Karlsruhe Institute of Technology

A Comparison of Zero/Few-shot Approaches to Natural Language Understanding

Master's Thesis of

Danqing Liu

KIT Department of Informatics
Institut for Anthropomatics and Robotics (IAR)
Interactive Systems Lab (ISL)

Reviewers:	Prof. Dr. Alex Waibel
	Prof. Dr. Tamim Asfour
Advisors:	M.Sc. Leonard Bärmann
	M.Sc. Stefan Constantin

30. September 2022 – 30 . March 2023

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

PLACE, DATE

. . . . . .

(Danqing Liu)

Acknowledgments

First, I would like to thank Prof. Dr. Alexander Waibel for the opportunity to write my thesis at the Interactive Systems Labs. I would also like to thank my advisors, Leonard Bärmann and Stefan Constantin. Leonard helped me in the early stages of my graduate studies as a fellow student. Later on, as my thesis main advisor, he provided me with invaluable guidance and professional opinions, helping me to overcome various difficulties in my research. Stefan introduced me to the field of Natural Language Processing and provided me with essential support during my thesis work. Furthermore, I also want to express my gratitude to Rainer Kartmann. His mentorship and expertise have shaped my research skills and enabled me to successfully complete my thesis. Finally, I would like to thank Patrick Hegemann for his professional advice, as well as his emotional support, especially his firm belief in my abilities. Without the support of these individuals, this research paper would not have been possible. I am grateful for their contributions.

最后，我想用中文感谢支持我学业的家人和朋友。他们的关心和鼓励使我能够顺利完成我的毕业设计。

Abstract

In a scenario where an artificial agent robot assists humans with everyday tasks, it is desired to create more intuitive and efficient communication between humans and machines. As natural language is the primary means of how people express their needs and desires, agents should be able to interact with humans using natural language.

Therefore, a natural language task-oriented dialog system is required, which helps users accomplish specific tasks through natural language interactions, such as booking a flight, ordering food, or scheduling an appointment.

One of the critical components in such task-oriented dialog systems is natural language understanding, which allows the accurate identification of the user's intent and extraction of important information which helps in completing the task and responding appropriately.

In this work, we propose three different systems for intent classification and slot-filling tasks in such a natural language understanding component. The first approach is based on semantic parsing of the user utterance, rule-based phrase extraction, and finally a classification using Sentence-BERT. The second approach works by fine-tuning a BERT model that is pre-trained on a large dataset. In the third and final approach, we use Sentence-BERT to select a known example that is similar to the user utterance, and then dynamically construct a prompt based on a designed template for response generation using GPT-3. We evaluate and analyze the performance of all three approaches under different experimental settings, with a focus on zero-shot and few-shot settings.

Based on the results of the experiments, we can conclude that the semantic parsing approach as our baseline for zero-shot learning performs better in domains where its intents and slots are similar to the wording of the user utterance. The fine-tuning approach, however, works better when more examples are given, and can adapt fast to a new task when it is pre-trained on a similar task, but not necessarily has better performance than without task-specific pre-training. Furthermore, it needs to be trained each time when solving tasks in a new domain, while the other two approaches only require the definition of the task (intents and slots). GPT-3 has shown its ability to adapt to tasks in different domains with minimum information, but it produces unorganized output in some cases, which increases the difficulty in the post-processing stage. For the three approaches mentioned above, the code for training and evaluation can be found at `https://github.com/DankiLiu/dliu-ds-ma.git`.

Contents

1 Introduction

An artificial agent aims to assist humans with everyday tasks. Under this scenario, it is desired to create a more intuitive and efficient communication between human and machines. Natural language is no doubt the primary means of how people express their needs and desires. By allowing humans to interact with agents using natural language, it not only bridges the gap between the capabilities of machines and the ways in which humans prefer to communicate, but also enables a wider range of people to interact with agents, as it eliminates the need for specialized technical knowledge or training. In addition, natural language interfaces can simplify complex tasks by breaking them down into a series of simple commands or questions, which can be easily understood and executed by the agent. This can help reduce errors and increase efficiency, making it easier for people to accomplish their tasks and achieve their goals.

For this purpose, a natural language task-oriented dialog system is required. A task-oriented dialog system is designed to help users accomplish specific tasks through natural language interactions, such as booking a flight, ordering food, or scheduling an appointment. A traditional pipeline structure dialog system consists of four components, which are natural language understanding (NLU), dialog state tracker (DST), dialog policy (Policy) and natural language generation (NLG). NLU is a critical part of a task-oriented dialog system for agents to solve everyday tasks. NLU enables an agent to understand and interpret the natural language input provided by the user, allowing it to accurately identify the user's intent, as well as the important information that helps the agent to complete the task and respond appropriately.

More specifically, NLU is responsible for two tasks: 1) intent classification and 2) slot-filling. An example is shown in Figure 1.1, when a person says to the artificial agent: "Show me the flights from New York to Atlanta.". The agent has to be able to know that 1) the person's intention is to *show flights*, and 2) the information needed to fulfill user's intention, such as the flight should `depart` from *New York* and `arrive` in *Atlanta*. With

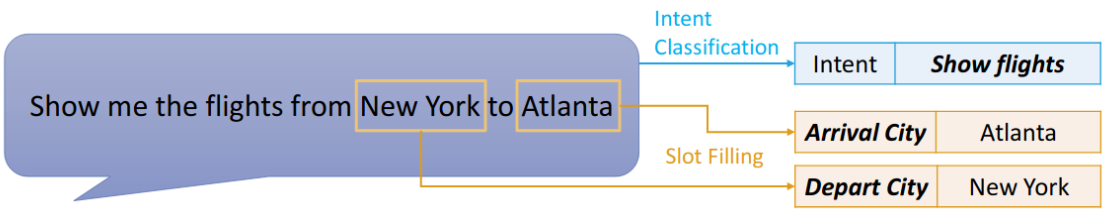

Figure 1.1: An example for intent classification and slot-filling tasks in a NLU component.

the information extracted by NLU, the agent can perform the task in the later components and interact with the user if further information is needed.

Traditionally, these tasks have relied on supervised learning algorithms that require large amounts of labeled data to achieve high accuracy. However, labeling data can be time-consuming and expensive, particularly in certain domains with relatively few resources.

Zero-shot and few-shot approaches aim to address these challenges by leveraging pre-existing knowledge and few labeled examples to achieve reasonable performance on the task. Zero-shot learning involves training a model on one task and testing it on a different but related task without any task-specific training examples. Few-shot learning, on the other hand, involves training a model with very few labeled examples and using it to perform the task.

By using zero-shot and few-shot approaches, the cost and effort required to train a natural language understanding component can be significantly reduced, making it more accessible and practical for a wider range of applications.

By developing an effective NLU component with zero/few-shot learning, a task-oriented dialog system can provide several benefits:

1. Improved accuracy: An effective NLU component can accurately understand the user's intent and extract information from user utterance, reducing errors and improving the overall accuracy of the system's responses and adapt to new domains with limited examples.

2. Greater flexibility: A well-designed NLU component can handle a wide range of inputs, allowing users to express their needs and preferences in a variety of ways.

3. Enhanced user experience: By accurately interpreting user input and providing relevant responses, an NLU component can enhance the overall user experience, making it easier and more intuitive for users to interact with the agent.

NLU aims to classify user utterances to an intent category, to extract important information and to annotate utterances with a semantic type using the prior knowledge of the possible intent categories and semantic types of a task in a specific domain. For consistency in wording, intent categories are referred to as intent labels, and semantic types are referred to as slot labels in the following content.

Mansimov et al. [16] utilize semantic parsing to predict intent and slots for a span of characters through iteration, while other work adopts pre-trained models to adapt to tasks in new domains [10, 36, 23, 3, 29, 4, 37]. With the huge success of pre-trained models, a combination of prompt engineering and a large pre-trained model shows its strength at tackling tasks in a task-oriented dialog systems [14, 29, 17]. Some work also takes a step in exploring zero-shot and few-shot approaches for solving NLU tasks [22, 34, 17, 18, 14, 33, 2].

In this work, we want to research different approaches for solving intent classification and slot-filling tasks in a natural language understanding component of a task-oriented dialog system and their performance with different experimental settings, especially their performance under zero-shot and few-shot settings. Three approaches are introduced: 1)

semantic parsing, 2) fine-tuning a pre-trained BERT model [6], and 3) dynamic prompt construction with GPT-3 [2].

In chapter 2, basic concepts about task-oriented dialog systems and language models are explained. Then the recent related work is introduced in chapter 3. The architecture of the three proposed approaches and their implementation are described in detail in chapter 4. In chapter 6, extensive experiments are designed for each approach from different perspectives, and their results are analyzed with respect to several key aspects. Finally, chapter 7 concludes the thesis by summarizing the evaluation results on three approaches.

2 Basics

This chapter explains the basic concepts and techniques that this work builds on. The reader should already have fundamental knowledge about deep neural networks (DNNs) and how they are trained through back-propagation.

2.1 Task-Oriented Dialog Systems

Task-Oriented Dialog (TOD) systems are designed to assist users to achieve predefined tasks in everyday scenarios such as booking a flight or making a restaurant reservation. To this end, conventional TOD systems typically use a pipeline approach. However, with the limitations of the conventional pipeline, as well as the power demonstrated by pre-trained sequence-to-sequence models, there is a strong trend towards end-to-end dialog systems. Both approaches for developing a TOD system, as well as other important concepts are described in the following.

Pipeline Architecture A traditional way of developing a TOD system is to use a pipeline architecture. A TOD has four components: a natural language understanding (NLU) module, a dialog state tracking (DST) module, a dialog policy (POL), and a natural language generation (NLG) module. NLU interprets the user's **intent** from the given utterance and extracts information that is relevant to accomplish the user's intent. The associated information is represented as task-specific **slots** and their **values**. DST then tracks the values of slots, and POL will decide which action the system will take. Finally, NLG generates the response according to the system action. This work focuses on two sub-tasks of NLU, intent classification, and slot-filling. For a task in a specific domain, intent classification predicts the intent of the user from predefined intent categories, and slot-filling identifies which information can fill into the slots of the pre-defined task-specific slots.

End-to-End Due to the arguments that a model with separated modules accumulates error, and that better performance of individual modules does not necessarily lead to a better performance of the whole model, recent work [10, 36, 17, 23, 28, 29] utilizes pre-trained models to implement end-to-end TOD systems. In an end-to-end TOD system, given the user utterance as input, the model outputs the response directly without having the information in the intermediate steps such as dialog state explicitly.

Domain In the context of TOD systems, a domain refers to a specific area or topic that the dialog system is designed to handle. For example, airline reservation is the domain for a TOD system that helps the user to search for and book flights. A domain contains a collection of intents and slots. Each slot can take one possible value, and each task given

by the user in an utterance contains only one intent. However, the number of slots is undetermined and depends on how much information is provided in the utterance.

2.2 The Encoder-Decoder Architecture

In the late 1980s, time delay neural network (TDNN) [32] is purposed to model sequential data and applied to a task of phoneme classification for automatic speech recognition. Nowadays, Encoder-Decoder [30] models are designed to solve Sequence-to-Sequence problems, which is a special class of Sequence Modelling Problems. This section first introduces the definition of Sequence Modelling Problems, then explains how an encoder-decoder model is built.

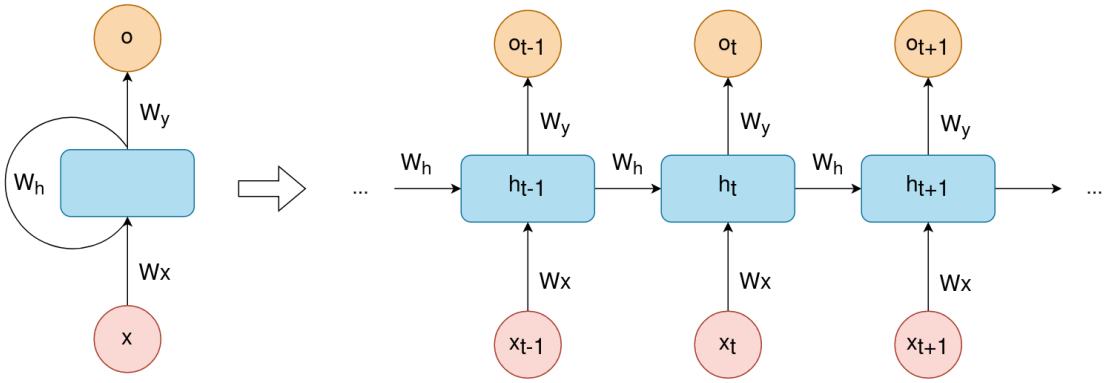

Figure 2.1: An unrolled recurrent neural network.

Sequence Modelling Problems Sequence Modelling Problems refer to problems/tasks where either the input and/or the output is a sequence of data. Consider the model for solving Intent Classification (IC) task in NLU, where given a user utterance as input, the model predicts a user intent. The user utterance is a sequence of words, and the classes of intent are pre-defined. Essentially, the model predicts an intent class from a sequence of words. The problem of predicting a single class from a sequence of data is called a *Sequence Classification* problem.

However, for a Slot-Filling task in NLU, the model takes a sequence of words as input and predicts a slot label for each word. The output of a slot-filling task is a sequence of slot labels (if a word does not belong to any of the defined slot labels, the output is a class representing "no slot label"). The problem where the input and the output are both sequences is referred to as a *Sequence-to-Sequence (Seq2Seq)* problem. More specifically, because the input and output sequences for Slot-Filling tasks have the same length, it is also called a *Token Classification* problem.

RNN Traditional *Deep Neural Networks (DNN)* require inputs and targets of a task that can be encoded into a vector of fixed length instead of a variable length. In the context

of natural language processing where the input is a sequence of words, if inputs are encoded into vectors of fixed length, important information such as positions of the words is missing. To solve this issue, *Recurrent Neural Networks (RNNs)*[27] are proposed to model sequential data.

As shown in figure 2.1, RNNs have a loop that can pass the history information from one step to the next, so that the information exists in the entire network. Taking an input x_{t-1} at position $t-1$, a neural network outputs a value o_{t-1}, and the hidden state h_{t-1} is passed to the next step as prior information. In the next step, the network hidden state h_t is the weighted sum of h_{t-1} and x_t by W_h, and the network makes the prediction based on both input x_t and h_{t-1}. 2.1 is the equation for calculating h_t.

$$h_t = f(W_h h_{t-1} + W_x x_t) \tag{2.1}$$

However, as the network proceeds through the sequence, the further the past information is from the current location, the smaller is its weight in the current hidden state. Therefore, the prediction in the current step depends more on the recent information and thus the network can not learn a long-range dependency. This is called the *vanishing gradient* problem.

LSTM and GRU To enable recurrent neural networks to learn from long-term memories, *Long Short-Term Memory (LSTM)* is proposed [27]. Each LSTM unit contains four parts: cell state, forget gate, input gate and output gate. The cell state passes the information throughout the network so that the information from early time steps can be carried all the way to the last step. The different gates in LSTMs are used to regulate the flow of information and learn which information is important to preserve or get rid of.

The approach of Gated Recurrent Units (GRU) is similar to that of LSTMs. Instead of using a cell state to transfer information, a GRU uses the hidden state. GRU has only two gates, a reset gate and an update gate.

Encoder and Decoder The encoder-decoder architecture aims to solve Seq2Seq problems. An encoder is a stack of several recurrent units (RNN, LSTM or GRU cells) that accept the inputs sequentially and encode them into a hidden vector. This hidden vector is then the input of a decoder. A decoder is a stack of recurrent units where each unit predicts an output at each time step.

2.3 Transformer

The transformer architecture is the basis for well-known models like BERT [6] and GPT-3 [14] which are also used in this work. Same as recurrent neural networks, transformers are also designed to solve Seq2Seq problems. However, instead of processing all the inputs sequentially, transformers compute the representations of the inputs all at once with the mechanism called self-attention.

Encoder-Decoder Architecture The transformer adopts an encoder-decoder architecture that is explained above, the encoder extracts features from input sequence, and the decoder

uses the features to produce an output sequence. The transformer's encoder is a stack of encoder blocks, and the output of the last encoder block is the input of the decoder. The decoder also consists of multiple decoder blocks.

Input Embedding and Positional Encoding The inputs are first tokenized into tokens. The token positions are important information for sequential data, with positional encoding, the model has the information of token positions.

Softmax and Output Probabilities The decoder outputs one token at a time, and this output token will become a part of the input for the next step. The output will go through a linear transformation and then a softmax layer so that its dimension is changed from the embedding vector size into the size of a number of output classes and then converted into probabilities. There are two ways to predict the class from the output probabilities. The first method is to choose the class with the highest probability, and it is called *greedy method*. The second method, however, looks for the best combination of the tokens rather than selecting the most probable class at a time, and is called *beam search*.

2.3.1 BERT

BERT (Bidirectional Encoder Representations from Transformers, [6]) is based on transformer architecture that was pre-trained simultaneously on two tasks: language modelling and next sentence prediction. With these two training objectives, BERT learns latent representations of words and sentences in context. After pre-training, it can be fine-tuned with fewer resources on smaller datasets to adapt to specific tasks such as intent classification and slot-filling tasks.

Masked LM In language modelling task, 15% of the words in each input sequence are replaced with a [MASK] token. The model aims to predict the original value of the masked words based on the context provided by the other, non-masked works in the sequence. In order to make predictions, a classification layer is added on top of the BERT which takes the hidden representation of the sequence from BERT as input and outputs a distribution over all vocabulary.

Next sentence prediction In the next sentence prediction task, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. In order to distinguish both sentences, a [CLS] is inserted at the beginning of the first sentence and [SEP] is inserted at the end of each sentence, and a sentence embedding is added to indicate which part is the first sentence and which is the second. Finally, a positional embedding is added to mark the positional information of each token. When training the BERT model, Masked LM and Next Sentence Prediction are trained together, with the goal of minimizing the combined loss function of the two strategies.

Fine-tuning With the pre-trained BERT, we can use it for a specific task by fine-tuning the model on the new dataset. In our work, it is defined that each user utterance has one user intent and multiple slots. Due to this reason, we can consider intent classification as

a sequence classification task, and slot-filling task as a token classification task. We can make use of a pre-trained BERT model by adding a classification layer on top.

There are two ways of fine-tuning a pre-trained BERT model to a new task, one way is to update parameters in both the encoder (pre-trained BERT) and the classifier, another way is to freeze the parameters in the encoder and only update the parameters in the classifier.

2.3.2 Sentence-BERT

The construction of BERT makes it unsuitable for semantic similarity search as well as for unsupervised tasks like clustering [26]. Sentence-BERT is a modification of the pre-trained BERT network that use siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity.

3 Related Work

Motivated by the need to build task-oriented dialog systems that can quickly adapt to a new task with minimal training data, this work focuses on comparing different approaches for building a natural language understanding component that solves intent classification and slot-filling tasks under a few-shot setting, and furthermore discusses their abilities to adapt to a new domain with limited examples.

Architecturally, the implementation of a dialog system can generally be divided into two categories, implementing each component of a TOD separately so that each component is trained to be excellent at one specific task, or building an end-to-end TOD so that it reaches the overall best performance without caring much about the transaction details between components. However, building a TOD often requires a large amount of annotated data for each specific domain which is hard to obtain. From these perspectives, in this chapter, the related work will be introduced mainly in three aspects: different approaches that adopt a pipeline structure for building a dialog system are described in section 3.1, work that trains an end-to-end TOD in section 3.2, and work that focuses on discussing how to apply few-shot or zero-shot approaches for realizing a TOD in section 3.3.

3.1 Pipeline Structured Task-Oriented Dialog Systems

In this section, the works that apply a pipeline structure to solve tasks in TOD are introduced. Most of the work aims at solving intent classification and slot-filling tasks, while other works focus on other tasks in a TOD, yet its approaches are also valuable to have a closer look at. Semantic parsing aims to analyze natural language sentences, extract their meaning, and process them into a formal, structured representation. [16] introduces an encoder network that incrementally builds a semantic parse tree for each user utterance. As shown in Figure 3.1, at each iteration, it predicts either an intent label or a slot label for a span of characters in the utterance, as long as the prediction is not terminated, the predicted label is inserted to the input at the predicted position to be the input of the next iteration. The model continues the prediction iteration until the end position is at the termination position. Same as the task defined in this thesis, a user utterance is used as input rather than dealing with multi-turns dialogues. The semantic parsing approach in this thesis constructs a similar semantic parse tree as this work, however, each label annotates a sequence of words in the user utterance instead of annotating characters. Moreover, it utilizes existing parsing models to build a baseline model for zero-shot semantic parsing. In other words, no training is involved in the semantic parsing approach. [7] presents three different improvements to a semantic parsing model, contextualized embeddings, ensembling, and pair-wise re-ranking based on a language model.

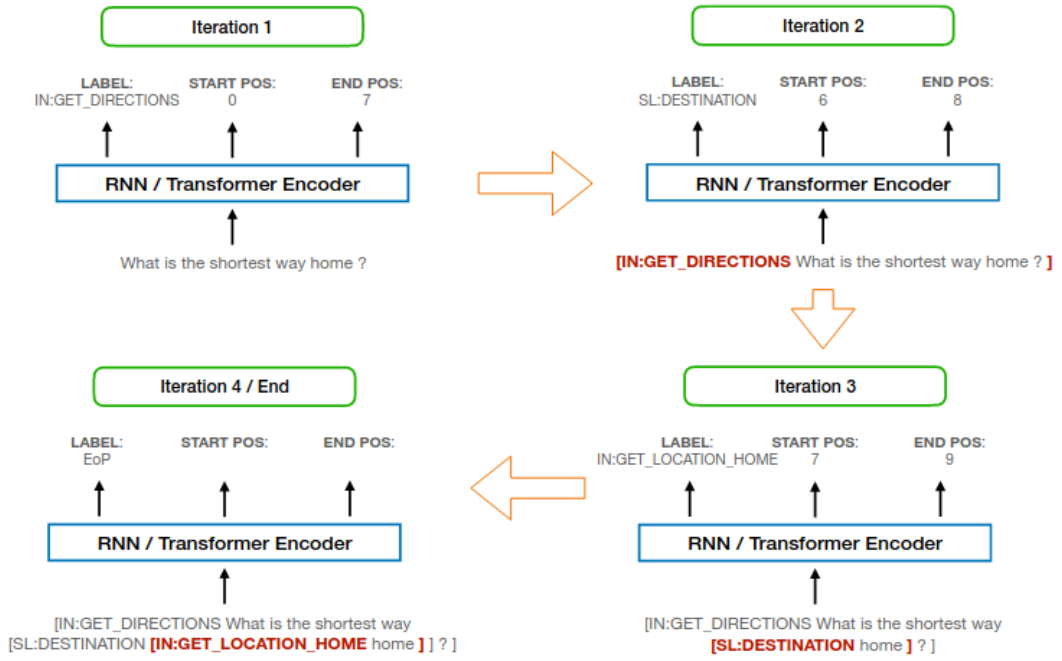

Figure 3.1: Overview of generation of a semantic parse tree in [16].

Sequence labeling can be adopted for natural language tasks such as slot-filling. [33] and [34] adopt self-training to achieve a better performance on sequence labeling tasks with low-resource labeled data.

[4] extends BERT to a joint intent classification and slot-filling model. The input tokens are embeddings of the input text that starts with special token [CLS] and ends with [SEP]. BERT then produces a hidden state vector from the input tokens. For intent classification tasks, an intent label is predicted based on the hidden state of the special token [CLS], denoted h_1. Equation 3.1 shows how the classifier predicts the intent label. W^i and b^i are the weight matrix and bias for the intent classification task.

$$y^i = softmax(W^i h_1 + b^i) \tag{3.1}$$

And for the slot-filling task, a slot label is predicted for each hidden state of other tokens. The hidden state h_n corresponds to the n_{th} token, and W^s and b^s are the weight matrix and bias for the slot-filling task.

$$y_n^s = softmax(W^s h_n + b^s), n \in 1...N \tag{3.2}$$

The second approach in this work also utilizes a pre-trained BERT to train our own multi-task model for solving NLU tasks. Although the two works have the similar idea of using BERT as a pre-trained model and adapt it to solve intent classification and slot-filling tasks, the architecture of the models are not the same. Instead of predicting an intent label from the first special token, we add another classification layer and predict the user's intent with the hidden representation from the whole user utterance.

[12] tackles the intent classification task by considering the features of the dialog text and Chinese characters. Instead of using word-level features used by most work, it applies CNN to extract character-level local features of the Chinese characters, and a bidirectional gated recurrent unit layer architecture to capture the contextual semantic information to predict the intent of the user.

[38] tackles cross-domain slot-value prediction tasks by applying a slot gate to learn to predict whether a value should be predicted for a domain-slot. The value is either a span in the utterance, or a value from a candidate list for that slot. This work focuses on training a dialog system on a single domain and aims to exam the ability for a single-domain dialog system to transfer to another domain with few examples.

[22] develops a multi-layer Transformer neural network model SC-GPT to convert a dialog act representation in a semantic form into a response in natural language. SC-GPT is also trained in a first pre-train, then fine-tune manner.

As introduced in this section, to solve NLU tasks in a TOD separately from other components, a popular way is making use of semantic parsing approaches that analyze the syntactic and structure of a sentence. Another idea is to apply transfer learning. A pre-trained model that is already trained on a large unlabeled corpus such as BERT is a powerful encoder and can be fine-tuned with a relatively small amount of data for a target task, yet still has a promising performance.

3.2 End-to-End Task-Oriented Dialog Systems

In this section, the work that builds an end-to-end TOD is introduced. Despite training in an end-to-end manner, some work can be used for down-stream tasks in NLP such as intent classification and slot-filling. Most of the work applies transfer learning and uses models such as BERT and T5 as a backbone. Furthermore, a concept in machine learning called prompt engineering is explained in this section.

Transfer Learning Inspired by the recent success of applying transfer learning to Natural Language Processing (NLP) tasks, [23] presents SOLOIST for building task bots at scale. SOLOIST deals with multi-turns dialogues and each dialog turn is a concatenation of dialog history, dialog belief, database state, and delexicalized dialog response. [23] uses GPT-2 as the model backbone and trains a pre-trained task-grounded response generation model on large dialog corpora. Then the pre-trained model is fine-tuned and adapted to a new task using a machine teaching tool. The authors did a component-wise evaluation on NLP tasks. The intent classification is defined as classifying a user utterance into one of several pre-defined classes and the last hidden state by SOLOIST is used as the representation for sequence classification. Compared to our transfer learning approach, the difference is that they use SOLOIST as a encoder to produce hidden representation for intent classification while we use a fine-tuned BERT. A slot-filling task in SOLOIST is defined as a turn-based span extraction problem where we define a token classification problem and only observe single-turn utterance. [37] also makes use of GPT-2 and fine-tunes it to build an end-to-end TOD.

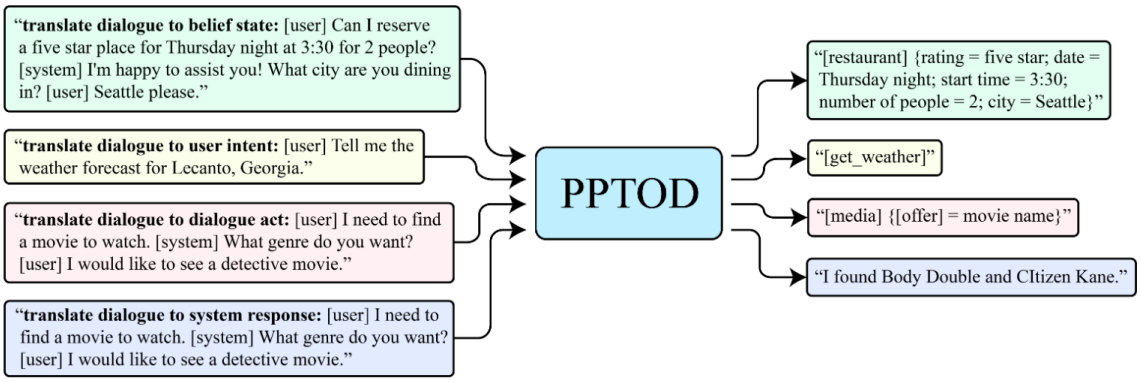

Figure 3.2: An overview for PPTOD. For each task, the model tasks the dialog context and the task-specific prompt as input and learns to generate the corresponding target text [29].

Prompt Engineering in Transfer Learning Based on pre-training, researchers have proposed prompting methods to increase the model performance. A prompting methods survey [15] mentions that prompt engineering can be used in text classification as well as information extraction tasks. A natural way to create prompts is to manually create intuitive templates based on human introspection. [29] introduces a unified plug-and-play model for multi-task TOD named PPTOD. The authors integrate components in a TOD into a unified model and enable the model to handle multi-tasks by plugging a task-specific prompt into the dialog context as the model input. An overview is shown in figure 3.2. PPTOD is first initialized with T5 [25] and pre-trained on a heterogeneous set of dialog corpora that consist of partially-annotated data, and then fine-tuned to a new task with task-specific labelled data. Different from our approach, PPTOD solves the intent classification problem as a generation problem instead of a classification problem, it directly generates the text of intent label. Another end-to-end model CINS purposes a Comprehensive Instruction to different TOD downstream tasks. To solve intent classification, dialog state tracking, as well as natural language generation tasks, [17] also employs T5 model to generate the output text for each task. The model input for each task consists of a input text as well as a Comprehensive Instruction that consists of a *task definition* that defines the task, a *task constraint* that defines the output space of the task, and a *prompt*. Figure 3.2 shows an example of the Comprehensive Instruction of a intent classification task.

3.3 Zero-Shot and Few-Shot Learning

Using data-driven approaches for TODs often requires fine-grained annotations to learn the dialog model in a specific domain, which is one of the biggest challenges for task-oriented dialog systems [40]. Acquiring a large amount of annotated dialog data can be very expensive, especially due to the diversity and complexity of dialog tasks in different domains. Thus, exploring few-shot methods for building dialog systems is a promising research direction that can yield significant benefits in practical applications. Transfer

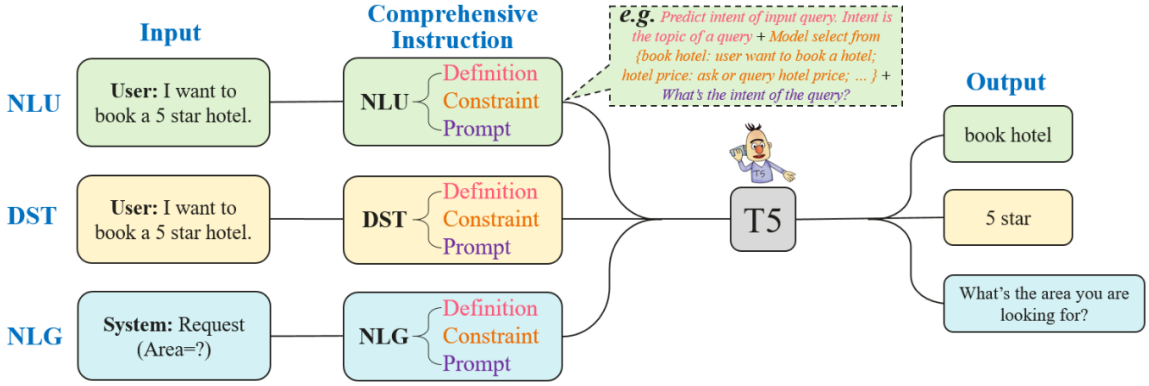

Figure 3.3: The unified framework of applying Comprehensive Instruction to different TOD downstream tasks [17].

learning takes few-shot in TODs a big step forward, as it enables large language models to quickly adapt to a new domain with limited annotated data.

Some the above mentioned transfer learning based approaches performed evaluations in a low-resource setting [23, 17, 29, 22]. [23] performs a task-grounded pre-training with a joint objective, while [29] trains the pre-trained model with a maximum likelihood objective, and [17] directly makes use of the pre-trained T5 without additional training. The few-shot testing on all above mentioned models draw a consistent conclusion: a larger model is a better few-shot learner. Furthermore, in slot-filling tasks, the transfer ability of a model also depends on domain types. The model performs better on a group of domains that are similar to each other, yet performs bad on other few. The authors assume this is because some domain has domain-specific slots that are rare in other domains [17].

Another way to tackle with the the data-shortage problem is to apply *self-training*, so that it can make use of the large amount of unlabeled, or partially labeled data to train a high-performing model. The idea is to first train a *Teacher* on the labeled examples for generating pseudo-labels for unlabeled data, and then select data from all labeled data to train a *Student*. The whole process is iterative, in the next iteration, the *Student* becomes the *Teacher* [18, 34, 33].

Applying prompt engineering to large language models has been a great success, researchers continue to explore their possibilities for zero-shot learning. [14] shows that by simply adding "Let us think step by step." as a single prompt template can greatly improve the zero-shot performance on a pre-trained large language model. To explore more on the few-shot/zero-shot ability on a large language model, in this work, different prompt templates are designed to make use of GPT-3 [2] for solving intent classification task and slot-filling task in a TOD system.

4 Task Definition and Three Approaches

The purpose of this work is to implement different approaches for solving intent classification and slot-filling tasks in a natural language understanding component of a task-oriented dialog system, as well as explore their ability to transfer to new domains and evaluate their performance with limited data. Three approaches are proposed, namely 1) semantic parsing approach, 2) fine-tuning a pre-trained BERT model, and 3) dynamic prompt construction with GPT-3.

In this chapter, the intent classification and slot-filling tasks are defined in detail, and the architecture of each of the three approaches is described.

4.1 Task Definition

Intent classification and *slot-filling* are two important tasks in the natural language understanding component of a dialog system. We consider a single command from the user as the task input, i.e., multi-turns dialog is not in the scope of this thesis. Given a user utterance, the goal of the intent classification task is to assign a semantic label that represents the `intent` of the user. In the meantime, the goal of the slot-filling task is to label tokens in the user utterance with semantic types referred to as `slots`, the labeled tokens are key information for performing the task.

For each different domain, a set of semantic labels for intents and slots are defined. For example, when the goal is to help customers to book flights, the user's intent can be to look for flight information, book a flight, or provide customer service, and the slot labels can be the time of flight, price information, and so on. However, in the scenario where the goal is to manage the alarm for the user, the user's intent can be to set an alarm or remove an alarm, and the slots for information can be the time of the alarm and the date of the alarm.

In our definition, each user utterance has only one user intent, whereas an utterance can contain information that matches one or more slots. For example, given the user utterance "Book me a flight from New York to Boston". The intent of the user is `book flight`, and the slot categories for "New York" and "Boston" are `depart city` and `arrival city` respectively. The labels such as `book flight`, `depart city` as well as `arrival city` are pre-defined for the task-domain `flight`.

4.2 Semantic Parsing

Despite the irregularity of user expression, the user's intention can be inferred from the grammatical structure of the utterance. With this in mind, the following parsing methods

are considered: part-of-speech tagging (POS-tagging), dependency parsing (DP) as well as named-entity recognition (NER).

Part-of-speech tagging In grammar, a part of speech is a category of words (or, more generally, of lexical items) that have similar grammatical properties [24]. Words that are assigned to the same part of speech generally display similar syntactic behavior. For example, in the sentence “I like to read books”, the word "like" is tagged as `VBP`, which stands for "Verb, non-3rd person singular present", indicating that the word "like" is recognized as a "verb".

Dependency parsing Dependency parsing is the task of extracting a dependency parse of a sentence that represents its grammatical structure and defines the relationships between words [21]. When a word is dependent on another word, a relation between them exists and a dependency type is assigned to this relation indicating which type of dependency they have. For example, in the sentence “The cat eats tasty fish”, "The" is dependent on "cat" with the dependency type "dt", which means the word "The" is a `determiner` of the word "cat".

Named-entity recognition Named-entity recognition (NER) is a sub-task of information extraction that can locate and classify tokens into pre-defined categories such as person names, organizations, locations, and so on [19]. A NER system can annotate the entity in the text by the names of entities. For example, in the text “Jim bought 300 shares of Acme Corp. in 2006.”, "Jim" and "Acme Corp." will be recognized as a `Person` and a `Organization` respectively [20].

With the above-mentioned semantic parsing methods, an approach for solving intent classification and slot-filling tasks can be designed. The important information in text often appears in phrases rather than individual words. Combining part-of-speech tagging and dependency parsing, phrases carrying important information can be extracted. However, in order to extract the important information and clear out the unrelated information as much as possible, rules are needed to filter the tokens in the utterance. The intuition of rule designing is that some dependencies and POS tags may indicate that the target token carries more important information than others.

4.2.1 Architecture

The semantic parsing approach consists of three steps. In the first step, the user utterance is parsed using POS, NER, and DP. The parsing outputs are then processed in the next step using the heuristic rules and phrases that represent the user intent, and pieces of information for the slots are extracted. In the third step, an intent label is predicted from the pre-defined intent labels according to their similarity to the intent phrases, the slot labels are also assigned in the same way, only that for each slot phrase, one slot label is assigned.

Figure 4.1 shows an overview of the semantic parsing approach. Each block in the figure colored in light green represents a step in the semantic parsing approach.

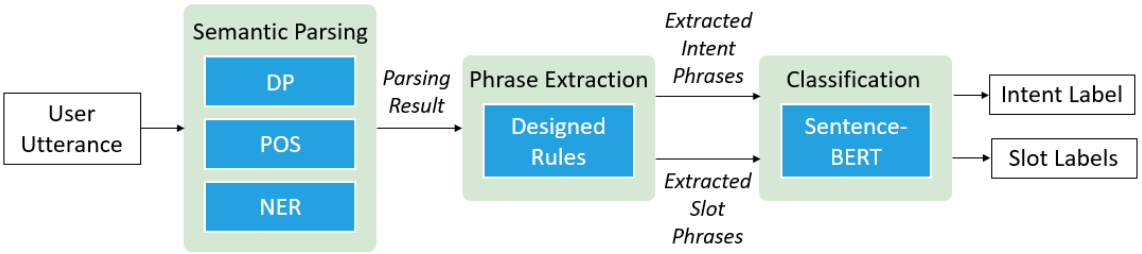

Figure 4.1: An overview of semantic parsing approach.

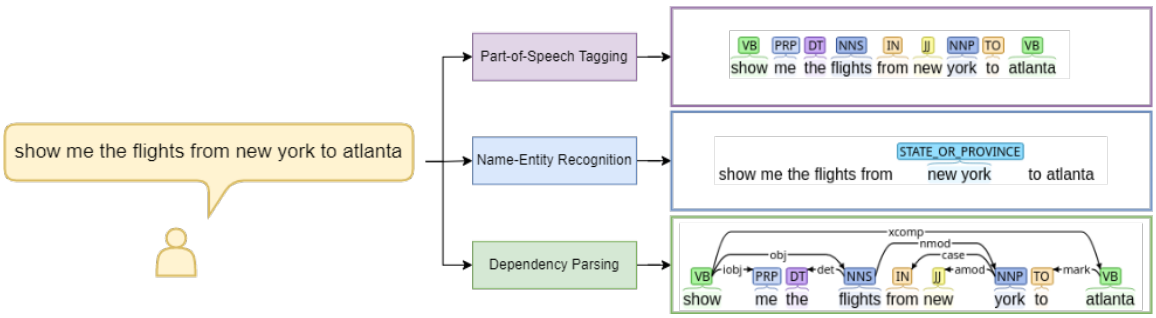

Figure 4.2: A user utterance is parsed using three semantic parsers (POS, NER and DP).

Semantic parsing The input of this first step is a user utterance in natural language text. The input utterance is processed by three different semantic parsing methods, namely part-of-speech tagging (POS), dependency parsing (DP), and named-entity recognition (NER). An example is shown in Figure 4.2. Part-of-speech tagging categorizes each word in the utterance with a particular part-of-speech token. Named-entity recognition labels words or phrases that are recognized as an entity. In our example, even though "new york" and "atlanta" are both city names in the USA in lowercase, only "new york" is recognized as *STATE_OR_PROVINCE*. We use the part-of-speech tagging, named-entity recognition, and dependency parsing provided by the Natural Language Toolkit (NLTK) to parse the user utterance and output the parsing result for the next step.

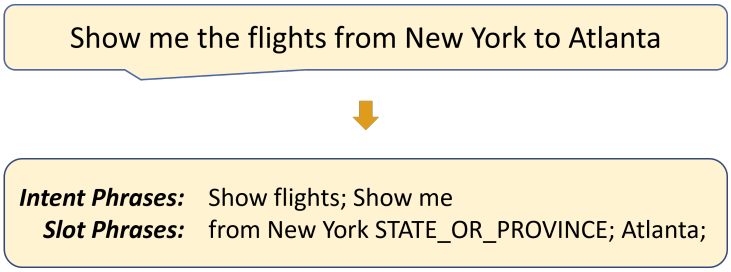

Figure 4.3: A user utterance and intent and slot phrases extracted from it.

Phrase extraction With the result from the last step, the goal of this step is to extract the phrases that can represent user intent and other important information for completing the task. As the example shown in the last step, part-of-speech tags and dependency graph of a sentence contain information such as the role of a word in the context that can be used to predict the user's intention and other slot information. Named-entity recognition can provide extra information about the words in the utterance. Hence, some simple rules are designed to extract the phrases from the utterance that can represent the user's intention and other important information.

Each parsed result can be mapped to each token of the sentence. Tokens are groups of characters when the sentence is split by spaces. Each token has a part-of-speech tag and is a node in the dependency graph. If this token has a dependency with another token, the index of this other token and the type of dependency will be stored in this node. A token can have zero to more than one dependency.

The basic idea of the rule design is that we consider some part-of-speech and types of dependency to carry more information than others. For example, *VB* is the part-of-speech tag of a verb, its object combined with itself usually can indicate the intention of the user. On the contrary, *Dt* is the tag of a determiner, which can usually be omitted from the sentence without changing the meaning of the sentence.

The algorithm for extracting the intent and slot phrases is described in the following: We define the following annotations: T is a list of tokens in the given utterance, and each element t_i at index i is a string. POS are part-of-speech tags of tokens in T, each element pos_i matches a token t_i. DPs is a list of dependency nodes that match one-by-one to tokens in T. Each dependency node has a dictionary that stores how other tokens are dependent

on this token in key-value pairs. The key is the type of dependency and the value is the index of the other token. *NER* is a list of name-entity types of the tokens. For each token t_i, if it is a recognized name-entity, the name-entity type will be assigned to the element at that index. Because elements in T, POS, DPs, and NER all match one-to-one to the tokens in utterance, so they have the same length.

We iterate through elements in lists by index, if a dependency dictionary exists, we iterate through its key-value pairs. If the key (dependency type) is not in our defined dependency types we consider this dependency as not important and skip it, otherwise, we construct a phrase. How to construct the phrase is described in the second procedure. Given tokens list T and dependency key-value pair of the current token, we get the indices of the tokens that dependent on current token from value, sort them in the increasing order and then concatenate the token at those indices with a space in between.

After constructing the phrase, we determine whether this phrase is an intent phrase or a normal token phrase by examining the part-of-speech tag of the current token. If pos_i is VB or VBP, then it is recognized as an intent. The phrase will also be categorized as an intent phrase and added into PI, otherwise, it is a slot phrase and will be added to PT.

Classification The intent label and slot labels are predicted in this step given intent phrases and slot phrases from the last step. Theoretically, there can be an arbitrary number of intent phrases and we need to predict one intent label from them. The process of how an intent label is classified is shown in Figure 4.4, assuming there are two intent phrases and four intent labels. The intent phrases and labels are both embedded with Sentence-BERT [26], resulting in six embedding vectors. The cosine similarity between each intent phrase and intent label is calculated. The similarity scores are shown in the table. To this end, the intent label with the largest similarity score is selected and classified as the intent of the utterance. In our example, the second intent phrase and the third intent label have the largest similarity score, and the third intent label is our predicted user's intent.

Slot label assignments are similar to intent labels, only that we assign one label to each slot phrase instead of only one for all phrases. For each slot phrase, the cosine similarity between Sentence-BERT embedding of itself and of all slot labels are calculated and compared, and the label with the largest cosine score to the slot phrase is assigned to the phrase. To this end, an intent label, as well as a set of slots are predicted for the given user utterance.

4.2.2 Post-processing

The output from the semantic parsing approach consists of two parts, an intent label for the utterance, and a correspondence between slot labels and phrases. Because the phrases are extracted using heuristic algorithms, it does not exist in other approaches, and thus can not be used directly to compare with the outputs from other approaches as well as the ground truth. In order to make a fair comparison to the outputs from other approaches, post-processing is necessary. The solution is to process the intent label, as well as the slot-phrase correspondence into a list of key-value pairs and format them in organized text.

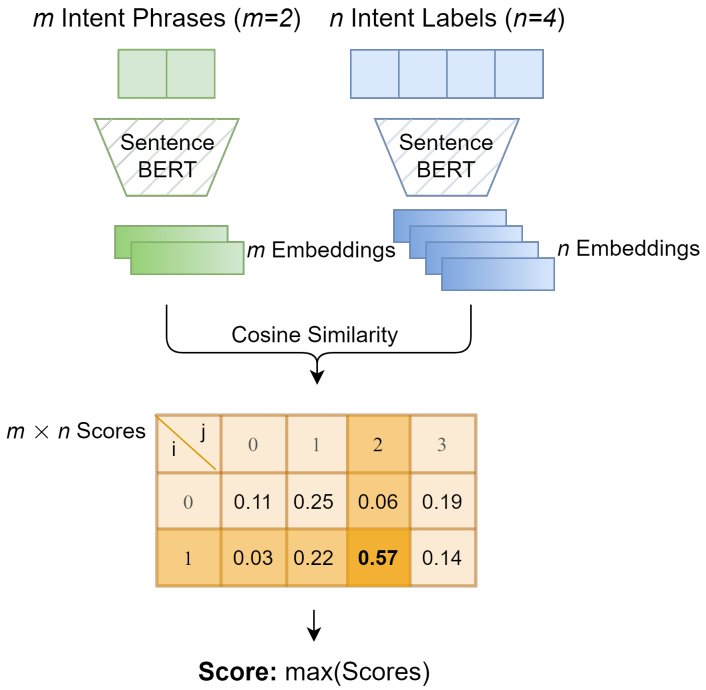

Figure 4.4: Intent classification with Sentence-BERT.

4.3 Fine-tuning a Pre-trained BERT Model

A pre-trained model is a saved network that was previously trained on a large dataset. The model can be used to solve similar problems by training it on a new dataset. The process of training the pre-trained model on the new dataset is called fine-tuning. Given this idea, we can make use of a pre-trained model that does similar tasks as the slot-filling task and fine-tune it on the new dataset. Pre-trained BERT [6] is designed to solve token classification tasks, as well as sequence classification tasks by fine-tuning all layers or one additional output layer only for a specific task. Because each utterance has only one intent, we can consider the intent classification task as a sequence classification task. As for labeling the information in the utterance for different slots, we can consider it a token classification task. If a token is recognized as key information for a slot, it will be labeled a slot label, otherwise, it will be labeled as "O".

4.3.1 Architecture

Because different tasks have different slots and intent labels, the shape of the output layers of fine-tuned BERT model is also different. That is why the pre-trained BERT model has to be fine-tuned for each domain. To make token-level predictions, as well as sequence-level predictions, a multi-task model needs to be constructed.

Model overview An overview of the model is shown in Figure 4.5. The input is a user utterance that will first be tokenized by a tokenizer. The tokenized input will then be

encoded with the pre-trained BERT model to a hidden representation. The hidden representation will be forwarded into both the token classification head and the sequence classification head. Token classification outputs a token prediction, which contains possibility distributions for all tokens over all token labels. For each token, the label with the largest probability is selected as output. The output of the token classification task is a sequence of token labels. The sequence classification head, however, produces only one probability distribution and outputs an intent label that has the largest probability.

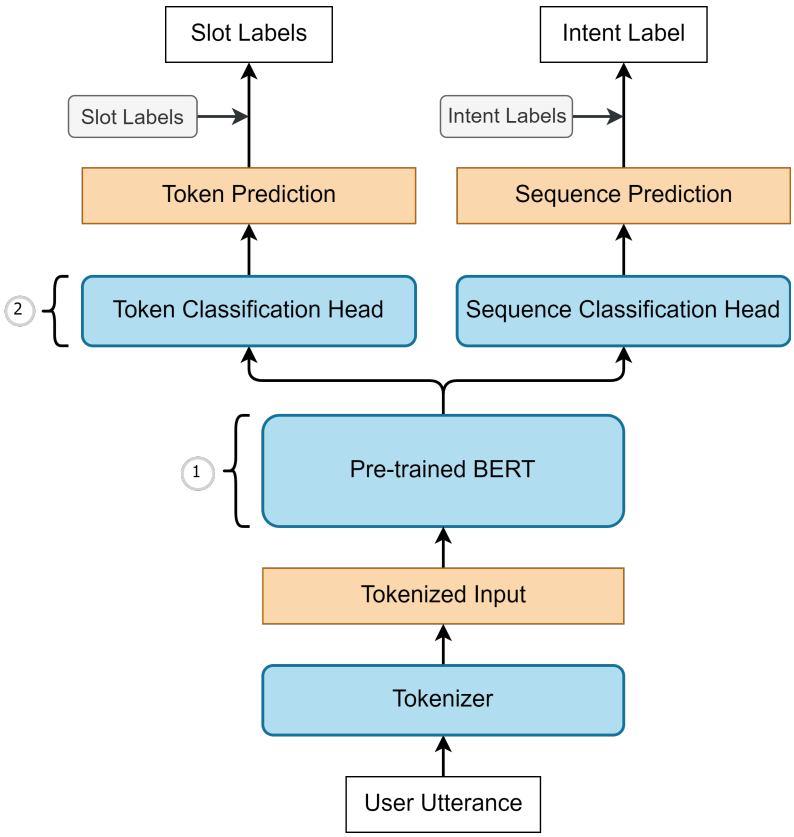

Figure 4.5: Fine-tuned BERT Architecture

Model training For each specific domain, a set of intent and slot labels are defined. A token classification head and a sequence classification head are initialized according to the number of intents and slots respectively. There are two ways for training a model, one way is to train both encoder and classifiers when adapting to a new domain, and the other way is to freeze the parameters in the encoder and only train the classifiers. We adopted both training methods in our experiments for different evaluation purposes.

Decoding The output of the sequence classification head is a distribution over all intent classes, and the output of the token classification head is a sequence of distributions of slot classes with length n, where n is the number of input tokens. For outputs from both heads, the prediction labels are selected greedily. Each distribution contains probabilities of all classes, the class with the highest probability will be selected as the target class. For

intent classification, one intent class with be selected, and for the slot-filling task, a slot label will be selected for each token.

4.3.2 Post-processing

Like in semantic parsing, the output from the fine-tuned model also need to be processed in a unified format. When adjacent tokens have the same slot label, these tokens are considered as a phrase, and this phrase is the value of the slot label. The intent label is the value of the slot "intent". All the slot-value pairs are concatenated together.

4.4 Dynamic Prompt Construction with GPT-3

GPT-3 is an auto-regressive language model that produces text. Given an initial text as prompt, it will produce text that continues the prompt. Because of the above mentioned nature of GPT-3, we expect to get the right answer when we ask the right question (when we have the right prompt). We expect that GPT-3 can solve intent classification and slot-filling tasks when a suitable prompt is given.

4.4.1 Architecture

For the GPT-3 approach, a pipeline is designed. Different templates are designed to construct prompts, if an example is required when constructing a prompt, an example will be selected from the database. A prompt will then be constructed and used as input for GPT-3 to generate the output.

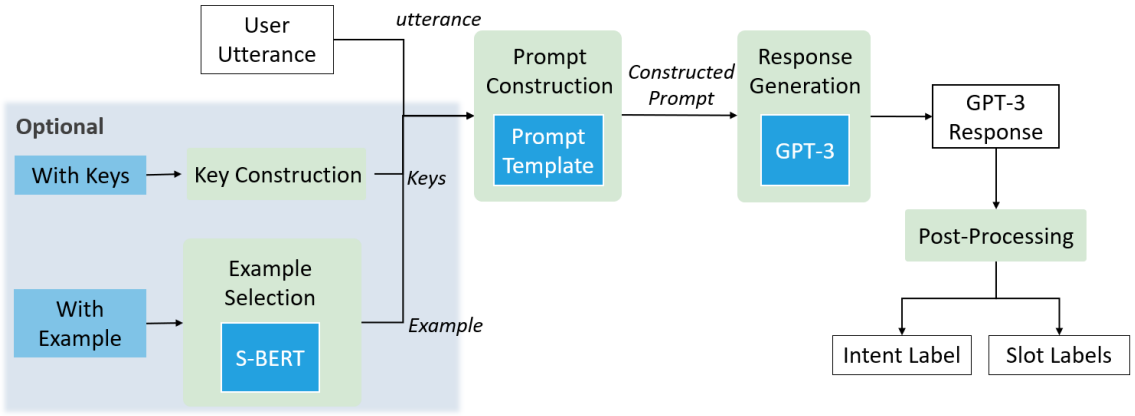

Figure 4.6: Dynamic prompt construction with GPT-3 - an overview.

An overview is shown in Figure 4.6. Given a user utterance, if the task is a one-shot learning task, an example is selected from a set of examples. The example selection will be explained in detail later. The selected example and the user utterance are forwarded to the prompt construction step and replace the placeholder in the prompt template. The

constructed prompt is the input for GPT-3, and finally, GPT-3 generates a response for the task.

Example selection In this work, we designed templates for constructing zero-shot and one-shot prompts for GPT-3. When an example is needed, it is necessary to select a better example from the available examples. Figure 4.7 shows how an example is selected from a set of examples. In this work, we prepared some example tasks in each domain by randomly selecting one example for each intent and slot label. Because an example for one intent can also contain sampled slots, and an example of a slot can also contain a sampled intent, there can be multiple examples for each intent and slot class. The number of examples is the sum of the number of intents and the number of slots in this domain.

Both user utterance and examples are encoded using Sentence-BERT, the cosine similarity of the embedded utterance and samples are calculated and the score is a vector of length n, where n is the number of examples from the database. The example that has the largest similarity score is selected for prompt construction.

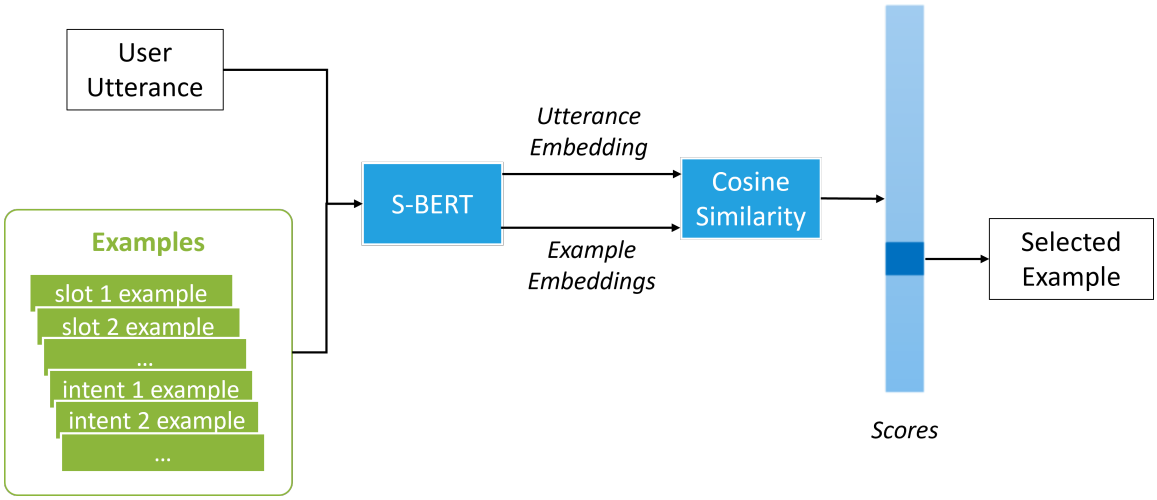

Figure 4.7: Select one example from a set of examples for GPT-3 using Sentence-BERT.

Prompt templates Due to the reason that the example selection and prompt construction happens at inference time, although we can accelerate the process by embedding the examples in advance and store them in the memory, it is not ideal in terms of time and cost to have multiple examples for each task. Thus, we designed different prompt templates for constructing prompts for GPT-3 dynamically. As shown in table 4.1, each template has placeholders in curly brackets. These placeholders can be replaced with the task-specific information accordingly. For example, the placeholder {oneshot_example} can be replaced by a selected example from the target domain and inserted into the prompt. With these templates, we can design a prompt that contains no example (zero-shot) or one example (few-shot), or a prompt that has information about the possible intent and slot categories (`keys`) by choosing to add information at the placeholders. The input user utterance will be placed into the placeholder {utterance}.

Table 4.1: Prompt templates for dynamic prompt constructing of GPT-3.

`Info_Extractor`	Extract the intent and slots for the following user utterance and show them in key-value pairs format. {keys}{oneshot_example} utterance:{utterance} output:
`NLU_Machine`	A machine receives user utterances as commands and detects the user intent and extracts specific pieces of information or entities from the utterance. These pieces of information or entities are referred to as slots. The machine outputs them in key-value pairs format, such as "intent user_intent;slot1 entity1;slot2 entity2;..."{keys} {oneshot_example} Given the following user utterance, what is the machine output? utterance:{utterance} output
`Robot_Translator`	A robot receives user utterances as commands, but it can only assist the user when the user's intent and other key information in the utterance are translated into key-value pairs. In this format, each key describes a value, which is a word or phrase from the original sentence. such as "someKey someValue;..." {keys}{oneshot_example} Given user utterance {utterance}, I want to translate the utterance so that the robot understands it, translation

4.4.2 Post-processing

GPT-3 is a text generation model that generates free text given a prompt. Due to this reason, the responses from GPT-3 are unorganized and may contain special characters such as "\n", "{" and "}". For different prompts, the output of GPT-3 can have quite different formats, depending on what the GPT-3 model understands from the prompt. Thus, it can be very complicated to process the data into the unified format. We need to observe the output of GPT-3 and design the post-processing process for each prompt category. The general idea is to remove special characters in the response and transfer it into key-value pairs. Nevertheless, if we defined the output format specifically or provide an example with the output format we want GPT-3 to generate in the prompt, the output can be formatted into a certain form which is easy to process.

5 Dataset Selection and Data Processing

In this work, semantic parsing, fine-tuning a pre-trained model, and constructing prompts for GPT-3 are purposed to solve intent classification and slot-filling tasks. After implementing the architectures of the three approaches, it is essential to test the approaches not only internally for exploring how each approach adapts to different datasets, but also across approaches to find out how different approaches behave on the same dataset.

To this end, ATIS [11] and MASSIVE [8] are selected for evaluating the approaches 5.1. Moreover, it is also crucial to consider the statistics of the datasets, due to the different sensitivities of different approaches to class imbalance. For a fair comparison, the data for all approaches should be consistent and organized to a format that can be used and analyzed for all approaches. In each section, the dataset detail and its statistics are analyzed, as well as the pre-processing of the dataset.

5.1 The ATIS Dataset

The ATIS (Airline Travel Information Systems) [11] dataset has been popular in the NLP community since its first release, it serves as a standard benchmark dataset widely used for intent classification and slot-filling tasks, which are two sub-tasks of NLU components in a task-oriented dialog system. It consists of audio recordings and corresponding manual transcripts about humans asking for flight information on automated airline travel inquiry systems. However, this work is text-based and audio information is out of scope.

Figure 5.1 shows an example from ATIS. The utterance is a sentence where the user asks the machine for flight information, and each utterance corresponds to one goal/intent and multiple slots, the slots correspond to words in the utterance.

The ATIS data used in this work is organized by [9] into 17 unique intent categories, 129 slot types, and a vocabulary size of 943 . Some tasks have combined intents, which leads to a total of 26 intents. Moreover, the ATIS dataset from [9] contains 4978, 500, and 893 samples for training, validation and testing respectively.

Dataset	#Training	#Testing
ATIS	4978	893
Alarm	390	96
Audio	290	62
Iot	764	220

Table 5.1: Number of training and testing samples in ATIS dataset and domains in MASSIVE dataset.

Utterance	*How much is the cheapest flight from Boston to New York tomorrow morning?*
Goal:	Airfare
Cost_Relative	*cheapest*
Depart_City	*Boston*
Arrival_City	*New York*
Depart_Date.Relative	*tomorrow*
Depart_Time.Period	*morning*

Figure 5.1: An example utterance from the ATIS dataset [31].

Intent	Training Set	Test Set
Abbreviation	2.4%	3.6%
Aircraft	1.6%	0.9%
Airfare	9.0%	5.8%
Airline	3.4%	4.3%
Airport	0.5%	2.0%
Capacity	0.4%	2.4%
City	0.3%	0.6%
Day_Name	0.1%	0.1%
Distance	0.4%	1.1%
Flight	73.1%	71.6%
Flight_No	0.3%	1.0%
Flight_Time	1.2%	0.1%
Ground_Fare	0.4%	0.8%
Ground_Service	5.5%	4.0%
Meal	0.1%	0.6%
Quantity	1.1%	0.9%
Restriction	0.3%	0.1%

Figure 5.2: The frequency of intents for the training and test sets [31].

5.1.1 Statistics

Looking into the detail of ATIS dataset one can notice that it is in fact very unbalanced. [31] performs a deep analysis with respect to the statistics of ATIS intents, it has mentioned that the prior distribution of ATIS is heavily skewed, and the most frequent intent, `Flight` represents about 70% of the traffic. Figure 5.2 shows the frequency of the intents in the corpus for training and test sets.

Not only is the intent frequency strongly skewed, but the occurrences of the slot labels are also unbalanced. Figure 5.3 displays the occurrences of some selected slots in ATIS. For example, slot label `fromloc.city_name` occurs 4326 times in the training set, while slot label `transport_type` only 48 times.

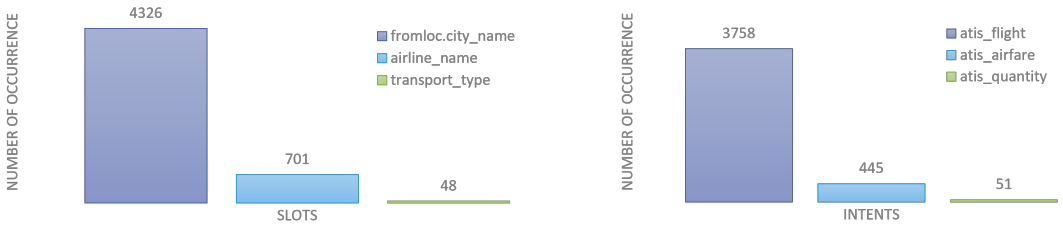

Figure 5.3: The number of occurrence of some intent and slot labels in ATIS training dataset.

Original Slot Label	Simplified Slot Label
B-depart_time.start_time	
I-depart_time.start_time	
B-depart_time.time	
I-depart_time.time	
B-depart_time.period_of_day	
B-depart_time.period_mod	*depart time*
B-depart_time.end_time	
I-depart_time.end_time	
B-depart_time.time_relative	
I-depart_time.time_relative	
B-toloc.state_code	
B-toloc.airport_code	
B-toloc.airport_name	
I-toloc.airport_name	
B-toloc.city_name	*to location*
I-toloc.city_name	
B-toloc.state_name	
I-toloc.state_name	

Table 5.2: An example of original and simplified slot labels.

Original Intent Label	Simplified Intent Label
atis_flight#atis_airfare *atis_airline#atis_flight_no* *atis_aircraft#atis_flight#atis_flight_no* *atis_airfare#atis_flight_time*	*flight information*
atis_ground_service#atis_ground_fare	*ground information*

Table 5.3: An example of original and simplified intent labels.

5.1.2 Data Processing

In the ATIS dataset, the utterances are labeled with IOB (Inside-Outside-Beginning) tags as slot labels. The `I-` prefix before a tag indicates that the tag is inside a chunk. An `O` tag indicates that a token belongs to no chunk. The `B-` prefix before a tag indicates that the tag is the beginning of a chunk that immediately follows another chunk without `O` tags between them. In the slot-filling task, when the token has `O` as its label, it indicates that this token does not belong to a slot. To make use of annotations as ground truth to train approaches in this work and to make a fair comparison between the model outputs, it is necessary to transfer the labels into a representation that is closer to natural language. Some IOB labels are excessively detailed, for example, information of a `return_date` is further divided into `return_date.date_relative`, `return_date.month_name`, and `return_date.day_number`. As the purpose of this work is to evaluate the approaches‘ ability to extract information rather than to do a token classification task, some labels that represent the same entity can be merged together. Furthermore, the ATIS dataset contains tasks with multi-intent such as `flight#airfare`, however, this work focuses on the task with a single intent.

To this end, in the data pre-processing step, the original slot labels are replaced by a simplified version of labels, and multi-intent labels are replaced by newly-defined intent labels. Some replacements are shown in table 5.2 and 5.3.

5.2 The MASSIVE dataset

MASSIVE dataset [8] is a Multilingual Amazon Slu resource package (SLURP [1]) for intent classification, slot-filling, and Virtual assistant evaluation. It contains 1M realistic, parallel, labeled virtual assistant utterances spanning 51 languages, 18 domains, 60 intents, and 55 slots. The authors created 50 new text corpora which represent 49 different spoken languages. In this work, only the English text corpus, which is mapped from the original dataset SLURP, is used. SLURP was collected for developing an in-home personal robot assistant, its utterances are directed at a device, rather than a person, which makes it suitable for building a task-oriented dialog system on a robot. The corpus primarily consists of interrogatives and imperatives rather than declarative utterances.

The dataset is organized into JSON lines, like the example shown in 5.1, some important keys of the JSON object are explained in the following:

Listing 5.1: An example of tasks in MASSIVE dataset.

```
{
    "id": "0",
    "locale": "en-US",
    "partition": "test",
    "scenario": "alarm",
    "intent": "alarm_set",
    "utt": "wake me up at five am this week",
    "annot_utt": "wake me up at [time : five am] [date : this week]",
    "worker_id": "1"
}
```

- `id` maps to the original ID in the SLURP collection. Mapping back to the SLURP en-US utterance, this utterance served as the basis for this localization.
- `locale` is the language and country code, in this case, English is used.
- `partition` is either `train`, `dev` or `test` and is used for training, validation, and testing accordingly.
- `scenario` is the general domain of an utterance. In this work, one scenario is seen as one domain that has its own set of intents and slots.
- `intent` is the domain specific intent of an utterance within a domain formatted as {`scenario`}_{`intent`}.
- `utt` is the raw utterance text without annotations.
- `annot_utt` is the text from `utt` with slot annotations formatted as [{`label`} : {`entity`}].

5.2.1 Statistics

Scenario	Intents	Slots
Alarm	3	13
Weather	1	9
Iot	9	14
Music	4	14
Takeaway	2	14

Table 5.4: Number of intents and slots in different scenarios.

As each scenario in MASSIVE has its own set of intents and slots, a single scenario is seen as a specific task. The table shows some scenarios and their number of intents and

slots. Some scenarios such as `weather` and `news` only have one intent, however, a scenario like `iot` has 9 intents. Different scenarios also have different numbers of slots. Overall, there are significant differences between certain tasks, which makes MASSIVE suitable to evaluate the approaches' transfer ability to different tasks.

5.2.2 Data Pre-processing

A scenario in the MASSIVE dataset defines a task domain, and therefore the pre-processing for each scenario is performed separately. As shown in Listing 5.1, the slot annotation of MASSIVE data is formatted as [{`label`} : {`entity`}]. In order to use it as training data for fine-tuning a pre-trained BERT model, and constructing consistent ground truth for all three approaches, the data is processed into the token-wise annotation. The utterance is first split into words and then "BOS" and "EOS" tokens are added to the beginning and end respectively. Each utterance has one intent and each token in the utterance will be assigned a label. For the `entities` that are annotated with slot `labels`, the `label` will be assigned to every word of the `entity`. And other not labeled words are labeled with `O`. "BOS" and "EOS" tokens are labeled as "[CLS]" and "[SEP]". Furthermore, either semantic parsing or GPT-3 prefers information in natural language, instead of directly using the original labels of the MASSIVE dataset, they are rewritten into natural language. For example, "alarm_set" is changed to "set alarm". An example of a data entry after pre-processing is shown in 5.2.

Listing 5.2: An example of pre-processed data in scenario "alarm" from MASSIVE dataset.

```
{
    "id": 3379,
    "intent": "set alarm",
    "text": ["BOS", "is", "my", "alarm", "set", "for", "seven", "am", "EOS"],
    "labels": ["[CLS]", "O", "O", "O", "O", "O", "time", "time", "[SEP]"]
}
```

6 Experiment and Evaluation

In the previous chapters, the implementation of the semantic parsing approach, prompt construction for GPT-3 model, and fine-tuning of the pre-trained BERT model is explained in detail. Furthermore, two datasets (ATIS and MASSIVE) that are popular for natural language understanding tasks are selected for evaluating three approaches. In this chapter, experiments are designed to test and analyze three approaches from different perspectives. In the first section, the experimental setups are introduced, and then the metrics that are used for evaluation are introduced in the second section. Finally, the results from different models under various settings are compared and analyzed.

6.1 Experimental Setup

To obtain comprehensive knowledge of how to design a better natural language understanding component based on the three approaches purposed in this work, it is essential to test them from different perspectives. In this section, we introduce the data that are selected for experiments and how it is used for each approach. Experiments are designed for each method and comparisons between approaches are made after analyzing the results individually.

6.1.1 Few-shot Datasets

To test the approaches under a few-shot setting, the ATIS and MASSIVE dataset introduced in chapter 5 are used. Although they are both popular for intent classification and slot-filling tasks, ATIS is a single-domain dataset that provides tasks concerning airline travel information, and MASSIVE in another hand, contains tasks in multiple domains for a virtual assistant. Because both datasets are oriented in two different general domains of human life (airline and household), it is meaningful to explore the performance of each approach and analyze its ability to transfer to a new domain. Furthermore, we can also observe how an approach performs on a similar domain by using the tasks from different domains in the MASSIVE dataset.

1. For the ATIS dataset, a few sets of simplified labels are designed to overcome the problem that its labels are too much into detail and lead to even fewer samples per intent and slot due to its skewed data distribution. In this chapter, one set of designed simplified labels is used for experiments on the three approaches.

2. In the MASSIVE dataset, domain `alarm`, `audio`, `iot`, `weather`, `takeaway` and `music` are selected to test the performance of semantic parsing approach, and the first three are also used to test the other two approaches. Each domain focuses on a specific

task, for example, the domain `alarm` contains tasks for setting an alarm, and the domain `iot` is about controlling devices in the household.

6.1.2 Experiments on Semantic Parsing

Semantic parsing is a zero-shot approach that acquires no training data. We test it on ATIS as well as on the `alarm`, `audio`, `iot`, `music`, `news`, `takeaway` and `weather` domains in MASSIVE. For all experiments performed on the semantic parsing approach, the structure of the approach as well as the designed heuristic rules are not changed.

6.1.3 Experiments on Pre-trained BERT Model

Each domain has its own set of intents and slots. For a pre-trained model like BERT, this means a different number of intent and slot classes that need to be predicted by the model. Thus, tasks in different domains can share the same encoder but not the same classifier. The dimension variation of the classifier leads to the fact that it can not be directly applied to a new domain without adapting to new intents and slots. In other words, zero-shot testing is not feasible for fine-tuning a pre-trained BERT approach. Therefore, experiments for fine-tuning a pre-trained BERT are focused on few-shot learning. A multi-task BERT model that is introduced in section 2.3.1 is initialized with the pre-trained BERT model `bert-base-uncased` provided by Hugging-face, the model after initialization is referred to as `BERT-v0`. The following experiments are designed for fine-tuning a pre-trained BERT model. In all training processes, we use the Adam optimizer [13] for optimizing the parameters, and we use the greedy method to decode the output from classification layers.

Experiments on the ATIS dataset To enable the model to perform the tasks on the ATIS dataset, we fine-tune the `BERT-v0` model on the ATIS dataset with a max-epoch of three, a learning rate of $2.99e-05$, and a batch size of 3982. The learning rate and batch size are automatically found by PyTorch Lightning. The parameters in both encoder and classifiers are updated in the fine-tuning process. The fine-tuned model is referred to as `BERT-v1` in the later content.

Experiments on the MASSIVE dataset To explore the model's ability to adapt to a new task and how a pre-training on a similar task may help the adapting process, we train the classification layers of `BERT-v0` and `BERT-v1` with the domain data in the MASSIVE dataset respectively. All models are trained with a max-epoch of 300 and with early stopping.

Experiments with few-shot learning The above-mentioned training on both datasets made use of all the available training data, however, we want to explore the pre-trained approach‘s ability to adapt to a new task with a few examples. For this purpose, we fine-tuned `BERT-v0` and `BERT-v1` on 12, 20, 30, 50, 100, 200, and 300 examples separately.

6.1.4 Experiments on GPT-3

To test how GPT-3 performs on intent classification and slot-filling tasks and how the performance is affected by the prompt design, different prompt templates are designed to

construct prompts for GPT-3. GPT-3 is also tested on both the ATIS dataset and domains in the MASSIVE dataset. In all experiments, we use the model `text-davinci-003` with the `temperature` set to 0 and select the response greedily.

Experiments on the ATIS dataset Considering that ATIS has the most testing samples compared to other domains and the cost of time and money testing on GPT-3 is high, we test two settings on the ATIS dataset, GPT-3 without examples and keys information and GPT-3 with examples and keys information. For each domain, a set of examples is selected from the training dataset. We iterate through the intent and slot labels of that domain and randomly choose one example for each label. After constructing the prompt, it will be sent to GPT-3.

Experiments on the MASSIVE dataset We tested the GPT-3 approach on the same domains as the other two approaches with different prompts on the domains `alarm`, `audio` and `iot`, in order to explore how different prompts affect the performance of GPT-3, and how the domain-specific information increases its performance.

6.2 MUC-5 Evaluation Metrics

We use the metrics for the Fifth Message Understanding Conference (MUC-5) [5] for evaluation, which are originally designed for evaluating information extraction systems. To evaluate each approach, the outputs are compared to the ground truth constructed from the datasets‘ original annotations. The results are evaluated based on slots, since each task has an intent, the intent can be seen as a specific slot. Given a specific task and the prediction and ground truth of this task:

Correct (COR) If a slot exists in both prediction and ground truth and their values are equivalent, the category is correct (COR).

Partial (PAR) If a slot exists in both prediction and ground truth and their values are judged as a near match, the category is partial (PAR). It is a near match when both values contain the same words.

Incorrect (INC) If a slot exists in both prediction and ground truth and their values do not match, the category is incorrect (INC).

Missing (MIS) If a slot exits in the ground truth but not in the prediction, the category is missing (MIS).

Spurious (SPU) If a slot exists in prediction but ground truth does not contain this slot, the category is spurious (SPU).

Some intermediate metrics can be calculated using five metrics above:

$$total = INC + PAR + INC + MIS + SPU \tag{6.1}$$

$$possible = COR + PAR + INC \tag{6.2}$$

$$actual = COR + PAR + INC + SPU \tag{6.3}$$

Precision and recall can be calculated by

$$recall = \frac{correct + (partial \times 0.5)}{possible} \tag{6.4}$$

$$precision = \frac{correct + (partial \times 0.5)}{actual} \tag{6.5}$$

$$\tag{6.6}$$

In this work, the results of approaches are evaluated using accuracy and F1 score, which are calculated by

$$Accuracy = \frac{COR + PAR}{total} \tag{6.7}$$

$$F1 = \frac{precision \times recall}{precision + recall} \tag{6.8}$$

Furthermore, the evaluation methods can be divided into two ways, approach level evaluation and label level evaluation. In approach level evaluation, MUC-5 evaluation metrics are calculated based on all slots (intent and slot labels) of the evaluated approach on each testing dataset, and an accuracy and F1 score for the approach is calculated. In other words, each approach has one accuracy and one F1 score on each testing dataset. However, to determine how an approach performs on specific labels, we also evaluated some approaches on a slot level, where the accuracy and F1 scores of every label are calculated. Both of the evaluation methods will be used in section 6.3.

6.3 Evaluation

In this section, the experimental results are shown and evaluated with the MUC-5 metrics mentioned in section 6.2 and are analyzed from different perspectives.

6.3.1 Results of Semantic Parsing

Dataset	Accuracy	F1
ATIS	0.295	0.266
Alarm	0.178	0.243
Audio	0.297	0.400
Iot	0.330	0.372
Weather	0.334	0.361
Takeaway	0.108	0.133
Music	0.080	0.122

Table 6.1: Results of semantic parsing approach tasks in ATIS dataset and tasks in domains of MASSIVE dataset.

Table 6.1 shows the accuracy and F1 score of the semantic parsing approach on tasks in different domains rounded to the third decimal. The `alarm`, `audio`, `weather`, `iot`, `music` and `takeaway` datasets are subsets of the MASSIVE dataset, each of them containing tasks in a different domain. The scores are calculated based on both intent classification and slot-filling results. As we can tell from the scores, the semantic parsing approach performs better on some domains than other domains. For some domains in the MASSIVE dataset such as `audio` and `weather`, the results are better than for ATIS, and for other domains like `music`, it performs a lot worse.

If we want to have a deeper understanding of the results, we have to take a look at the results on each label in the domain. Table 6.2 and 6.3 show the results of some intent and slot labels in the ATIS dataset. We can see that the semantic parsing approach performs better on some of the label types such as `flight time`, `from location`, yet worse on others such as `ground service` and `relative to today`. After observing the prediction of intent and slot phrases of semantic parsing, we can easily find out that the semantic parsing approach works better at classifying intent and slot labels that are close to the original utterance text and struggles with abstract concepts.

For example, in the original utterance, the user may ask “Can you find me the flight from Boston to Atlanta?”, `Boston` and `Atlanta` are recognized as `location` by NER, the phrases "from Boston" and "to Atlanta" will be extracted from the utterance and will be classified as "from location" and "to location". However, due to semantic parsing‘s lack of ability to understand the whole sentence, it is very difficult for semantic parsing to classify labels like `relative to today`.

Intent	**Accuracy**	**F1**
Flight time	0.867	0.464
Flight information	0.536	0.319
Airfare	0.187	0.248
City	0.018	0.018
Ground service	0.0	0.0

Table 6.2: Results of semantic parsing approach on different intents in ATIS dataset.

Slot	**Accuracy**	**F**
From location	0.762	0.415
To location	0.603	0.362
Airport name	0.242	0.186
Arrive time	0.0294	0.034
Relative to today	0.0	0.0

Table 6.3: Results of semantic parsing approach on different slots in ATIS dataset.

Furthermore, when we take a look at the results of different intents and slots in domain `alarm` shown in Table 6.4 and 6.5, the same situation appears. The approach is much better

Intent	Accuracy	F1
Query alarm	0.061	0.077
Remove alarm	0.667	0.667
Set alarm	0.443	0.486

Table 6.4: Results of intents on domain `alarm` in MASSIVE dataset with semantic parsing approach.

Slot	Accuracy	F1
Time of day	0.375	0.273
Time	0.021	0.020
Date	0.167	0.116
Device type	0.0	0.0
Event name	0.0	0.0

Table 6.5: Results of slots on domain `alarm` in MASSIVE dataset with semantic parsing approach.

at predicting the intent `remove alarm` than `query alarm`. The reason is when we want to delete an alarm, it is natural that we use the phrase “remove the alarm”, but when we want to query the alarm information, we usually do not use the word “query”. The approach is bad at predicting such an abstract concept like “query”. The same conclusion applies to the slot-filling tasks. Another example is in the domain `music`, "queen" should be recognized as an "artist_name" and "barcelona" should be recognized as a "song_name", the labels and the actual tokens in the utterance do not have a close similarity to these labels based on plain text. In conclusion, the naming of the intent and slot labels has a great impact on the performance. The semantic parsing approach works better at predicting labels that are named closely to the natural language and worse at predicting the labels named after abstract concepts.

6.3.2 Results of Fine-tuning a Pre-trained BERT Model

Dataset	Accuracy	F1
`ATIS`	0.918	0.946

Table 6.6: Results of fine-tuned BERT on tasks in ATIS.

The BERT-v1 is the model fine-tuned on the ATIS dataset based on the pre-trained BERT model BERT-v0. Table 6.6 shows that BERT-v1 performs excellently on tasks from testing data in ATIS with an accuracy of 0.918 and an F1 score of 0.946, since the ATIS dataset has the most training samples compared to the domains in the MASSIVE dataset. With

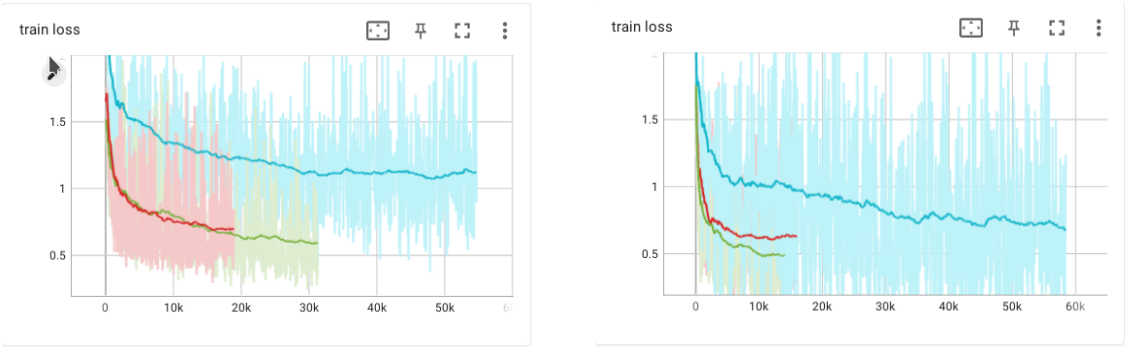

Figure 6.1: Train losses of models of domain `alarm` (green), `audio` (red) and `iot` (blue) in fine-tuned on BERT-v0 (left) and BERT-v1 (right).

sufficient training samples, the fine-tuned BERT can have a decent performance on the tasks.

To explore how well a pre-trained model adapts to a new task and how pre-training on a similar task benefits the adaptation, Figure 6.1 shows the training loss of the models fine-tuned on BERT-v0 and BERT-v1 in domain `alarm`, `audio` and `iot`. In both case, only the parameters in the classifiers are trained. It can be seen that the models trained on BERT-v1 have steeper training loss curves, indicating that it can adapt to a new task faster than training directly on BERT-v0.

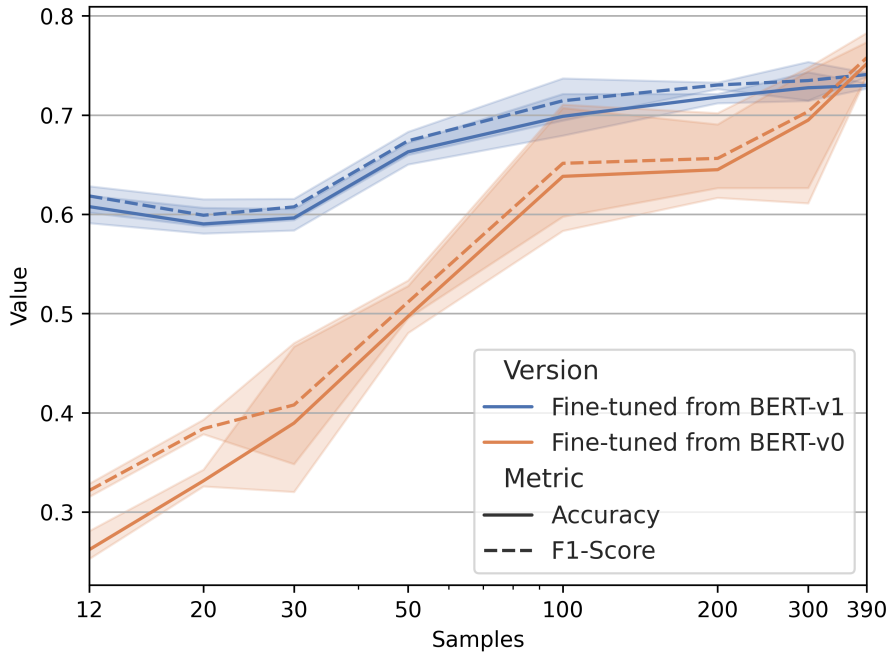

Figure 6.2: Results of models fine-tuned from BERT-v0 and BERT-v1 on the `alarm` domain in the MASSIVE dataset.

In order to gain a deeper understanding about the difference between fine-tuning on the model with pre-training on a similar task and without, numerous experiments are

#Shot	With Keys	Accuracy	F1
0	✘	0.109	0.099
1/41	✔	0.566	0.670

Table 6.7: Results of prompt `Info_Extractor` on tasks in ATIS dataset.

conducted on the `alarm` domain with training samples from 12 to 390. The result is shown in Figure 6.2. The X-axis shows the number of examples used to fine-tune the model, and the Y-axis shows the value of the scores for each model. We trained each model three times and the average accuracy and F1 score is plotted using solid and dashed lines respectively. The orange lines show the results of the models fine-tuned from the original pre-trained BERT (BERT-v0) and the blue ones show results of models fine-tuned from the model that is pre-trained on ATIS (BERT-v1). When only 12 samples are available for training, pre-training on a similar task helps a lot, and the model fine-tuned on BERT-v1 performs much better than the one without task-specific pre-training. Given more examples, the performance of models trained from BERT-v0 improves a lot, while the ones trained on BERT-v1 improve at a slower rate. Finally, when we use all 390 samples for fine-tuning on both models, the model trained on BERT-v0 outperforms the one trained on BERT-v1.

With the above observation from our experiments, we can conclude that, if we have only little data, it is beneficial to pre-train the model on data of similar tasks. However, if sufficient data is provided for a specific task, it might be better to directly train the model using samples from this task instead of using a pre-trained model. The pre-training process can reduce the ability of model to adapt to a new task.

6.3.3 Results on Dynamic Prompt Construction on GPT-3

GPT-3 with different designed prompts is tested on the ATIS dataset and selected domains in the MASSIVE dataset. Table 6.7 shows its results on ATIS dataset with the prompt template `Info_Extractor`. When neither examples nor key information (possible intent and slot labels) is provided to GPT-3, it is not able to make an accurate prediction. However, when both one example and key information are provided in the prompt, the accuracy and F1 score of GPT-3 increased by 0.457 and 0.571 respectively. This suggests that for the GPT-3 model, one example can largely improve its ability in solving intent classification and slot-filling tasks.

We further tested all the designed prompt templates on tasks in the domain `alarm` in the MASSIVE dataset. The difference of the prompts are whether they select an example for prompt construction and whether they contain information of the possible intent and slot categories. The results for different prompts are listed in Table 6.8. Number of shots indicate how many examples are provided for constructing a prompt for GPT-3, `1/12` means that in the domain `alarm`, 12 examples are provided and one of the examples is selected to construct a prompt for GPT-3. The same concept applies to the other domains in the following content.

We can observe that for each prompt template, adding one example selected from the given examples can improve GPT-3's performance. All four combinations are tested with

Prompt Template	#Shot	With Keys	Accuracy	F1
Info_Extractor	0	✘	0.052	0.026
	0	✔	0.483	0.314
	1/12	✘	0.463	0.532
	1/12	✔	0.669	0.776
NLU_Machine	1/12	✘	0.438	0.507
	1/12	✔	0.643	0.751
Robot_Translator	0	✘	0.114	0.149
	1/12	✔	**0.686**	**0.781**

Table 6.8: Results of GPT-3 with different prompts on domain alarm in MASSIVE dataset.

the Info_Extractor prompt template. Although adding only key information can already improve the performance, the model works better when an example is provided. When both example and key information are provided, the model reaches the best performance. For the other two prompt templates NLU_Machine and Robot_Translator, the same results have shown.

Table 6.9: Responses from GPT-3 with prompt template Info_Extractor.

User utterance	i would like to find a flight from charlotte to las vegas that makes a stop in st.louis
Without example	\nIntent: FindFlight \nSlots: \n{\n \"From\": \"Charlotte\", \n \"To\": \"Las Vegas\", \n \"Stop\": \"St. Louis\"\n}
With example	intent:flight;from location:charlotte;to location:las vegas;stop location:st. louis;

If we have a closer look at the responses GPT-3 generated, we can see that when no example and key information are given, GPT-3 can already extract the important information from the original utterance. However, without any knowledge about the intent and slot labels, it is not able to predict the right intent and slot categories for the extracted information. Furthermore, it outputs the response in free text that contains a lot of special characters, which also increases the difficulty for finding a match in the ground truth. With an example, however, GPT-3 tries to output the response in the format that is shown by the given example. In other words, GPT-3 has the ability to extract correct information from the user utterance even without an example and other extra information, but it fails at producing a response that can be directly used in the next step. This also explains why only one example and extra key information can help a lot at increasing its performance.

Figure 6.10 shows the results of GPT-3 with the prompt template Robot_Translator on the domains audio and iot. For both domains, adding an example into the prompt can lead to better performance on the tasks. If we compare the results with both example and key information on the domains audio and iot to the domain alarm, we can see that GPT-3 works better on alarm and worse on audio. Hence, we can conclude that GPT-3 tackles tasks in some domains better than others.

Domain	#Shot	Accuracy	F1
Audio	0	0.364	0.492
	1/8	0.523	0.657
Iot	0	0.557	0.652
	1/14	0.645	0.720

Table 6.10: Results of GPT-3 with prompt template `Robot_Translator` on domain `audio`, `iot` in MASSIVE dataset with key information.

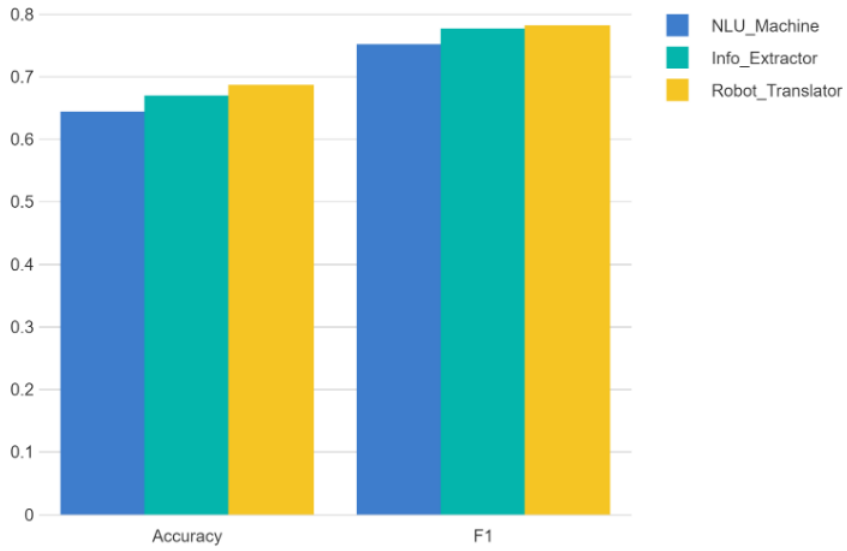

Figure 6.3: Accuracy and F1 scores of GPT-3 with different prompt templates on domain `alarm` in MASSIVE dataset.

Domain	Semantic Parsing		GPT-3 with Prompt	
	Accuracy	F1	Accuracy	F1
ATIS	0.295	0.266	0.109	0.098
Alarm	0.178	0.243	0.483	0.314

Table 6.11: Accuracy and F1 scores of semantic parsing and GPT-3 in zero-shot setting.

Domain	Pre-traing (12)		GPT-3 with Prompt	
	Accuracy	F1	Accuracy	F1
Alarm	0.608	0.618	0.669	0.776

Table 6.12: Accuracy and F1 scores of fine-tuning a pre-trained model and GPT-3 in few-shot setting.

Figure 6.3 compares the accuracy and F1 scores of GPT-3 with different prompt templates with example and key information provided. Three different prompt templates are used on tasks in the domain alarm. An example and key information of the domain is provided for all experiments. The results show that GPT-3 using the prompt template NLU_Extractor has the worst performance, while GPT-3 with the prompt template Robot_Translator has the best performance. This indicates that GPT-3 tackles the tasks better when an appropriate prompt is designed. With this in mind, we should not ignore the importance of prompt design when working with GPT-3.

6.3.4 Comparison of Three Approaches

From the three proposed approaches, the semantic parsing approach works without any training examples, so it is a zero-shot approach, and GPT-3 with prompt can also work under a zero-shot setting when no example is provided while making predictions. Therefore, we can compare these two approaches under a zero-shot setting. Due to the reason that semantic parsing needs to know the intent and slot labels to make a prediction, we also provide GPT-3 the key information in the prompt to make a fair comparison.

Table 6.11 shows the results of semantic parsing and GPT-3 in a zero-shot setting. For both ATIS dataset, we only tested GPT-3 without key information and we can see that semantic parsing performs better compared to GPT-3 without domain-specific information. However, in the domain alarm key information is provided, and GPT-3 outperforms semantic parsing. We can conclude that when both approaches have same knowledge about the tasks in a specific domain, GPT-3 works better at identifying the user intent and filling in the slots.

When we provide one example for constructing prompt for GPT-3, the example is selected from a set of examples. In the domain alarm, 12 examples are available for selection. Therefore, we can compare GPT-3 with prompt with pre-trained model fine-tuned using 12 examples. The results of both approaches are listed in Table 6.12. The key information is provided to GPT-3 because the fine-tuned model has information about the intent and slot classes. The results show that GPT-3 performs better at intent classification and slot-filling tasks in the domain alarm. Although the model in the fine-tuning approach

was already pre-trained on the ATIS dataset, it still performs worse on these tasks than GPT-3.

In conclusion, each approach has its own strengths and weaknesses. Semantic parsing does not require any training, therefore requires no training data, and for tasks in new domain, we do not need to alter its structure. It is suitable for the type of tasks that is defined close to natural language and can not handle abstract concepts. The semantic parsing approach can be improved by using parsers with specific domain knowledge, so that it can recognize the entities in a specific domain. Furthermore, if better heuristic rules are designed for semantic parsing, the performance can also be increased.

As for the appraoch of fine-tuning a pre-trained model, the model can achieve a decent performance when sufficient data is provided, or alternatively, pre-training the model on a similar task can also increase the model performance. In this work, we use pre-trained BERT as the backbone model of our approach, but we can also utilize other language models for the task. The downside of pre-trained models is that for each new domain, the model has to adapt to the tasks by re-structuring and training its classification layers, and this adaptation process usually requires sufficient data.

Same as the semantic parsing approach, GPT-3 with prompt also requires no training process, although we need to provide some information to the model so that it can generate useful output. GPT-3 shows its power with little information, and it generates the output that is most human-like. However, in order to generate a prompt that can maximize the correctness of the model output, extensive experiments and testing are required, and with only the help of few examples and a prompt, the model still can not reach the performance of a pre-trained model trained with sufficient data. Furthermore, GPT-3 generates responses as free text, which causes difficulties when processing the data into our required format. Another major downside is that the model is very expensive to use, considering that GPT-3 charges per token. We want to describe the tasks in a specific domain in detail, so that GPT-3 generates responses in the format we need, yet it increases the token size of the prompt, therefore will result in a higher cost using GPT-3.

7 Conclusion

This thesis proposes three approaches for solving intent classification and slot-filling tasks in the Natural Language Understanding (NLU) component of a task-oriented dialog system. The semantic parsing approach utilizes existing parsing methods such as part-of-speech parsing, dependency parsing, and named-entity recognition to analyze the grammatical structure of the user utterance and extract phrases that can represent the user intent and slot information that can be further used to compare with the pre-defined labels and make classifications based on the similarity between them. The second method is to make use of a pre-trained BERT model and consider the intent classification task as a sequence classification task and the slot-filling task as a token classification task. We add two classification layers for both tasks and fine-tune the model on the dataset of the new domain to adapt to new tasks. As large language models show their power on different sub-stream tasks, we use GPT-3 with dynamically constructed prompts to solve NLU tasks in different domains. If examples are available, we select one example that is most similar to the given user utterance using sentence-BERT combined with cosine-similarity.

We have evaluated three approaches on the ATIS dataset and the MASSIVE dataset. Differing from ATIS, which contains tasks in one domain, the MASSIVE dataset has a collection of tasks in 18 different domains. We selected some of the domains to evaluate the approaches. Experiments are conducted to explore the performance of each approach on zero-shot and few-shot learning.

Semantic parsing, as a baseline for the zero-shot methods, does not perform well in intent classification and slot-filling tasks. It is not feasible to enable a pre-trained model to solve tasks in new domains without any examples due to its structural nature. However, GPT-3 outperforms the semantic parsing approach when the intents and slots information is provided. For tasks that have limited examples, we can utilize GPT-3 with an example provided in the prompt and few-shot learning with a pre-trained BERT model. When we adapt the pre-trained model directly to a new domain, it can not achieve good performance without sufficient training data. However, the model works much better with few examples if it is first pre-trained on a similar task. And yet, if we provide the same amount of data to the BERT model pre-trained on a similar task and GPT-3 with the prompt, although only one example is selected for the prompt construction for GPT-3, GPT-3 still outperforms the fine-tuned pre-trained model.

In conclusion, we can utilize semantic parsing in tasks that are defined close to the utterance a user may use in the command and avoid the tasks that contain a lot abstract concepts. Fine-tuning a pre-trained method, however, should be applied to tasks where enough annotated data is available. Alternatively, we can also pre-train the model on similar tasks in advance so that it can adapt to new tasks with few examples. GPT-3 shows its power with prompts that contain domain-specific information. It is important to define the desired output format in the prompt so that GPT-3 produces responses that can be

easily used in the down-stream components of NLU without a complicated post-processing step. In the meantime, we also need to pay attention to the cost of using GPT-3, as it is not free of charge.

In future work, for semantic parsing approach, we can apply parsers that have specific domain knowledge and design better heuristic rules for extracting the intent and slot phrases. As for fine-tuning a pre-trained model, we can utilize other language models and use them as backbone models for fine-tuning for new tasks. To reduce the experimental costs, we can combine prompt design with other large language models such as OPT [39] and BLOOM [35] instead of GPT-3.

Bibliography

[1] Emanuele Bastianelli et al. *SLURP: A Spoken Language Understanding Resource Package*. 2020. DOI: 10.48550/ARXIV.2011.13205. URL: https://arxiv.org/abs/2011.13205.

[2] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. DOI: 10.48550/ARXIV.2005.14165. URL: https://arxiv.org/abs/2005.14165.

[3] Paweł Budzianowski and Ivan Vulic. "Hello, It's GPT-2 - How Can I Help You? Towards the Use of Pretrained Language Models for Task-Oriented Dialogue Systems". In: *EMNLP*. 2019.

[4] Qian Chen, Zhu Zhuo, and Wen Wang. *BERT for Joint Intent Classification and Slot Filling*. 2019. DOI: 10.48550/ARXIV.1902.10909. URL: https://arxiv.org/abs/1902.10909.

[5] Nancy Chinchor and Beth Sundheim. "MUC-5 Evaluation Metrics". In: *Fifth Message Understanding Conference (MUC-5): Proceedings of a Conference Held in Baltimore, Maryland, August 25-27, 1993*. 1993. URL: https://aclanthology.org/M93-1007.

[6] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[7] Arash Einolghozati et al. *Improving Semantic Parsing for Task Oriented Dialog*. 2019. DOI: 10.48550/ARXIV.1902.06000. URL: https://arxiv.org/abs/1902.06000.

[8] Jack FitzGerald et al. *MASSIVE: A 1M-Example Multilingual Natural Language Understanding Dataset with 51 Typologically-Diverse Languages*. 2022. DOI: 10.48550/ARXIV.2204.08582. URL: https://arxiv.org/abs/2204.08582.

[9] Dilek Hakkani-Tür et al. "Multi-Domain Joint Semantic Frame Parsing using Bi-directional RNN-LSTM". In: *Proceedings of The 17th Annual Meeting of the International Speech Communication Association (INTERSPEECH 2016)*. ISCA, June 2016. URL: https://www.microsoft.com/en-us/research/publication/multijoint/.

[10] Dong-hyun Ham et al. "End-to-End Neural Pipeline for Goal-Oriented Dialogue Systems using GPT-2". In: *ACL*. 2020.

[11] Charles T. Hemphill, John J. Godfrey, and George R. Doddington. "The ATIS Spoken Language Systems Pilot Corpus". In: *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*. 1990. URL: https://aclanthology.org/H90-1021.

[12] Jiawei Huang, Tingting He, and Xinhui Tu. "Dialogue intent classification with character-CNN-BGRU networks". In: *Multimedia Tools and Applications* 79 (Feb. 2020). DOI: 10.1007/s11042-019-7678-1.

[13] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[14] Takeshi Kojima et al. *Large Language Models are Zero-Shot Reasoners*. 2022. DOI: 10.48550/ARXIV.2205.11916. URL: https://arxiv.org/abs/2205.11916.

[15] Pengfei Liu et al. *Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing*. 2021. DOI: 10.48550/ARXIV.2107.13586. URL: https://arxiv.org/abs/2107.13586.

[16] Elman Mansimov and Yi Zhang. *Semantic Parsing in Task-Oriented Dialog with Recursive Insertion-based Encoder*. 2021. DOI: 10.48550/ARXIV.2109.04500. URL: https://arxiv.org/abs/2109.04500.

[17] Fei Mi et al. *CINS: Comprehensive Instruction for Few-shot Learning in Task-oriented Dialog Systems*. 2021. arXiv: 2109.04645 [cs.CL].

[18] Fei Mi et al. *Self-training Improves Pre-training for Few-shot Learning in Task-oriented Dialog Systems*. 2021. arXiv: 2108.12589 [cs.CL].

[19] David Nadeau and Satoshi Sekine. "A Survey of Named Entity Recognition and Classification". In: *Lingvisticae Investigationes* 30 (Aug. 2007). DOI: 10.1075/li.30.1.03nad.

[20] *Named-entity recognition*. https://en.wikipedia.org/wiki/Named-entity_recognition.

[21] Timothy Osborne. *A Dependency Grammar of English: An introduction and beyond*. John Benjamins, 2019. URL: https://www.jbe-platform.com/content/books/9789027262288.

[22] Baolin Peng et al. *Few-shot Natural Language Generation for Task-Oriented Dialog*. 2020. arXiv: 2002.12328 [cs.CL].

[23] Baolin Peng et al. *SOLOIST: Building Task Bots at Scale with Transfer Learning and Machine Teaching*. 2021. arXiv: 2005.05298 [cs.CL].

[24] *POS tags*. https://www.sketchengine.eu/blog/pos-tags/. Sketch Engine. Lexical Computing. 2018-03-27. Retrieved 2018-04-06.

[25] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2019. DOI: 10.48550/ARXIV.1910.10683. URL: https://arxiv.org/abs/1910.10683.

[26] Nils Reimers and Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. 2019. DOI: 10.48550/ARXIV.1908.10084. URL: https://arxiv.org/abs/1908.10084.

[27] Alex Sherstinsky. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network". In: *Physica D: Nonlinear Phenomena* 404 (Mar. 2020), p. 132306. DOI: 10.1016/j.physd.2019.132306. URL: https://doi.org/10.1016%2Fj.physd.2019.132306.

[28] Charlie Snell et al. *Context-Aware Language Modeling for Goal-Oriented Dialogue Systems*. 2022. DOI: 10.48550/ARXIV.2204.10198. URL: https://arxiv.org/abs/2204.10198.

[29] Yixuan Su et al. *Multi-Task Pre-Training for Plug-and-Play Task-Oriented Dialogue System*. 2021. DOI: 10.48550/ARXIV.2109.14739. URL: https://arxiv.org/abs/2109.14739.

[30] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. DOI: 10.48550/ARXIV.1409.3215. URL: https://arxiv.org/abs/1409.3215.

[31] Gokhan Tur, Dilek Hakkani-Tur, and Larry Heck. "What is left to be understood in ATIS?" In: Jan. 2011, pp. 19–24. DOI: 10.1109/SLT.2010.5700816.

[32] A. Waibel et al. "Phoneme recognition using time-delay neural networks". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.3 (1989), pp. 328–339. DOI: 10.1109/29.21701.

[33] Yaqing Wang et al. *Adaptive Self-training for Few-shot Neural Sequence Labeling*. 2020. DOI: 10.48550/ARXIV.2010.03680. URL: https://arxiv.org/abs/2010.03680.

[34] Yaqing Wang et al. "Meta Self-training for Few-shot Neural Sequence Labeling". In: *SIGKDD 2021 (Research Track)*. Aug. 2021. URL: https://www.microsoft.com/en-us/research/publication/adaptive-self-training-for-few-shot-neural-sequence-labeling/.

[35] BigScience Workshop et al. *BLOOM: A 176B-Parameter Open-Access Multilingual Language Model*. 2023. arXiv: 2211.05100 [cs.CL].

[36] Xuesong Yang et al. "End-to-end joint learning of natural language understanding and dialogue manager". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2017), pp. 5690–5694.

[37] Yunyi Yang, Yunhao Li, and Xiaojun Quan. *UBAR: Towards Fully End-to-End Task-Oriented Dialog Systems with GPT-2*. 2020. DOI: 10.48550/ARXIV.2012.03539. URL: https://arxiv.org/abs/2012.03539.

[38] Jian-Guo Zhang et al. *Find or Classify? Dual Strategy for Slot-Value Predictions on Multi-Domain Dialog State Tracking*. 2019. DOI: 10.48550/ARXIV.1910.03544. URL: https://arxiv.org/abs/1910.03544.

[39] Susan Zhang et al. *OPT: Open Pre-trained Transformer Language Models*. 2022. arXiv: 2205.01068 [cs.CL].

[40] Zheng Zhang et al. *Recent Advances and Challenges in Task-oriented Dialog System*. 2020. DOI: 10.48550/ARXIV.2003.07490. URL: https://arxiv.org/abs/2003.07490.