on a turn level, where we consider a turn as correct if the entire action set is predicted correctly:

$$ACC = \frac{|l : \hat{A}_l = A_l|}{L}. \tag{4.7}$$

In this case, the predicted action set $\hat{A}_l$ is considered correct if it is precisely the same as the ground-truth action set A_l.

4.3.3 Macro Averaged Metrics

Additionally, *macro-averaged* versions of the precision, recall, and F1-score can be calculated. These metrics compute the metric independently for each action and then take the average (hence treating all actions equally):

$$\begin{aligned}
P_{\text{macro}} &= \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} P_a \\
R_{\text{macro}} &= \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} R_a, \\
F1_{\text{macro}} &= \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} F1_a.
\end{aligned}$$

4.3.4 Weighted Averaged Metrics

The *weighted-averaged* versions of the precision, recall, and F1-score can be calculated by giving more importance to the more frequent actions in the dataset. The number of instances of that action in the dataset weights each action's metric. Let N_a denote the total number of instances of action a in the dataset, and N denotes the total number of instances. The weighted averages are defined as follows:

$$\begin{aligned}
P_{\text{weighted}} &= \frac{1}{N} \sum_{a \in \mathcal{A}} N_a \cdot P_a, \\
R_{\text{weighted}} &= \frac{1}{N} \sum_{a \in \mathcal{A}} N_a \cdot R_a, \\
F1_{\text{weighted}} &= \frac{1}{N} \sum_{a \in \mathcal{A}} N_a \cdot F1_a.
\end{aligned}$$

In these formulations, actions with more instances contribute more to the overall metrics, making them beneficial when dealing with datasets with imbalanced action distributions.

4.4 Model Training Details

We use the HuggingFace Transformers library[1] (Wolf et al., 2020) to implement our models. We employ the 'google/flan-t5-base'[2] model for dialogue state tracking and action generation. The FLAN T5 model (Chung et al., 2022) is a variant

[1] `https://github.com/huggingface/transformers`
[2] `https://huggingface.co/google/flan-t5-base`

of the T5 (Raffel et al., 2020). Its configuration comprises 12 Transformer encoder and decoder layers, 12 attention heads, and a hidden size of 768. It has a maximum positional embedding of 512, defining an upper limit on input size. We further decreased the maximum input length to 260 tokens and the output length to 230 tokens, as no examples in our dataset exceeded this limit. Any input exceeding this limit is truncated to ensure compatibility. The dropout rate for this model is 0.1. We use a learning rate of 10^{-4}.

For the classification approach foraction selection, we utilize the 'roberta-base'[3] model. RoBERTa (Liu et al., 2019) is an improved version of BERT (Devlin et al., 2019). It includes 12 Transformer layers, 12 attention heads, and a hidden size 768, with the same maximum input size and dropout rate as the FLAN T5 model. We use a learning rate of $2 \cdot 10^{-5}$.

Both models are trained with the Adam optimizer (Kingma and Ba, 2017), an efficient and widely-used method for training deep neural networks. Scripts used for training the models are described in Appendix A.3. The trained models are available at HuggingFace hub.[4]

[3] `https://huggingface.co/roberta-base`

[4] `https://huggingface.co/jaroslavsafar`

5. Results and Discussion

In this chapter, we present the results of our evaluation using both automatic metrics (see Section 4.2, 4.3) and human evaluation. We present the results in two main sections: results for dialogue state tracking in Section 5.1 and for action selection in Section 5.2. The final Section 5.3 is dedicated to manual analysis.

5.1 Dialogue State Tracking Results

In this section, we present the performance of our dialogue state tracking (DST) models, trained on various subsets of the available training dataset. Each model is denoted as flan-t5-base-DST(r), where the parameter r signifies the relative size of the training set $\mathcal{D}_r^{\text{train}}$ used, as explained in Section 4.1.4.

Performance of baseline DST models: SGD-baseline (Rastogi et al., 2017) and TRADE (Wu et al., 2019), on MultiWOZ 2.2 dataset as described by Zang et al. (2020) are summarized in Table 5.1.

We evaluate the models using several metrics discussed in Section 4.2. These include joint goal accuracy (JGA), global precision (P_{global}), global recall (R_{global}), and global F1 score ($\text{F1}_{\text{global}}$). Table 5.2 summarizes our models' performance on the test dataset $\mathcal{D}^{\text{test}}$. As we can see, our approach is effective, as indicated by the high scores across all metrics. As expected, the flan-t5-base-DST(1.0) model, trained on the full training set $\mathcal{D}_{1.0}^{\text{train}}$, achieved the highest scores across all metrics. However, both models trained on smaller datasets - specifically, flan-t5-base-DST(0.5) and flan-t5-base-DST(0.3) - also showcase impressive performance. The minimal decrease in scores as the size of the training set decreases emphasizes our models' robustness in data scarcity conditions.

This good performance is likely due to the power of the pre-trained Flan-T5 language model, which captures complex patterns in dialogue data. Even when the training dataset is pruned, the model performs effectively in dialogue state tracking due to its pre-training on a large corpus. This resilience against the size of the training set demonstrates our approach's potential in situations where the availability of training data might be limited. All the evaluation scripts, metrics, and results are in the attached files described in Appendix A.3.

Model	**JGA**
TRADE	0.454
SGD-baseline	0.420

Table 5.1: Performance of baseline DST models based on the JGA metric. Source: Table 4 of Zang et al. (2020)

5.2 Action Selection Results

In this section, we present the results of the action selection models trained on various subsets of the training data. Each model is referred to as flan-t5-base-AS(r)

Model	**JGA**	$\mathbf{P_{global}}$	$\mathbf{R_{global}}$	$\mathbf{F1_{global}}$
flan-t5-base-DST(1.0)	0.736	0.984	0.969	0.977
flan-t5-base-DST(0.5)	0.723	0.983	0.968	0.975
flan-t5-base-DST(0.3)	0.702	0.981	0.965	0.973

Table 5.2: Performance of DST models trained on different subsets of the training set. JGA denotes joint goal accuracy, P_{global} refers to global precision, R_{global} is global recall, and $F1_{\text{global}}$ stands for global F1 score. All models are evaluated with ground-truth dialogue state from the previous turn S_{t-1} on the input.

or roberta-base-AS(r), for generative or classification based models respectively, where r indicates the relative size of the training set $\mathcal{D}_r^{\text{train}}$ used.

The models were evaluated using metrics described in Section 4.3, including turn level accuracy (ACC), weighted average versions of precision, recall, and F1 score (P_w, R_w, F1_w), and macro average versions of the same metrics (P_m, R_m, F1_m). The performances of these models on the test dataset $\mathcal{D}^{\text{test}}$ are summarized in Table 5.3.

Model	**ACC**	$\mathbf{P_w}$	$\mathbf{R_w}$	$\mathbf{F1_w}$	$\mathbf{P_m}$	$\mathbf{R_m}$	$\mathbf{F1_m}$
flan-t5-base-AS(1.0)	0.197	0.517	0.480	0.465	0.281	0.228	0.226
flan-t5-base-AS(0.5)	0.192	0.503	0.489	0.466	0.257	0.230	0.220
flan-t5-base-AS(0.3)	0.198	0.509	0.491	0.466	0.261	0.232	0.224
flan-t5-base-AS(1.0)*	0.194	0.502	0.471	0.455	0.263	0.221	0.217
flan-t5-base-AS(0.5)*	0.190	0.496	0.484	0.461	0.249	0.226	0.215
flan-t5-base-AS(0.3)*	0.194	0.496	0.480	0.455	0.249	0.224	0.215
roberta-base-AS(1.0)	0.184	0.515	0.425	0.449	0.227	0.161	0.176
roberta-base-AS(0.5)	0.168	0.499	0.421	0.446	0.218	0.162	0.178
roberta-base-AS(0.3)	0.146	0.461	0.340	0.373	0.175	0.109	0.125
roberta-base-AS(1.0)*	0.182	0.507	0.422	0.444	0.217	0.159	0.172
roberta-base-AS(0.5)*	0.167	0.490	0.413	0.436	0.218	0.159	0.175
roberta-base-AS(0.3)*	0.146	0.461	0.337	0.370	0.177	0.108	0.124

Table 5.3: Performance of action selection models trained on different subsets of the training set. ACC denotes turn level accuracy, P_w, R_w, F1_w refer to the weighted average versions of precision, recall, and F1 score respectively, and P_m, R_m, F1_m are the macro average versions of the same metrics. The models marked with * are evaluated with the input containing the dialogue state generated from the corresponding DST model flan-t5-base-DST trained on the same dataset (with the same r), representing the actual use in the real setting. Models without * are evaluated in a standard way, with the input containing only the ground-truth values.

It is crucial to highlight that the task of action selection is challenging. The full training dataset $\mathcal{D}_{1.0}^{\text{train}}$ encompasses 248 possible actions. As described in Section 4.1.6, we pruned the action space to use only those actions with the support of at least 10, resulting in 210 actions. For smaller datasets $\mathcal{D}_{0.5}^{\text{train}}$ and $\mathcal{D}_{0.3}^{\text{train}}$, the action space is typically pruned more, but the overall action set remains large, with at least 200 actions. Thus, selecting actions is difficult from the point of view of machine learning tasks. Particularly in the classification setting as a multi-label classification into more than 200 classes. Hence, the metric scores of

the action selection models is not as high as those of the DST models.

When comparing the models' performances, we note a few significant patterns. Firstly, as expected, both generative (flan-t5) and classification (roberta-base) models perform better when trained on a larger dataset, which suggests that having more training data is beneficial.

When comparing models trained on the same subset of data, the generation models consistently outperform the classification models across all metrics. This can be attributed to the generation model's capacity to generate new sequences, making them more versatile and capable of handling unseen situations during training.

Another interesting observation is the comparison between models evaluated with ground-truth dialogue state and the ones evaluated with the dialogue state generated from the corresponding DST model (models denoted *) that was trained on the same train dataset $\mathcal{D}_r^{\text{train}}$. As expected, models evaluated with the ground-truth dialogue state perform better. The drop in performance for models using generated states highlights the cumulative error in a dialogue system where the output of one module serves as the input for the next. However, the relatively small difference between the performances underlines our models' robustness and ability to operate despite the uncertainty introduced by previous modules.

Notably, the ACC metric used in our evaluation is a strict one. It expects an exact match between the predicted and the ground-truth action sets. The ACC value approaching 20% for the flan-t5-base-AS(1.0) is a good result considering this context. The distribution of actions is also incredibly unbalanced, introducing additional complexity into the learning process. The details about the support for each action are provided in the Appendix A.1, and all the metrics and results are in the attached files described in Appendix A.3.

In conclusion, while the task of action selection in the setting of MultiWOZ with a considerable action space is challenging, the performance of the proposed models shows their ability to tackle this complex task.

5.3 Manual Analysis

We conducted a human evaluation of a subset of the model's predictions and compared them to the ground truth, which allowed us to understand the qualitative performance of the models. One such analysis is shown in the Appendix A.2.

Dialogue State Tracking

The generative DST modes are generally very good at predicting the state of the dialogue based on the user's utterance, context, and the previous state. They can understand and keep track of complex conversations involving multiple topics (like booking a train, finding a hotel, or making restaurant reservations). They correctly identify and encode crucial information (like dates, destinations, and preferences) in the predicted dialogue state. We encountered only a few examples where the prediction was wrong, and the mistake was usually only at one slot, allowing the overall dialogue state to represent the information very well.

Action Selection

Predicting the appropriate system actions is a more challenging task. Both generative and classification models do reasonably well, but they often miss some actions or predict extraneous ones. The classification model is more accurate in predicting necessary actions than the generative model. The generative model sometimes predicts actions that provide more information than the user has requested, indicating a potential over-generation issue.

Both models struggle with complex user utterances involving multiple requests or steps. The models often miss some actions required to address these complex queries fully. This suggests the need for improvement in handling nuanced or multi-step user requests. A key limitation is that the models only sometimes fully understand the context or user intent, leading to missed or incorrect actions. For example, they might fail to provide necessary information or request more information without reason. Also, the models occasionally fail to adapt to the conversational flow and produce actions that feel somewhat out of place. Such behavior is expected because the models were trained in a supervised setting, and their training goal was to predict actions at a turn level. Finetuning the models using RL techniques to create a full policy would undoubtedly improve the performance of action selection models.

6. Conclusion

In this thesis, we presented a study on using pretrained neural architectures for practical dialogue management. The primary aim was to study approaches to dialogue management for practical applications, focusing on two fundamental modules of dialogue systems: dialogue state tracking (DST) and action selection. Three models were proposed and evaluated, trained on varying dataset sizes which allowed an exploration of the model's performance under data-limited conditions.

For the DST task, we introduced the generative model flan-t5-base-DST(1.0), which utilized the robustness of the T5 pretrained language model (Raffel et al., 2020; Chung et al., 2022) to generate a custom text representation of the current dialogue state based on the user utterance, context, and text representation of the previous dialogue state. This model achieved impressive results, achieving a joint goal accuracy (JGA) of 74% (Table 5.2), the highest across all evaluated models, and demonstrated a remarkable ability to process and encode complex details from the current user utterance, context, and previous dialogue state, consequently generating an accurate, updated dialogue state.

The action selection task contained two separate models: a generative model flan-t5-base-AS(1.0) also based on the T5 pretrained language model (Raffel et al., 2020; Chung et al., 2022), and a classification RoBERTa (Liu et al., 2019) based model roberta-base-AS(1.0). These models used similar input as the DST model, specifically, the user utterance, context, the current dialogue state (which would, in a real-world setting, be the output of the DST model), and database count information to select delexicalized actions to execute by the system. Despite the inherent challenges of the task, given the large number of possible actions and the complexity of the conversational context, both models showed reasonable performance.

While the models demonstrated potential in handling the complexities of dialogue management, their limitations are equally vital for subsequent research. The task of action selection, in particular, showed that these models struggle with complex or multi-step user requests. They are prone to miss or misinterpret actions that are not immediately apparent or direct, leading to a less-than-optimal response. This observation reveals the necessity of incorporating tools for more holistic and forward-looking approaches to handling dialogues.

Using supervised learning to train our models was an intended choice driven by our objective to build practical dialogue management. This approach is commonly employed as an initial step in creating practical dialogue systems, providing a foundation that can be further improved with other techniques. One area for further research is the application of reinforcement learning (RL) to improve the performance of action selection models, transforming them into policies that can look ahead to achieve task completion. The RL approach, however, requires a fully functioning dialogue system capable of interacting with users or simulations to train. A well-performing supervised model as a starting point is also a preferred scenario. Another possible area for future research is exploring the use of lexicalized actions. Delexicalized actions, while beneficial for the current scope of the work, contain only part of the complete information about the action. However, a model that predicts delexicalized actions well is a prerequisite for an

effective lexicalization process.

In conclusion, this thesis demonstrated the potential and limitations of applying pretrained language models to practical dialogue management. The proposed models, especially for DST, have achieved notable success and show promise for future applications and improvements.

Bibliography

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. URL https://arxiv.org/abs/1607.06450.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.

Antoine Bordes, Y-Lan Boureau, and Jason Weston. Learning end-to-end goal-oriented dialog. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=S1Bb3D5gg.

Hayet Brabra, Marcos Baez, Boualem Benatallah, Walid Gaaloul, Sara Bouguelia, and Shayan Zamanirad. Dialogue management in conversational systems: a review of approaches, challenges, and opportunities. *IEEE Transactions on Cognitive and Developmental Systems*, pages 1–15, 2022. doi: 10.1109/TCDS.2021.3086565. URL https://hal.science/hal-03626466.

Petter Brandtzaeg and Asbjørn Følstad. Why people use chatbots. 11 2017. ISBN 978-3-319-70283-4. doi: 10.1007/978-3-319-70284-1_30.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Jack Chess, Jack Clark, Chris Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. URL https://arxiv.org/abs/2005.14165.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. MultiWOZ - a large-scale multi-domain Wizard-of-Oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5016–5026, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1547. URL https://aclanthology.org/D18-1547.

Guan-Lin Chao and Ian R. Lane. BERT-DST: Scalable end-to-end dialogue state tracking with bidirectional encoder representations from transformer. *CoRR*, abs/1907.03040, 2019. URL http://arxiv.org/abs/1907.03040.

Hongshen Chen, Xiaorui Liu, Dawei Yin, and Jiliang Tang. A survey on dialogue systems: Recent advances and new frontiers. *SIGKDD Explor. Newsl.*, 19 (2):25–35, nov 2017. ISSN 1931-0145. doi: 10.1145/3166054.3166058. URL https://doi.org/10.1145/3166054.3166058.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL https://aclanthology.org/D14-1179.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models, 2022. URL https://arxiv.org/abs/2210.11416.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990. ISSN 0364-0213. doi: https://doi.org/10.1016/0364-0213(90)90002-E. URL https://www.sciencedirect.com/science/article/pii/036402139090002E.

Mihail Eric, Rahul Goel, Shachi Paul, Abhishek Sethi, Sanchit Agarwal, Shuyang Gao, Adarsh Kumar, Anuj Goyal, Peter Ku, and Dilek Hakkani-Tur. MultiWOZ 2.1: A consolidated multi-domain dialogue dataset with state corrections and state tracking baselines. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 422–428, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://aclanthology.org/2020.lrec-1.53.

Milica Gasic, Filip Jurcicek, Blaise Thomson, Kai Yu, and Steve Young. Online policy optimisation of spoken dialogue systems via live interaction with human subjects. *2011 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2011, Proceedings*, 12 2011. doi: 10.1109/ASRU.2011.6163950.

Milica Gašić and Steve Young. Effective handling of dialogue state in the hidden information state POMDP-based dialogue manager. *ACM Trans. Speech Lang. Process.*, 7(3), jun 2011. ISSN 1550-4875. doi: 10.1145/1966407.1966409. URL https://doi.org/10.1145/1966407.1966409.

Rahul Goel, Shachi Paul, Tagyoung Chung, Jérémie Lecomte, Arindam Mandal, and Dilek Hakkani-Tür. Flexible and scalable state tracking framework for goal-oriented dialogue systems. *CoRR*, abs/1811.12891, 2018. URL http://arxiv.org/abs/1811.12891.

Rahul Goel, Shachi Paul, and Dilek Hakkani-Tür. HyST: A hybrid approach for flexible and accurate dialogue state tracking. *CoRR*, abs/1907.00883, 2019. URL http://arxiv.org/abs/1907.00883.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Michael Heck, Carel van Niekerk, Nurul Lubis, Christian Geishauser, Hsien-Chin Lin, Marco Moresi, and Milica Gasic. TripPy: A Triple Copy Strategy for Value Independent Neural Dialog State Tracking. In *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 35–44, 1st virtual meeting, July 2020. Association for Computational Linguistics. URL https://aclanthology.org/2020.sigdial-1.4.

Matthew Henderson, Blaise Thomson, and Steve Young. Deep Neural Network Approach for the Dialog State Tracking Challenge. In *Proceedings of the SIGDIAL 2013 Conference*, pages 467–471, Metz, France, August 2013. Association for Computational Linguistics. URL https://aclanthology.org/W13-4073.

Matthew Henderson, Blaise Thomson, and Steve Young. Robust dialog state tracking using delexicalised recurrent neural networks and unsupervised adaptation. In *2014 IEEE Spoken Language Technology Workshop (SLT)*, pages 360–365, 2014. doi: 10.1109/SLT.2014.7078601.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/ioffe15.html.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12), mar 2023. ISSN 0360-0300. doi: 10.1145/3571730. URL https://doi.org/10.1145/3571730.

Filip Jurčíček, Blaise Thomson, and Steve Young. Natural Actor and Belief Critic: Reinforcement Algorithm for Learning Parameters of Dialogue Systems Modelled as POMDPs. *ACM Trans. Speech Lang. Process.*, 7(3), jun 2011. ISSN 1550-4875. doi: 10.1145/1966407.1966411. URL `https://doi.org/10.1145/1966407.1966411`.

J. F. Kelley. An iterative design methodology for user-friendly natural language office information applications. *ACM Trans. Inf. Syst.*, 2(1):26–41, jan 1984. ISSN 1046-8188. doi: 10.1145/357417.357420. URL `https://doi.org/10.1145/357417.357420`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184. IEEE, 1995.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in neural information processing systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`. Commonly referred to as the ”AlexNet” paper.

Jonáš Kulhánek, Vojtěch Hudeček, Tomáš Nekvinda, and Ondřej Dušek. AuGPT: Auxiliary tasks and data augmentation for end-to-end dialogue with pre-trained language models. In *Proceedings of the 3rd Workshop on Natural Language Processing for Conversational AI*, pages 198–210, Online, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.nlp4convai-1.19. URL `https://aclanthology.org/2021.nlp4convai-1.19`.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

Chia-Hsuan Lee, Hao Cheng, and Mari Ostendorf. Dialogue state tracking with a language model using schema-driven prompting. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4937–4949, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.404. URL `https://aclanthology.org/2021.emnlp-main.404`.

Wenqiang Lei, Xisen Jin, Min-Yen Kan, Zhaochun Ren, Xiangnan He, and Dawei Yin. Sequicity: Simplifying task-oriented dialogue systems with single sequence-to-sequence architectures. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1437–1447, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1133. URL `https://aclanthology.org/P18-1133`.

Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz. End-to-end task-completion neural dialogue systems. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 733–743, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing. URL `https://aclanthology.org/I17-1074`.

Zhaojiang Lin, Andrea Madotto, Genta Indra Winata, and Pascale Fung. MinTL: Minimalist transfer learning for task-oriented dialogue systems. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3391–3405, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.273. URL `https://aclanthology.org/2020.emnlp-main.273`.

Zachary Lipton, Xiujun Li, Jianfeng Gao, Lihong Li, Faisal Ahmed, and li Deng. BBQ-Networks: Efficient exploration in deep reinforcement learning for task-oriented dialogue systems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32, 11 2017. doi: 10.1609/aaai.v32i1.11946.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL `http://arxiv.org/abs/1907.11692`.

Ilya Loshchilov and Frank Hutter. Loshchilov, ilya and hutter, frank. *CoRR*, abs/1711.05101, 2017. URL `http://arxiv.org/abs/1711.05101`.

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, 1999.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Sarah McLeod, Ivana Kruijff-Korbayova, and Bernd Kiefer. Multi-task learning of system dialogue act selection for supervised pretraining of goal-oriented dialogue policies. In *Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue*, pages 411–417, Stockholm, Sweden, September 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5947. URL `https://aclanthology.org/W19-5947`.

Michael McTear. *Conversational AI Dialogue Systems, Conversational Agents, and Chatbots*. Morgan & Claypool Publishers, 2021.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL `https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gašić, Pei-hao Su, David Vandyke, Tsung-Hsien Wen, and Steve J. Young. Multi-domain Dialog State Tracking using Recurrent Neural Networks. *CoRR*, abs/1506.07190, 2015. URL http://arxiv.org/abs/1506.07190.

Baolin Peng, Chunyuan Li, Jinchao Li, Shahin Shayandeh, Lars Liden, and Jianfeng Gao. Soloist: Building task bots at scale with transfer learning and machine teaching. *Transactions of the Association for Computational Linguistics*, 9:807–824, 2021. doi: 10.1162/tacl_a_00399. URL https://aclanthology.org/2021.tacl-1.49.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL https://aclanthology.org/D14-1162.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *OpenAI blog*, 2018. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1), jan 2020. ISSN 1532-4435. URL https://www.jmlr.org/papers/volume21/20-074/20-074.pdf.

Janarthanan Rajendran, Jatin Ganhotra, Satinder Singh, and Lazaros Polymenakos. Learning End-to-End Goal-Oriented Dialog with Multiple Answers. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3834–3843, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1418. URL https://aclanthology.org/D18-1418.

Abhinav Rastogi, Dilek Hakkani-Tür, and Larry P. Heck. Scalable Multi-Domain Dialogue State Tracking. *CoRR*, abs/1712.10224, 2017. URL http://arxiv.org/abs/1712.10224.

Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

Alexander I Rudnicky, Eric H Thayer, Paul Constantinides, Chris Tchou, Ravi Shern, Kevin A Lenzo, Wei Xu, and Alice Oh. Creating natural dialogs in the carnegie mellon communicator system. In *Proc. 6th European Conference on Speech Communication and Technology (Eurospeech 1999)*, pages 1531–1534, 1999. doi: 10.21437/Eurospeech.1999-344.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

Iulian V. Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. Building End-to-End Dialogue Systems Using Generative Hierarchical Neural Network Models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 3776–3783. AAAI Press, 2016.

Bayan Shawar and Eric Atwell. Chatbots: Are they really useful? *LDV Forum*, 22:29–49, 07 2007. doi: 10.21248/jlcl.22.2007.88.

Pei-Hao Su, Milica Gašić, Nikola Mrkšić, Lina Maria Rojas-Barahona, Stefan Ultes, David Vandyke, Tsung-Hsien Wen, and Steve J. Young. Continuously learning neural dialogue management. *CoRR*, abs/1606.02689, 2016. URL http://arxiv.org/abs/1606.02689.

Pei-Hao Su, Paweł Budzianowski, Stefan Ultes, Milica Gašić, and Steve Young. Sample-efficient actor-critic reinforcement learning with supervised data for dialogue management. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 147–157, Saarbrücken, Germany, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-5518. URL https://aclanthology.org/W17-5518.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Yuxuan Wang, R. J. Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc V. Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif A. Saurous. Tacotron: A fully end-to-end text-to-speech synthesis model. *CoRR*, abs/1703.10135, 2017. URL http://arxiv.org/abs/1703.10135.

Joseph Weizenbaum. ELIZA — a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1): 36–45, jan 1966. ISSN 0001-0782. doi: 10.1145/365153.365168. URL `https://doi.org/10.1145/365153.365168`.

Jason Williams, Antoine Raux, and Matthew Henderson. The dialog state tracking challenge series: A review. *Dialogue & Discourse*, April 2016. URL `https://www.microsoft.com/en-us/research/publication/the-dialog-state-tracking-challenge-series-a-review/`.

Jason D Williams and Steve Young. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech & Language*, 21(2):393–422, 2007. ISSN 0885-2308. doi: https://doi.org/10.1016/j.csl.2006.06.008. URL `https://www.sciencedirect.com/science/article/pii/S0885230806000283`.

Jason D. Williams, Kavosh Asadi, and Geoffrey Zweig. Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 665–677, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1062. URL `https://aclanthology.org/P17-1062`.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL `https://aclanthology.org/2020.emnlp-demos.6`.

Chien-Sheng Wu, Andrea Madotto, Ehsan Hosseini-Asl, Caiming Xiong, Richard Socher, and Pascale Fung. Transferable multi-domain state generator for task-oriented dialogue systems. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 808–819, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1078. URL `https://aclanthology.org/P19-1078`.

Wenhan Xiong, Xiaoxiao Guo, Mo Yu, Shiyu Chang, Bowen Zhou, and William Yang Wang. Scheduled policy optimization for natural language communication with intelligent agents. *CoRR*, abs/1806.06187, 2018. URL `http://arxiv.org/abs/1806.06187`.

Steve Young. Probabilistic methods in spoken dialogue systems. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 358, 12 2000. doi: 10.1098/rsta.2000.0593.

Steve Young, Jost Schatzmann, Karl Weilhammer, and Hui Ye. The hidden information state approach to dialog management. volume 4, pages IV–149, 05 2007. ISBN 1-4244-0728-1. doi: 10.1109/ICASSP.2007.367185.

Steve Young, Milica Gašić, Simon Keizer, François Mairesse, Jost Schatzmann, Blaise Thomson, and Kai Yu. The hidden information state model: A practical framework for POMDP-based spoken dialogue management. *Computer Speech & Language*, 24:150–174, 04 2010. doi: 10.1016/j.csl.2009.04.001.

Dong Yu and Li Deng. *Automatic Speech Recognition: A Deep Learning Approach*. Springer, 2014.

Xiaoxue Zang, Abhinav Rastogi, Srinivas Sunkara, Raghav Gupta, Jianguo Zhang, and Jindong Chen. MultiWOZ 2.2 : A dialogue dataset with additional annotation corrections and state tracking baselines. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pages 109–117, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.nlp4convai-1.13. URL https://aclanthology.org/2020.nlp4convai-1.13.

Tiancheng Zhao, Allen Lu, Kyusong Lee, and Maxine Eskenazi. Generative encoder-decoder models for task-oriented spoken dialog systems with chatting capability. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 27–36, Saarbrücken, Germany, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-5505. URL https://aclanthology.org/W17-5505.

Lukáš Žilka and Filip Jurčíček. LecTrack: Incremental dialog state tracking with Long Short-Term Memory Networks. In *Proceedings of the 18th International Conference on Text, Speech, and Dialogue - Volume 9302*, TSD 2015, page 174–182, Berlin, Heidelberg, 2015. Springer-Verlag. ISBN 9783319240329. doi: 10.1007/978-3-319-24033-6_20. URL https://doi.org/10.1007/978-3-319-24033-6_20.

List of Figures

1.1 Unrolled recurrent neural network 17
1.2 The Gated Recurrent Unit (GRU) 18
1.3 The Transformer architecture 20
1.4 The Transformer multi-head self-attention 22

2.1 The architecture of the BERT-DST model 28
2.2 The architecture of the TripPy model 28
2.3 The architecture of the TRADE model 29
2.4 The overview of generative DST approaches for the multi-domain scenario using the T5 model 30
2.5 Example of generative DST for multi-domain scenario using T5 model 30
2.6 A Hybrid Code Network architecture 32

A.1 Directory structure of the project 87

List of Tables

4.1 MultiWOZ 2.2 dataset split 47
4.2 MultiWOZ 2.2 slots 48

5.1 Performance of Baseline DST models 54
5.2 Performance of DST models 55
5.3 Performance of action selection models 55

A.1 Actions and their support 75

List of Abbreviations

ASR automatic speech recognition. 7, 9

CNN convolutional neural network. 16, 17, 19, 31

DAG directed acyclic graph. 12, 13, 16, 17

DM dialogue management. 4, 8, 26, 30, 31, 34, 39

DNN deep neural network. 12, 13, 15

DST dialogue state tracking. 8, 26, 27, 29, 34, 38, 39, 54, 56, 58, 80

FNN feed-forward neural network. 16, 17

GRU Gated Recurrent Unit. 17, 27, 29

LLM large language model. 24, 27

LM language model. 10, 11

LSTM Long Short-Term Memory. 17, 27, 31

MDPs Markov Decision Processes. 27, 31

MLE maximum likelihood estimation. 11

MLM masked language model. 23–25

NLG natural language generation. 4, 8, 9, 34

NLP natural language processing. 9, 23, 25, 26

NLU natural language understanding. 4, 7–9, 27, 31, 34

POMDPs Partially Observable Markov Decision Processes. 27, 31

RL reinforcement learning. 31, 32, 57, 58

RNN recurrent neural network. 12, 17–19, 27, 32

SGD stochastic gradient descent. 16

TTS Text-To-Speech. 8, 9

List of Glossaries

action selection A sub-component of dialogue management module, which determines the system's response to the user's latest utterance, based on the information present in the updated belief state. 8, 9, 26, 31, 34, 39, 51, 53, 54, 58

automatic speech recognition The process that translates spoken language into written text. 7

backpropagation The backpropagation algorithm is used in artificial neural networks to calculate the gradient of the loss function with respect to the network weights using the chain rule of differentiation. It is a two-step process: a forward pass to calculate the network output and determine the loss, and a backward pass to calculate the gradient of the loss with respect to each weight and adjust the weights to minimize the loss. It is commonly used with the gradient descent optimization algorithm. 12, 16

belief state A data structure maintained by the dialogue state tracking system that represents the dialogue history. It is a dynamic structure, continually updated with new information extracted from user utterances as the conversation progresses. It includes various elements such as slots and their assigned values, user preferences, and past system actions. 26, 27, 31

convolutional neural network Convolutional neural network (CNN) is a class of deep learning neural networks, most commonly applied to analyzing and processing visual inputs such as images. 16

deep neural network A deep neural network is an artificial neural network with multiple layers between the input and output layers. The DNN represents, in the form of a directed acyclic graph, the correct mathematical manipulation to turn the input into the output, whether it be a linear relationship or a non-linear relationship. 12, 13, 16, 27

dialogue act Dialogue acts are the structured semantic representations of an utterance in the context of a conversation created by NLU. They are often structured as an *intent*, which captures the purpose of the utterance, and *slots*, which are specific pieces of information related to the intent. Slots usually have assigned *values*. 7, 8

dialogue context Dialogue context refers to the information from previous utterances that helps to understand and interpret subsequent ones. It provides continuity to the dialogue and enables the system to maintain the coherence and relevance of the conversation. 7

dialogue management The component of a dialogue system responsible for controlling the flow of the conversation. It includes dialogue state tracking and action selection. 4, 8, 26, 27, 34

dialogue state tracking A sub-component of dialogue management. DST is responsible for maintaining a representation of the dialogue's context by managing a data structure known as the 'belief state'. This belief state includes various elements such as slots and their assigned values (extracted through NLU), user preferences, and past system actions. 8, 9, 26, 27, 34, 49, 52, 54, 58

directed acyclic graph A directed acyclic graph (DAG) is a concept from graph theory in mathematics. A DAG is a finite directed graph with no directed cycles. It consists of vertices and edges (ordered 2-tuples of vertices), with each edge directed from one vertex to another so that following those directions will never form a closed loop. 12

embedding In machine learning, embedding is a mapping from discrete objects, such as words, tokens, or items, into a vector of continuous values. This mapping allows these objects to be represented in a way that captures semantic or contextual meaning. In NLP, word embeddings represent words in a high-dimensional space where the distance and direction between words indicate their semantic relationship. 19, 27

feed-forward neural network A feed-forward neural network, or FNN, is an artificial neural network where connections between the nodes do not form a cycle. The information in a feed-forward network moves in only one direction, forward, from the input layer, through the hidden layers, to the output layer. 16

language model A type of statistical model that is used to predict the probability of a sequence of words or generally tokens. 10, 11, 34

logits In machine learning, logits are the raw, non-normalized predictions a model produces. In the context of neural networks, logits often refer to the vector of raw predictions that a classification model generates before passing through an activation function, such as a softmax or sigmoid function, which transforms the logits into probabilities. 20

Markov assumption In the context of language models, the Markov assumption states that the probability of a token in a sequence is dependent only on a fixed number of previous tokens, rather than all the preceding tokens. 10, 12

maximum likelihood A statistical method or function that is used for estimating the values of the parameters of a model from data so that these data are the most probable. 11

n-gram A contiguous sequence of n items, such as words or tokens. N-grams are widely used in natural language processing and computational linguistics. 10–12

natural language generation The part of a dialogue system that transforms the representation of the system's intended response into a coherent and natural-sounding text. 4, 8

natural language processing A field of artificial intelligence that focuses on the interaction between computers and humans through natural language. 9, 23, 26

natural language understanding The component of a dialogue system which interprets the meaning of the input text. 4, 7, 27

recurent neural network Recurrent neural networks (RNNs) are a type of artificial neural network designed to recognize patterns in data sequences, such as text, genomes, or sound. Unlike feed-forward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. 17, 27

speech synthesis Speech synthesis, also known as text-to-speech (TTS), is the process of converting written text into spoken words. This technology is critical for spoken dialogue systems, enabling them to communicate with users in natural language speech. 8, 9

token In the context of natural language processing, a token typically refers to the smallest unit of processing in a language model. Depending on the actual model, a token could represent an individual word, a subword, or even a single character. The process of converting text into such tokens is referred to as tokenization. 10, 19

topological ordering In the context of a DAG, a topological ordering is a linear ordering of its nodes such that for every directed edge (u, v) from node u to node v, u comes before v in the ordering. 13

transformer An architecture introduced in the "Attention is All You Need" paper by Vaswani et al., 2017. It is based on self-attention mechanisms and feed-forward networks. The architecture includes an encoder and a decoder, each composed of a stack of identical layers.. 12, 19, 34, 35, 39

turn In a conversation, a turn refers to the opportunity for a participant to speak or respond. In task-oriented dialogue systems, turns are typically alternated between the user and the system. 7

utterance An utterance is a continuous block of speech or text from one participant in a conversation. It represents a coherent expression of a single thought or idea. 7

A. Attachments

A.1 Action support

Table A.1: Actions and their support.

Action	Support	Supported
general-reqmore	10866	True
general-bye	7288	True
booking-inform	4314	True
booking-book(ref)	3919	True
general-welcome	3882	True
restaurant-inform(name)	3043	True
hotel-inform(choice)	2837	True
train-inform(leaveat)	2584	True
hotel-inform(name)	2495	True
restaurant-inform(choice)	2460	True
train-inform(trainid)	2370	True
restaurant-inform(food)	2359	True
restaurant-inform(area)	2235	True
hotel-inform(type)	2165	True
train-inform(arriveby)	2141	True
train-offerbook	2050	True
attraction-inform(name)	1946	True
restaurant-inform(pricerange)	1890	True
hotel-inform(area)	1738	True
hotel-inform(pricerange)	1680	True
attraction-inform(area)	1651	True
hotel-request(area)	1601	True
attraction-inform(choice)	1591	True
general-greet	1585	True
train-offerbooked(ref)	1552	True
restaurant-inform(address)	1530	True
train-inform(choice)	1480	True
attraction-inform(address)	1469	True
taxi-inform(type)	1464	True
attraction-inform(type)	1463	True
taxi-inform(phone)	1451	True
train-request(leaveat)	1426	True
attraction-inform(entrancefee)	1418	True
restaurant-request(food)	1394	True
hotel-inform(stars)	1380	True
attraction-inform(phone)	1328	True
booking-request(bookday)	1266	True
hotel-recommend(name)	1230	True

Table A.1: Actions and their supports (continued)

Action	Support	Supported
train-inform(destination)	1222	True
attraction-inform(postcode)	1201	True
restaurant-inform(phone)	1194	True
restaurant-recommend(name)	1186	True
train-request(departure)	1177	True
train-request(day)	1173	True
hotel-inform(internet)	1170	True
attraction-recommend(name)	1123	True
train-request(destination)	1101	True
hotel-inform(parking)	1091	True
booking-book(name)	1081	True
train-inform(departure)	1052	True
hotel-request(pricerange)	1020	True
train-offerbooked(price)	1001	True
train-inform(price)	895	True
booking-nobook	894	True
restaurant-request(area)	845	True
hotel-inform(address)	782	True
booking-request(bookpeople)	763	True
train-request(arriveby)	745	True
booking-book(bookday)	744	True
train-inform(day)	735	True
restaurant-inform(postcode)	731	True
train-inform(duration)	730	True
booking-request(booktime)	691	True
booking-request(bookstay)	690	True
restaurant-request(pricerange)	686	True
attraction-request(type)	676	True
restaurant-nooffer	666	True
booking-book(bookpeople)	664	True
attraction-request(area)	646	True
taxi-request(leaveat)	557	True
taxi-request(departure)	542	True
hotel-inform(phone)	538	True
restaurant-nooffer(food)	516	True
hotel-inform(postcode)	512	True
booking-book(booktime)	476	True
train-request(bookpeople)	455	True
hotel-select	451	True
taxi-request(destination)	416	True
restaurant-select	408	True
booking-book(bookstay)	399	True
hotel-nooffer(type)	395	True
train-offerbooked(trainid)	346	True

Table A.1: Actions and their supports (continued)

Action	Support	Supported
hotel-recommend(area)	335	True
attraction-recommend(entrancefee)	327	True
taxi-request(arriveby)	324	True
attraction-recommend(address)	324	True
hotel-request(stars)	312	True
train-offerbooked(bookpeople)	311	True
restaurant-recommend(food)	303	True
hotel-nooffer	297	True
restaurant-recommend(area)	292	True
restaurant-nooffer(area)	292	True
hospital-inform(address)	289	True
attraction-recommend(area)	278	True
hotel-recommend(pricerange)	268	True
attraction-nooffer(area)	267	True
attraction-nooffer(type)	254	True
hospital-inform(phone)	251	True
hospital-inform(postcode)	247	True
hotel-recommend(stars)	244	True
restaurant-recommend(pricerange)	242	True
police-inform(phone)	234	True
hospital-inform(name)	231	True
booking-inform(name)	228	True
train-select	223	True
hotel-select(type)	223	True
police-inform(postcode)	211	True
hotel-request(parking)	210	True
hotel-request(type)	210	True
police-inform(address)	209	True
attraction-select	205	True
hotel-nooffer(area)	195	True
hotel-nooffer(stars)	189	True
train-offerbooked(leaveat)	185	True
taxi-inform(departure)	183	True
restaurant-recommend(address)	182	True
restaurant-select(food)	170	True
hotel-nooffer(pricerange)	167	True
attraction-recommend(type)	163	True
hotel-request(internet)	162	True
hotel-recommend(type)	160	True
hotel-recommend(internet)	156	True
taxi-inform(destination)	154	True
taxi-inform(leaveat)	149	True
train-offerbook(trainid)	145	True
restaurant-nooffer(pricerange)	145	True

Table A.1: Actions and their supports (continued)

Action	**Support**	**Supported**
hotel-recommend(parking)	143	True
attraction-recommend(postcode)	135	True
train-offerbook(leaveat)	128	True
police-inform(name)	126	True
attraction-recommend(phone)	122	True
hotel-recommend(address)	118	True
train-offerbooked(arriveby)	102	True
hotel-select(pricerange)	99	True
taxi-inform(arriveby)	98	True
train-offerbooked(departure)	91	True
attraction-nooffer	91	True
attraction-request(name)	90	True
train-offerbook(arriveby)	89	True
train-offerbooked(destination)	88	True
booking-book	87	True
attraction-select(type)	84	True
train-inform(ref)	83	True
restaurant-select(pricerange)	83	True
hospital-request(department)	79	True
hotel-select(area)	78	True
hospital-inform(department)	75	True
restaurant-select(area)	75	True
restaurant-select(name)	73	True
train-offerbooked(day)	72	True
hotel-request(name)	70	True
booking-nobook(name)	69	True
booking-inform(bookday)	67	True
attraction-request(entrancefee)	64	True
hotel-select(name)	64	True
restaurant-recommend(phone)	60	True
restaurant-inform(ref)	59	True
booking-nobook(bookday)	58	True
booking-inform(bookpeople)	58	True
booking-nobook(booktime)	52	True
train-select(leaveat)	48	True
restaurant-request(name)	46	True
hotel-inform(ref)	46	True
restaurant-recommend(postcode)	46	True
hotel-select(stars)	45	True
train-offerbook(destination)	42	True
general-thank	42	True
hotel-inform	41	True
taxi-inform	39	True
train-nooffer	38	True

Table A.1: Actions and their supports (continued)

Action	Support	Supported
restaurant-inform	38	True
hotel-recommend(postcode)	33	True
train-offerbook(departure)	33	True
train-nooffer(leaveat)	32	True
train-offerbook(bookpeople)	30	True
hotel-recommend(phone)	29	True
booking-nobook(bookstay)	29	True
hotel-select(choice)	28	True
train-offerbooked	28	True
hotel-nooffer(parking)	27	True
attraction-select(area)	27	True
booking-inform(booktime)	26	True
attraction-inform	26	True
attraction-select(name)	26	True
train-offerbook(day)	25	True
hotel-nooffer(internet)	25	True
train-select(arriveby)	24	True
attraction-select(entrancefee)	24	True
booking-inform(bookstay)	23	True
train-inform	22	True
train-select(trainid)	21	True
train-offerbook(price)	20	True
train-nooffer(departure)	18	True
booking-nobook(bookpeople)	17	True
train-nooffer(destination)	17	True
train-select(departure)	16	True
train-select(day)	15	True
restaurant-select(choice)	15	True
restaurant-recommend(choice)	14	True
train-nooffer(day)	14	True
booking-nobook(ref)	13	True
booking-inform(ref)	13	True
attraction-inform(openhours)	12	True
hotel-recommend	12	True
attraction-recommend(choice)	11	True
hotel-select(parking)	11	True
train-select(destination)	11	True
restaurant-recommend	11	True
train-inform(bookpeople)	10	True
hotel-select(internet)	10	True
train-offerbooked(duration)	9	False
hotel-recommend(choice)	9	False
hotel-nooffer(name)	8	False
taxi-request(bookpeople)	8	False

Table A.1: Actions and their supports (continued)

Action	Support	Supported
train-offerbook(choice)	8	False
train-offerbook(duration)	8	False
attraction-recommend	7	False
train-nooffer(arriveby)	6	False
train-select(choice)	5	False
restaurant-select(address)	5	False
attraction-select(choice)	5	False
hotel-nooffer(choice)	4	False
attraction-nooffer(name)	4	False
restaurant-nooffer(name)	4	False
train-offerbook(ref)	3	False
attraction-recommend(openhours)	3	False
train-offerbooked(choice)	3	False
booking-request	2	False
train-nooffer(trainid)	2	False
train-nooffer(choice)	2	False
hotel-select(address)	2	False
taxi-request	2	False
train-select(bookpeople)	2	False
attraction-select(phone)	1	False
restaurant-nooffer(choice)	1	False
hotel-select(phone)	1	False
taxi-request(type)	1	False
train-select(price)	1	False
attraction-nooffer(choice)	1	False
taxi-inform(choice)	1	False
police-inform	1	False
train-request	1	False
taxi-request(bookday)	1	False
attraction-request(phone)	1	False
attraction-select(address)	1	False
restaurant-request	1	False
attraction-nooffer(address)	1	False

A.2 Dialogue Analysis Example

Following is the example and analysis of 3 first turns of a dialogue that shows the effectiveness of the DST model (trained on the entire train dataset) and the shortcomings of action prediction models:

1. **User uterance:**
   "We need to find a guesthouse of moderate price."

Context:
None in this turn, as this is the first user utterance in the dialogue.

- **Reference string state:**
  "hotel - intent: find_hotel, pricerange: moderate, type: guesthouse"
  This is a summary of the key attributes of the hotel that the user wants to find. In this case, they're looking for a hotel, specifically a guesthouse, in a moderate price range.
- **Predicted string state:**
  "hotel - intent: find_hotel, pricerange: moderate, type: guesthouse"
  The predicted state matches the reference state, which means the model correctly understood the user's intent and the specifications they're looking for in a hotel.
- **Reference actions:**
  - hotel-request(area)
  - hotel-request(stars)

  These actions indicate that the system should be requesting more information about the area and star rating of the guesthouse that the user wants to find.
- **Predicted actions using generative model:**
  - hotel-inform(choice)
  - hotel-inform(pricerange)
  - hotel-inform(type)
  - hotel-request(area)

  The predicted actions include informing the user about choices, the price range, and type of guesthouses, and also includes a request for more information about the area, like in the reference actions.
- **Predicted actions using classification model:**
  - hotel-inform(choice)
  - hotel-request(area)

  The classification model predicted actions include informing the user about choices and requesting more information about the area. Unlike the generative model, it doesn't include actions to inform about the price range and type.

2. **User utterance:**
   "I would like it to have a 3 star rating."
   Context:
   "I can help you with that. Do you have any special area you would like to stay? Or possibly a star request for the guesthouse?"
   Here, the system asks for more details about the guesthouse, including the desired star rating. The user responds by expressing a preference for a 3-star guesthouse.

- **Reference string state:**
  "hotel - intent: find_hotel, pricerange: moderate, stars: 3, type: guesthouse"

  This state indicates that the user is still looking for a hotel, with the additional specification that they want a 3-star guesthouse.
- **Predicted textual state:**
  "hotel - intent: find_hotel, pricerange: moderate, stars: 3, type: guesthouse"

  The predicted state matches the reference state, indicating that the model has correctly incorporated the user's newly expressed preference for a 3-star guesthouse.
- **Reference Actions:**
  - booking-inform
  - hotel-inform(area)
  - hotel-inform(name)
  - hotel-inform(stars)
  - hotel-inform(type)

  These actions suggest that the system should provide some information related to booking, and about the area, name, star rating, and type of the hotel.
- **Predicted actions using generative model:**
  - hotel-inform(choice)
  - hotel-inform(pricerange)
  - hotel-inform(stars)
  - hotel-inform(type)
  - hotel-request(area)

  The generative model's predictions are partly aligned with the reference, although it includes an action to request information about the area and doesn't include the action to provide booking information.
- **Predicted actions using classification model:**
  - hotel-inform(choice)
  - hotel-inform(type)

  The classification model's predictions are more limited and do not include actions to provide information about the area, the name, or the star rating of the hotel, nor about booking.

3. **User utterance:**
   "Yes, please, for four people."

   Context:
   "The Bridge Guest House is a 3 star guesthouse. It is in the south area. Would you like to book a room?"

   In response to the system's suggestion and booking query, the user confirms their interest in booking a room for four people at the suggested guesthouse.

- **Reference string state:**
  "hotel - bookpeople: 4, intent: book_hotel, name: bridge guest house, pricerange: moderate, stars: 3, type: guesthouse"

  The reference state indicates that the user intends to book a hotel for four people. The specific hotel, the Bridge Guest House, is a moderately priced 3-star guesthouse.
- **Predicted string state:**
  "hotel - bookpeople: 4, intent: book_hotel, name: bridge guest house, pricerange: moderate, stars: 3, type: guesthouse"

  The predicted state matches the reference state, indicating that the model correctly understood the user's booking intent, hotel choice, and number of people for the booking.
- **Reference actions:**
  - booking-request(bookpeople)

  The reference action indicates that the system should process the request to book a room for four people.
- **Predicted actions using generative model:**
  - booking-request(bookday)
  - booking-request(bookstay)

  The generative model incorrectly predicts actions to request booking day and duration, while the user already confirmed the booking for four people.
- **Predicted actions using classification model:**

  The classification model does not predict any actions, indicating that it has failed to capture the user's intent to book a room for four people.

A.3 Source Code Description and Usage

This section provides a brief overview of the source code used for training and evaluation of our models. For a more detailed understanding of the code, we recommend direct examination of the Python scripts and associated documentation within the source code.

The codebase used for training and evaluation of our models is structured as follows (Figure A.1): the root directory of the project, `project_root`, contains the following subdirectories and files:

- `requirements.txt` is used to install packages into a Python environment. To install the required packages, you can call

  ```
  pip install -r requirements.txt
  ```

- `src`: Contains all source Python scripts for model training and evaluation. Important scripts to note are:

  - main_action_classification.py,
  - main_action_generation.py,
  - main_state_update.py,
  - evaluate_pipeline.py.

  These scripts accept several command-line arguments, as described below.

- data: Contains the following:
  - cache: A cache for the dataset. If this does not exist, it will be created.
  - database: Contains JSON files with database entries for each domain.
  - actions_support.csv: A CSV file with the values in Appendix A.1.
- models: Stores trained models.
- results: Saves metrics and results of evaluations.

  If you're interested in the output of our models, we recommend examining the contents of the results folder. This directory contains subdirectories named test<r>, where <r> is the ratio of the training dataset that the model was trained on (30, 50, or 100). Each test<r> folder contains two subfolders:

  - generated_state: Contains results for action selection models using the generated dialogue state as input.
  - ground_true_state: Contains results for action selection models using the ground-truth dialogue state as input.

    Each of these subfolders includes metrics and results:

    * test_action_cla_metrics.csv,
    * test_action_gen_metrics.csv,
    * test_state_metrics.csv
    * test_results.json,
    * test_results.csv,
    * test_results_subset.json
    * test_results_subset.csv.

A.3.1 evaluate_pipeline.py

This script is used for evaluating the models. An example of running this script in a terminal is:

```
python evaluate_pipeline.py
--state_model_name_or_path "[DST_model]"
--action_cla_model_name_or_path "[AS_classification_model]"
--action_gen_model_name_or_path "[AS_generation_model]"
--use_predicted_states False
```

```
--save_path "../results/test_all/ground_true_state"
--dataset_name 'test'
--random_seed 42
--data_path "../data"
```

A.3.2 main_action_classification.py

This script is used for training the action classification model. The command to run this script in a terminal is:

```
python main_action_classification.py
--model_name_or_path 'roberta-base'
--model_root_path "../models/action_classification"
--local_model False
--tokenizer_name 'roberta-base'
--train_subset_size 1.0
--batch_size 32
--max_seq_length 509
--epochs 30
--learning_rate 2e-5
--early_stopping_patience 15
--data_path "../data"
```

A.3.3 main_action_generation.py

This script is used for training the action generation model. Example command:

```
python main_action_generation.py
--model_name_or_path 'google/flan-t5-base'
--model_root_path "../models/action_generation"
--local_model False
--tokenizer_name 'google/flan-t5-base'
--train_subset_size 0.5
--batch_size 16
--max_source_length 260
--max_target_length 230
--epochs 50
--learning_rate 1e-4
--early_stopping_patience 20
--data_path "../data"
```

A.3.4 main_state_update.py

This script is used for training the DST model. Example command:

```
python main_state_update.py
--model_name_or_path 'google/flan-t5-base'
--model_root_path "../models/state"
--local_model False
```

```
--tokenizer_name 'google/flan-t5-base'
--train_subset_size 0.5
--batch_size 16
--max_source_length 260
--max_target_length 230
--epochs 50
--learning_rate 1e-4
--early_stopping_patience 15
--data_path "../data"
```

Please refer to the individual scripts for more detailed information about each argument.

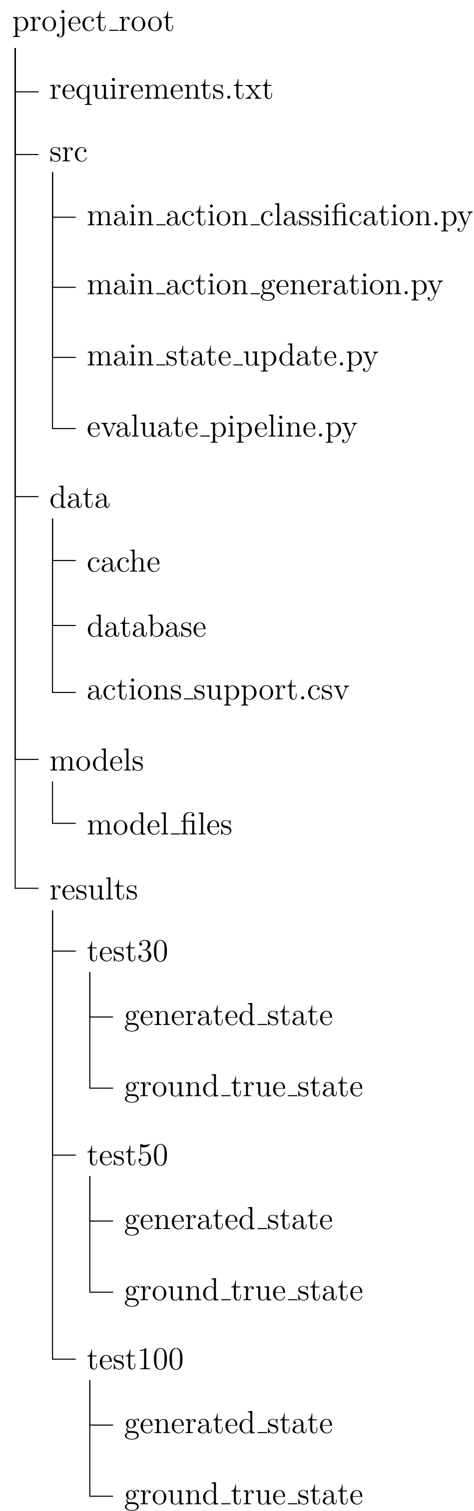

Figure A.1: Directory structure of the project.