Extended Abstract

In the post-pandemic era, the online education market embraces a boom with productive investment and increasing opportunities. This project aims at developing a speech dialogue system based on an NLU model that can understand task-oriented natural language texts to provide education-related services. It is believed that the system will assist in reducing the communication costs between students and teachers in the online teaching environment, ease the shift of students from physical to virtual classrooms, and alleviate the pressure on the teaching staff.

The past few years have witnessed a significant breakthrough in Natural Language Processing with advanced Deep Neural Network techniques; therefore, this project explores neural-network-based natural language processing methods for information extraction and task identification. A novel SOTA joint training approach for intent classification and slot filling is adopted in the development of the model. In order to incisively understand the model, great efforts are paid to experiments for different components of this model, including the alternation of BERT variants, the adoption of CRF, the optimal value of hyperparameters such as the number of training epochs, and the value of slot loss coefficient. A custom intent and slot filling dataset with goal-oriented corpus in the specific application scenarios of online teaching platforms are generated for model training due to the lack of such ready-made education-focused data resources.

The implementation of the front-end application is designed to cross various platforms with the support of *Flutter*, from web pages to mobile devices, which is innovative compared with most of the on-shelf virtual assistants that are dependent on operating systems such as Apple Siri, Microsoft Cortana, or Huawei Xiao Ai. The back-end system, based on Python *Flask* library, is integrated with the mentioned model trained on the custom dataset to provide support for intent identification and critical information capture.

Acknowledgements

I would like to express my extreme gratitude to Dr. Song Linqi, my supervisor, for his guidance in developing the basic idea into a creative application and visionary suggestions that inspired the innovation of this project. I am also extremely grateful to Ms. Liu Shuqi, a current Ph.D. candidate in the Computer Science Department. She provides me with excellent assistance on theoretical and practical details in every project stage. This project would never reach its current status without their supervision and advice.

I would also like to thank the support and resources from the department and CS Lab, without whom the proceeding of the project would not be so smooth. The curriculum of my program, Research Mentor Scheme, and in-campus part-time job develop my ability and prepare me for the final year project. The devices in CS Lab enable me to handle the experiments with a tight schedule.

I also wish to acknowledge the help of my classmates, without the discussion with whom some features in this project would never have been inspired. Besides, I am grateful for my family, which offer me financial support and regard for my interest and decision in my study. Finally, I would like to show my appreciation to Wang Anshi, the tremendous ancient reformer, economist, politician, and scholar. It is his experience and spirits that encourage me to continue when I feel depressed, lost, and gloomy.

Contents

1 Introduction

A virtual assistant, or chatbot, is a dialogue system designed to understand human languages in different modalities such as textual or audio mode and to respond by stimulating conversations with human users [1]. Recently, this topic has attracted continuous interest from researchers and professionals. Chatbots bring benefits to both service providers and customers in e-commerce, education, information querying, and some other fields [2]. For companies, chatbots may help reduce customer service costs and improve the ability to deal with multiple requests simultaneously. From the perspective of users, they may choose chatbots for various reasons, including productivity, social functionality, novelty, etc. [3].

Chatbots have become omnipresent thanks to the prevalence of AI-based Virtual Assistants such as Siri by Apple, Cortana by Microsoft, Alexa by Amazon, and some other famous brands [4, 5]. However, those assistants are always platform-based, implicitly restraining the accessibility of cross-platform queries and services. For example, Siri is only accessible on iOS or Mac devices but not Android systems. An objective of this project is to break the device constraint via a cross-platform product.

Another existing problem in the development of chatbots lies in the complexity of model architectures. Depending on different tasks, the structure of a chatbot may vary accordingly, but there are some mutual components shared by most chatbot systems. Figure 1.1 (modified from [3]) shows a general structure of a chatbot. Detailed information for each component is introduced in section 2.2.

The structure reflects the intricacy involved in processing dialogues as human beings, which is the exact objective a dialogue system is supposed to realize. Input utterances are first decomposed by NLU to remove ambiguity, only after which a reasonable action decision could be made by DM, leading to a relevant information search by IR and the final output generated by RG.

While the complexity of structure allows chatbots to handle comprehensive tasks, the time and effort required to develop a chatbot increased drastically as a tradeoff. For example, deep neural networks, a common measure to implement NLU, DM,

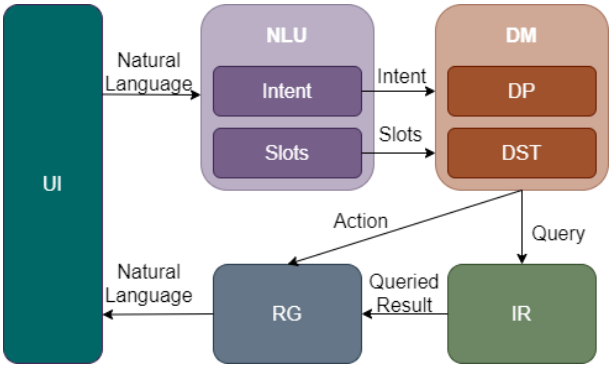

Figure 1.1: A general architecture of a chatbot (modified from [3])

and RG units, are constructed through copious data, substantive training iterations, and repeated hyperparameter tuning. Moreover, after the completion of all elements, integration may be exhausting since each model is trained separately. Therefore, further fine-tuning is usually necessary for model integration, which is also time- and effort-consuming.

With the continuous shifting trend from offline to online teaching mode due to the influence of the pandemic and the development of techniques on virtual communication, the necessity of a virtual assistant for education-related application scenarios is gradually being recognized. Unfortunately, the blank of relevant datasets hinders the realization of such products. This project attempts to generate a relevant education-focused dataset for Natural Language Understanding and explore a novel approach that teaches a chatbot based on the NLU unit to reduce the cost of development and provide services for educational purposes across different platforms.

2 Literature Review

This section first depicts a sketchy portrait of chatbots, and then focuses on the Natural Language Understanding (NLU) task description and model implementation. Section 2.1 classifies chatbots based on multiple criteria. Section 2.2 talks about the major components contained in a chatbot. In section 2.3, we dive deeper into NLU, the element where our chatbot prototype development starts. Lastly, section 2.4 discusses the datasets that are commonly used in NLU model development.

2.1 Classification of Chatbots

Chatbots can be divided into diverse classes according to different criteria. A clear description of a chatbot's knowledge domain, service, and goal will simplify the data collection and model design. With the scope narrowing down through the definition, topics of demanded data tend to cluster, and the omission of some components in the general structure may be possible. Table 2.1, restructured from Adamopoulou and Moussiades [3], describes the genres of chatbots.

This project focuses on creating an open domain goal-oriented intrapersonal chatbot equipped with NLU models to accomplish a range of tasks given by the user.

2.2 General Chatbot Architecture

As shown in Figure 1.1, a general chatbot architecture may consist of the following modules: User Interface (UI), Natural Language Understanding (NLU) unit, Dialogue Management (DM) unit, Information Retrieval (IR) unit, and Respond Generation (RG) unit. The following subsections give more details on the components' functionalities, except for 2.2.6, which justifies the feasibility of an NLU-based chatbot as proposed in the Introduction section.

2.2.1 User Interface (UI)

UI obtains user input and displays the output generated by the chatbot. The variation of application scenarios put forward a new challenge for the user interface

Criteria	Classification
Knowledge Domain	**Open domain**: able to respond to general topics. **Close domain**: focus on a particular domain and may fail to respond to other questions.
Service	**Interpersonal**: not companions of the user but provide information on user's requests. **Intrapersonal**: exist within personal domain of the user. Most virtual assistants belong to this genre.
Task	**Informative**: provide user with information in a fixed scope. **Conversational**: chat with user as if it is a human being. **Goal-Oriented**: perform a specific task such as reservation.

Table 2.1: Chatbot classification and corresponding criteria (adapted from [3])

to be cross-platform. For instance, PC users access the service via web pages and the Internet, while mobile devices users are more likely to expect an Android or iOS application.

2.2.2 Natural Language Understanding (NLU) Unit

NLU unit is responsible for the understanding of the natural language collected from the user. The purpose of NLU is to interpret given texts in a way that humans would understand it [6]. NLU typically consists of two subtasks, namely intent classification and slot filling, together with attempting to form a semantic parse for utterances. Intent classification tries to find out the query's intent, while slot filling focuses on the semantic information of the user's request [7].

As the first step where a chatbot starts to process input data, the NLU unit is intuitively a reasonable starting point to build a chatbot. In section 2.2.6, after introducing all the other units, we discuss the feasibility of building a dialogue system model based on NLU.

2.2.3 Dialogue Management (DM) Unit

DM unit receives intents and semantic meanings from the NLU unit and decides which action the chatbot should take. This process is known as Dialogue Policy

(DP). Meanwhile, Dialogue State Tracking (DST) is critical in multi-round conversations, which tracks the dialogue state from the beginning until the most recent input. A chatbot can therefore utilize the contexts to better understand the current utterance [8].

2.2.4 Information Retrieval (IR) Unit

While the DM unit decides the action, IR searches for appropriate information that matches the action [3]. For an open domain environment, the information is usually retrieved through the Internet or common knowledge, while the source becomes a database in a close domain.

2.2.5 Respond Generation (RG) Unit

Upon retrieval, a natural language response is generated by the RG unit and displayed to the user via UI. There exist three major alternatives for the implementation: rule-based, retrieval-based, and generative models [3].

A rule-based or retrieval-based strategy chooses outputs among a predefined finite set of answers according to the recognition of inputs. The difference is that retrieval-based methods offer more flexibility by querying possible response candidates through Application Programming Interfaces (APIs) [3]. As a result, the answers may be rigid and sometimes mismatch the intentions of users. A generative model generates the answer text through analysis of the conversation context and retrieved information. While the generation brings diversity and flexibility in responses, as a trade-off, the costs of time and resources may become an obstacle to development.

2.2.6 Feasibility of an NLU-based Chatbot

As mentioned in section 2.2.2, this section elaborates on the feasibility of a goal-oriented dialogue system based on NLU.

In terms of the feedback generation process, a typical response to a task is an action and a display of the result. Hence, a rule or retrieval-based solution is able

to cover most scenarios and is a more compact solution in a goal-oriented scope.

NLU identifies the intent and semantic information of input texts. The former describes the action assigned to the chatbot, while the latter is adequate for information retrieval in our application scenario. Given a task, DM decides the action based on the intent, while IR fetches the information that helps to realize the mission. Therefore, the functionalities of DM and IR could be combined into NLU and RG for a goal-oriented dialogue system.

After the removal of DM and IR, a more concise chatbot structure consists of only an NLU unit for input, a rule-based RG for output, and the UI for display. To sum up, under the scope of a goal-oriented open domain chatbot, the general architecture could be reduced to an NLU core with UI and rule-based RG.

2.3 NLU Techniques

This section mainly concentrates on mathematical modeling, model architectures, and training details of NLU. In 2.3.1, the mathematical representation of NLU subtasks is discussed. Section 2.3.2 reviews an eminent pre-trained model BERT, followed by an introduction and justification on training implementations in the pipeline (section 2.3.3) and joint methods (section 2.3.4).

2.3.1 NLU Mathematical Modelling

As mentioned in section 2.2.2, the NLU task can be further divided into an intent classification task and a slot sequence classification task.

Intent Classification Given an utterance $\boldsymbol{x}$ and a set of intent classes of size M, the mathematical interpretation of expected intent classification target $\hat{c}_i$ based on the content of $\boldsymbol{x}$ is [9]

$$\hat{c}_i = \text{argmax}_{i \in M}\, p(c_i \mid \boldsymbol{x}).$$

As the expression is similar to general classification tasks, standard classifiers are suitable for this task after appropriate conversion from texts to numeric values [9].

Slot Filling Described as a sequence classification task, slot filling aims at finding the most probable slot assignment $\hat{y}$ of the utterance $\boldsymbol{x}$. Denote $\mathcal{Y}(\boldsymbol{x})$ as a set of all the possible slot assignments of $\boldsymbol{x}$, the relationship between $\hat{y}$ and $\boldsymbol{x}$ can be described by the equation [9]:

$$\hat{y} = \text{argmax}_{y \in \mathcal{Y}(\boldsymbol{x})} \, p(y \mid \boldsymbol{x}).$$

Every slot assignment y of $\boldsymbol{x}$ has two critical dimensions contributing to $p(y \mid \boldsymbol{x})$ given a finite set of slot tags, namely the partition of $\boldsymbol{x}$ as well as the the slot tag assignment of each partitioned chunk. As a result, the search space $\mathcal{Y}(\boldsymbol{x})$ could be tricky for some standard classifiers [9].

In order to solve this issue more efficiently, the Conditional Random Fields (CRF) technique is introduced [10] and widely applied in slot filling tasks [7, 11, 12].

Conditional Random Fields (CRF) CRF requires a set of pre-defined feature functions [10]. A feature function could describe a state feature (denoted by s) or a transition feature (denoted by t) [10]. Both of them stem from a matching function b, where

$$b(\boldsymbol{x}, i) = \begin{cases} 1, & \text{if the } i\text{-th occurrence of } \boldsymbol{x} \text{ matches the condition of } b; \\ 0, & \text{otherwise.} \end{cases}$$

For instance, the condition in b could be: the designated word is *medical* .

The definition of s and t are given by

$$s(y_i, \boldsymbol{x}, i) = \begin{cases} b(\boldsymbol{x}, i), \text{ if } y_i \text{ satisfies certain condition;} \\ 0, \text{ otherwise.} \end{cases}$$

and

$$t(y_{i-1}, y_i, \boldsymbol{x}, i) = \begin{cases} b(\boldsymbol{x}, i), \text{ if } y_{i-1} \textit{ and } y_i \text{ satisfies certain condition;} \\ 0, \text{ otherwise.} \end{cases}$$

where y_i is the slot labels at position i.

For each position, these feature functions are evaluated and sum up. A weighted sum with normalization is then calculated as the CRF probability of $\boldsymbol{x}$. CRF learns the weights λ_k by maximizing the weighted sum. The following equations describe this procedure mathematically [9].

$$p(y \mid \boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} exp(\sum_{c \in C} \sum_{k} \lambda_k f_k(y_c, \boldsymbol{x}, c)),$$

$$Z(\boldsymbol{x}) = \sum_{y} exp(\sum_{c \in C} \sum_{k} \lambda_k f_k(y_c, \boldsymbol{x}, c)),$$

where c denotes the word cliques of $\boldsymbol{x}$ and f_k stands for the k-th feature function.

2.3.2 Bidirectional Encoder Representation Transformer (BERT)

BERT, short for Bidirectional Encoder Representation Transformer, has gained its empirically robust reputation through its impressive performances in various NLP tasks ever since it is introduced by [13].

BERT obtains the name from its structure. Prior to BERT, [14] apply the attention mechanism [15] to connect the encoder and decoder in natural language representation and put forward the Transformer architecture (Figure 2.1). Transformers are stacked and linked to form a network in BERT bidirectionally (Figure 2.2), which outperforms previous left-to-right language models like ELMo or OpenAI GPT Vaswani et al. [14].

BERT achieves new state-of-art performances in GLUE, MultiNLI, and SQuAD [13]. Besides, its potential value in NLP is continuously dug up by new variants [16, 17, 18] and applications [7]. ALBERT is a lite version of BERT that per-

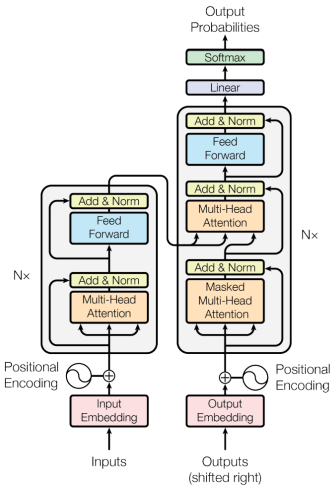

Figure 2.1: Transformer Architecture [14]

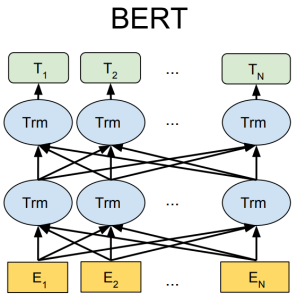

Figure 2.2: BERT Architecture [13]

forms outstandingly in a wide range of NLP tasks [19]. DistilBERT is a shrunk BERT model with a 40% smaller size and 97% retained capability of language understanding [18]. RoBERTa is an improved BERT with key hyperparameters fine-tuned and larger training data set applied. Its performance exceeds that of BERT in some critical NLP datasets such as GLUE, RACE, and SQuAD [16]. DeBERTa is an enhanced architecture of BERT and RoBERTa, and it also achieves outstanding performances in some mainstream NLP tasks [17].

2.3.3 Traditional Pipeline NLU Models

Having discussed the mathematical theories behind subtasks of NLU in section 2.3.1, this section combines the two in a pipeline fashion.

Traditional NLU development usually trains a Slot Filling model (SL) first, and then uses SL's prediction as a feature fed into the training of an Intent Classification model (IC) (Figure 2.3). However, this approach ignores the correlation of results in SL and IC, which means the training of SL cannot benefit from IC's prediction [12].

In order to solve the problem of insufficient information utilization, several joint

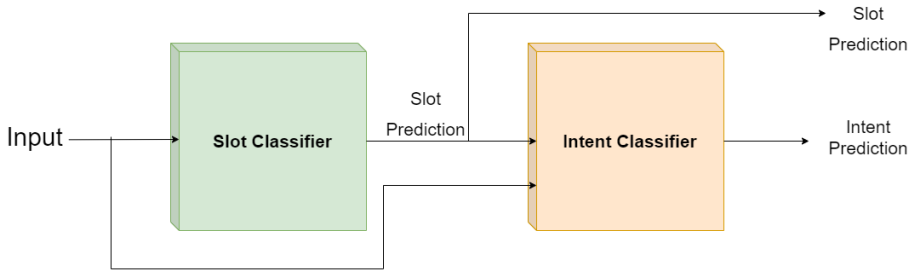

Figure 2.3: A Pipeline NLU Model (based on [12])

measures are explored.

2.3.4 Joint NLU Approaches

In this section, we discuss the joint approaches to NLU training. Divided by the stage where the combination occurs, three types of joint training are reviewed, each with one concrete example.

Joint in Mathematical Modelling This method attempts to modified the mathematical representation mentioned in 2.3.1 and make connections between the two targets. Among works in this field, Jeong and Lee [12] 's triangular CRF achieves outstanding performance compared to contemporary models.

Triangular CRF includes the intent classification into the original CRF structure by adding an intent-related factor $\phi(z, \boldsymbol{x})$ [12]:

$$p(y \mid \boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} exp(\sum_{c \in C} \sum_{k} \lambda_k f_k(y_c, \boldsymbol{x}, c) + \lambda_\phi \phi(z, \boldsymbol{x})).$$

Jeong and Lee [12] examined several variations to derive $\phi(z, \boldsymbol{x})$. For example, one variant calculates the joint probability of slot labels and the intent label. Though the triangular models outperform their concurrent counterpart, however, the new structure also results in more computations. Hence, triangular CRF may suffer from high computational costs when confronting a large-scale dataset.

Joint in Model Functionality Instead of modifying mathematical representation, another joint approach uses one model to reach two goals. Guo et al. [9] 's experiments of NLU joint training using Recursive Neural Networks (RecNNs) follow the exact idea.

RecNNs are capable of handling tree-structured inputs by recursively applying the same network at each tree node to generate the final output. Guo et al. [9] assume that each utterance $\boldsymbol{x}$ reflects its own constituency parse tree, with each word forming a node of the tree. RecNNs are first applied to each word to generate several semantic phrases in the second last level of the parse tree, and then recursively upwards until a final output is derived. This output vector is fed into a semantic classifier for joint intent and slot label decoding. The whole network is optimized through backpropagation to maximize the accuracy of the classifier.

The implementation of RecNNs is less complex compared to triangular CRF. Nevertheless, the joint performance does not show significant improvements in contrast to its separate counterparts, which may indicate that the joint benefit is not utilized thoroughly.

Joint in Loss Function Combining the two targets in the loss function may be a better way that concentrates the advantages of the two aforementioned methods. Chen et al. [7] 's work shows not only structural flexibility but also performance improvement.

Chen et al. [7] adopt pre-trained BERT to process the input text. The final layer and the hidden layers of BERT outputs are then sent to the intent classifier and slot classifier correspondingly, and the prediction losses are calculated separately. Finally, the total loss of the model is the weighted average of the two losses, and backpropagation is conducted to realize the combination of the two subtasks of NLU.

Despite the impressive conciseness and high accuracy, this method has realized, Chen et al. [7] 's work is not sufficient to use directly in our project. On the one hand, the dataset in their work does not match our project objective of an education-focused domain. On the other hand, their work does not discuss the significance

of some hyperparameters, which are covered in our experiments. More particulars about the dataset construction and experiments are displayed in section 3.

2.4 NLU Datasets

Two datasets that are applicable for the training of the joint approach are discussed in this section. Both are used in Chen et al. [7]'s work.

2.4.1 ATIS

One of the most popular datasets for spoken language understanding is the Airline Travel Information Systems (ATIS), which is extensively utilized in previous works in NLU [7, 9, 12, 20, 21]. ATIS provides transferred manuscripts from flight information inquiries collected from an automatic flight system. There are 17 intent classes, each associated with unique slot templates. The whole dataset, including training, development, and test sets, consists of more than 5,000 utterances.

2.4.2 SNIPS

Though ATIS provides sufficient data for NLU training, the intent labels cluster closely into the airline-related domain. On the contrary, containing more than 16,000 data samples, the SNIPS dataset consists of seven intent varying from weather queries to music playing. Compared with ATIS, SNIPS has a larger scale and more varied topics. However, this dataset is not sufficient for our project, either, as its topics are more related to common daily intents rather than education.

3 Methodology

This section contains the NLU model structure description, experiments and analysis on the model, the design and implementation of the custom education-related NLU dataset, and the fine-tuning process in our custom dataset. In order to avoid redundancy, the Software Design section (section 4) will not get into details of these particulars.

3.1 NLU Model Design

As mentioned in section 2.3.2, our model design maintains the skeleton of the loss function joint approach [7], while editing the pre-trained component and weighting scheme. Figure 3.1 illustrates the structure of our NLU model, which is drawn according to Chen et al. [7]'s code implementation.

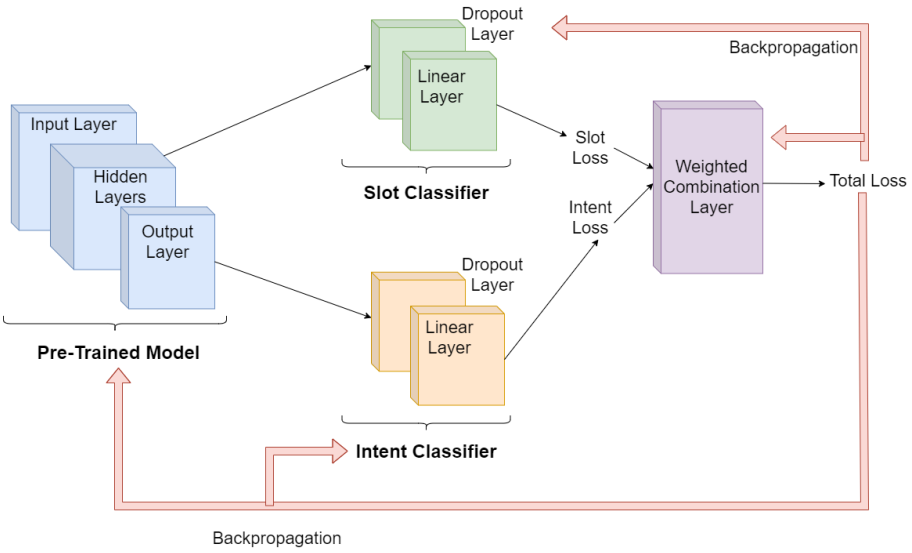

Figure 3.1: Structure of NLU Model (based on Chen et al. [7]'s code)

3.2 Pre-Trained Model Details and Hyperparameters

This section compares the pre-trained models and displays the hyperparameters used in the experiments. Table 3.1 shows some critical information about each

Name	Author(s)	Version	Size (MB)	SQuAD 1.1 F1 (dev/train)
bert	Devlin et al. [13]	bert-base-uncased	420	- / 88.5
albert	Lan et al. [19]	albert-xxlarge-v1	851	94.6 / 89.1
disdilbert	Sanh et al. [18]	distilbert-base-uncased	256	86.9 / 88.5
roberta	Liu et al. [16]	roberta-base	478	91.5 / 84.6
deberta	He et al. [17]	microsoft/deberta-base	533	93.1 / 87.2

Table 3.1: Pre-trained model details

learning rate	5e-5	**optimizer**	AdamW
epsilon	1e-8	**intent loss**	cross entropy loss
dropout rate	0.1	**slot loss**	cross entropy loss
slot loss coef	1.0	**slot loss with crf**	negative log-likelihood

Table 3.2: Hyperparameter values

pre-trained model according to the documentation provided by Huggingface.

Table 3.2 is a description of the hyperparameter values in the experiments. For the purpose of pre-trained models' comparison, these values are kept the same across all the different experiments.

3.3 Experiment Design and Implementation

This section discusses the design of model experiments, examining the effect of BERT variant alternations, the number of training epochs, the adoption of CRF, and the value of slot loss coefficient.

Different pre-trained models are tested to figure out the one that fits the joint NLU task best. Due to pre-training costs and integration issues, we mainly examine pre-trained models provided by huggingface, a popular NLP python library. Apart from BERT, several other pre-trained models are tested for the joint NLU task, including ALBERT, DistilBERT, RoBERTa, and DeBERTa. Each model is trained on ATIS and SNIPS datasets separately with the same set of hyperparameters. Each model is trained twice, one trial with CRF and one without it. The evaluations are recorded on the train and dev set after the training for one epoch. After the whole training process, the joint models are then evaluated on the test set. From the data

Dataset	Train	Dev	Test	Epoches
ATIS	4478	500	893	7
SNIPS	13084	700	700	20

Table 3.3: Dataset sizes and training epoches

recorded in the training process, we can also derive the effect of different numbers of epochs.

The influence of slot loss weight is examined in another set of experiments. Pre-trained models and epochs are kept the same, while the slot loss coefficient is changed to observe the resulting performance. Charts are sketched to compare the result clearly.

Table 3.3 describes the size of the subset and the maximum training epochs in the experiments for the two datasets.

3.4 Evaluation Matrix

The evaluation matrix of the joint intent classification and slot filling task consists of three components, intent accuracy (Intent Acc), slot F1 score (Slot F1), and semantic frame accuracy (Frame). Intent accuracy reflects the performance for intent classification. Slot F1 score is an illustration of the model's capability in the slot filling subtask. The semantic frame accuracy is a score that takes both subtasks into consideration, representing the proposition of samples whose prediction perfectly matches their truth intent labels and slot label sequences.

3.5 Experiment Results and Analysis

In this section, the results of the experiments are analyzed. The first section examines the capability of different pre-trained models via an evaluation matrix that consists of three indexes. The second section demonstrates the model performances along the training epochs. In the third section, the relationship between the adoption of CRF techniques and model performances is analyzed. Finally, the last section talks about the influence of the slot loss coefficient.

Name	Intent Acc	Slot F1	Frame	Name	Intent Acc	Slot F1	Frame
bert	0.982	0.9798	**0.926**	bert+crf	0.978	**0.9825**	0.924
albert	0.98	0.9697	0.9	albert+crf	0.982	0.9721	0.87
disdilbert	0.982	0.9781	0.924	disdilbert+crf	0.98	0.9787	0.912
roberta	**0.986**	0.9705	0.908	roberta+crf	0.98	0.9711	0.906
deberta	0.968	0.9239	0.768	deberta+crf	0.966	0.9306	0.782

Table 3.4: Evaluation matrix on ATIS dev set

Name	Intent Acc	Slot F1	Frame	Name	Intent Acc	Slot F1	Frame
bert [7]	0.975	0.961	0.882	bert+crf [7]	0.979	0.960	0.886
bert	**0.9787**	0.9566	**0.8836**	bert+crf	0.9765	**0.9592**	0.8802
albert	0.9776	0.9512	0.8634	albert+crf	0.9731	0.9505	0.8231
disdilbert	0.9754	0.9554	0.8746	disdilbert+crf	0.9753	0.9563	0.8746
roberta	0.9720	0.9562	0.8762	roberta+crf	0.9720	0.9534	0.8712
deberta	0.9675	0.8970	0.7346	deberta+crf	0.9395	0.9106	0.7480

Table 3.5: Evaluation matrix on ATIS test set

3.5.1 Alternations of Pre-Trained Models

Tables 3.4 to 3.7 record the best dev set scores each model obtains in their training process, as well as the final test score. The first rows of Tables 3.5 and 3.7 are cited from Chen et al. [7]. For each index across different models, the highest one (in rid of Chen et al. [7]'s results) is highlighted with bold fonts. If two models' scores are identical in one column, then both cells will be bold. Comparing the first two rows of the aforementioned tables, we could only tell a slight difference, which is likely to indicate that our implementation of the experiments is reliable as a replication and extension of Chen et al. [7]'s work.

It could be observed that the highest indexes are concentrated at the rows of 'BERT' and 'BERT+crf', except that in Table 3.7, the highest intent accuracy is earned by 'RoBERTa'. Whereas, it is worth noticing that apart from the winner of the highest score, several competitive models could obtain a value that nearly reaches the top for each table.

Name	Intent Acc	Slot F1	Frame	Name	Intent Acc	Slot F1	Frame
bert	0.99	0.9071	**0.9614**	bert+crf	**0.9971**	0.9606	0.91
albert	0.99	0.9619	0.9114	albert+crf	0.9943	0.9776	0.87
disdilbert	0.9886	0.9573	0.9029	disdilbert+crf	0.9886	**0.9787**	0.912
roberta	0.9914	0.9479	0.8729	roberta+crf	0.9886	0.9296	0.8457
deberta	0.9857	0.8677	0.7357	deberta+crf	0.9871	0.9074	0.7971

Table 3.6: Evaluation matrix on SNIPS dev set

Name	Intent Acc	Slot F1	Frame	Name	Intent Acc	Slot F1	Frame
bert [7]	0.986	0.970	0.928	bert+crf [7]	0.984	0.967	0.926
bert	0.9786	**0.9638**	0.9157	bert+crf	0.9857	**0.9638**	**0.9171**
albert	0.9786	0.9094	0.7771	albert+crf	0.9843	0.9477	0.8386
disdilbert	0.9786	0.9572	0.8971	disdilbert+crf	0.9786	0.9592	0.9043
roberta	**0.9886**	0.9530	0.9029	roberta+crf	0.9857	0.9350	0.8786
deberta	0.9671	0.8488	0.72	deberta+crf	0.9657	0.9025	0.7971

Table 3.7: Evaluation matrix on SNIPS test set

3.5.2 Epochs and Model Performances

This section discusses the changes in model performances alongside the increase of epochs. Figures 3.2 to 3.9 demonstrate the evaluation index changing trends via multiple iterations of training of each model on ATIS and SNIPS train and dev set. Charts in the same picture share the same y-axis scale in order that the comparison across models becomes more manageable and efficient. Figure 3.10 and 3.11 collects the curves of different model performances on ATIS and SNIPS dev sets.

From the figures, it is observed that different models obtain similar trends. With the increase of epoch numbers, the intent accuracies, slot F1 scores, and semantic frame accuracies on the train and dev set climb up and then stabilize at a high level, with training set values a bit higher than that of the dev set. In contrast, the loss values first decrease, and then the curves become plateau at a low level with a gap between train and dev set lines. These trends indicate that in all the experiments, the model converges after several epochs. Most curves reach a satisfactory score after five epochs in the training of the ATIS dataset, while this number for SNIPS is 10.

It is also worth noticing that in Figures 3.10 and 3.11, the models converge into several groups. In figure 3.10, *DeBERTa* and *DeBERTa + CRF* stabilize in a less expected performance level compared with all the other models. In figure 3.11, more levels of plateau appear, with *BERT* and *BERT + CRF* in a leading position in all the four sub-charts. To sum up, for this specific joint training task of intent classification and slot filling, except for *DeBERTa*, all the examined models display satisfactory capability, among which *BERT* shows a dominant leading performance on both datasets.

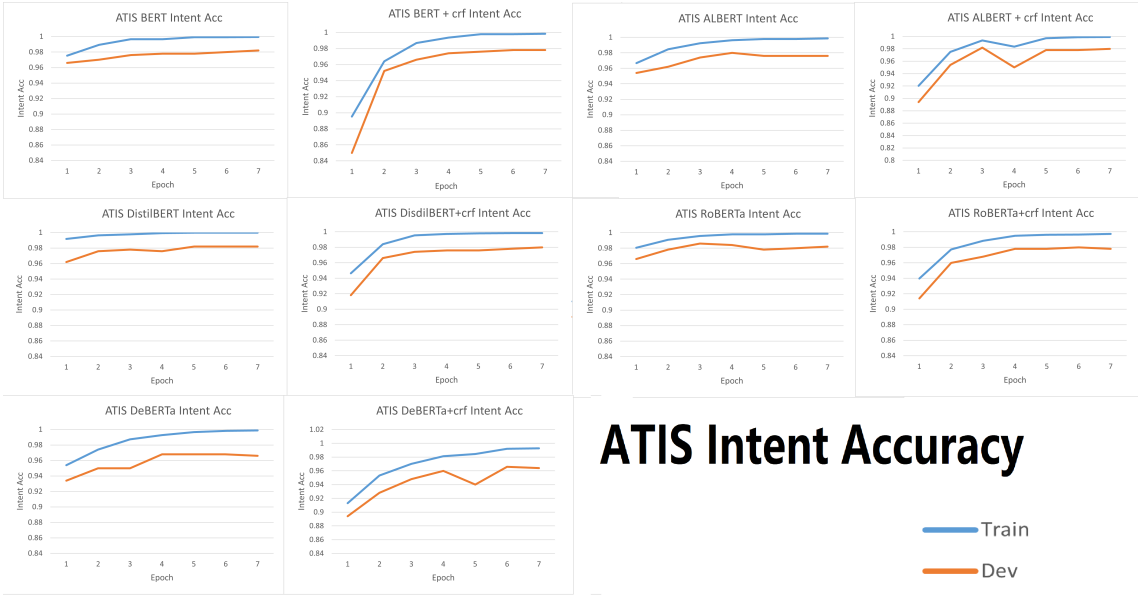

Figure 3.2: ATIS Intent Accuracy Trends along Epoch

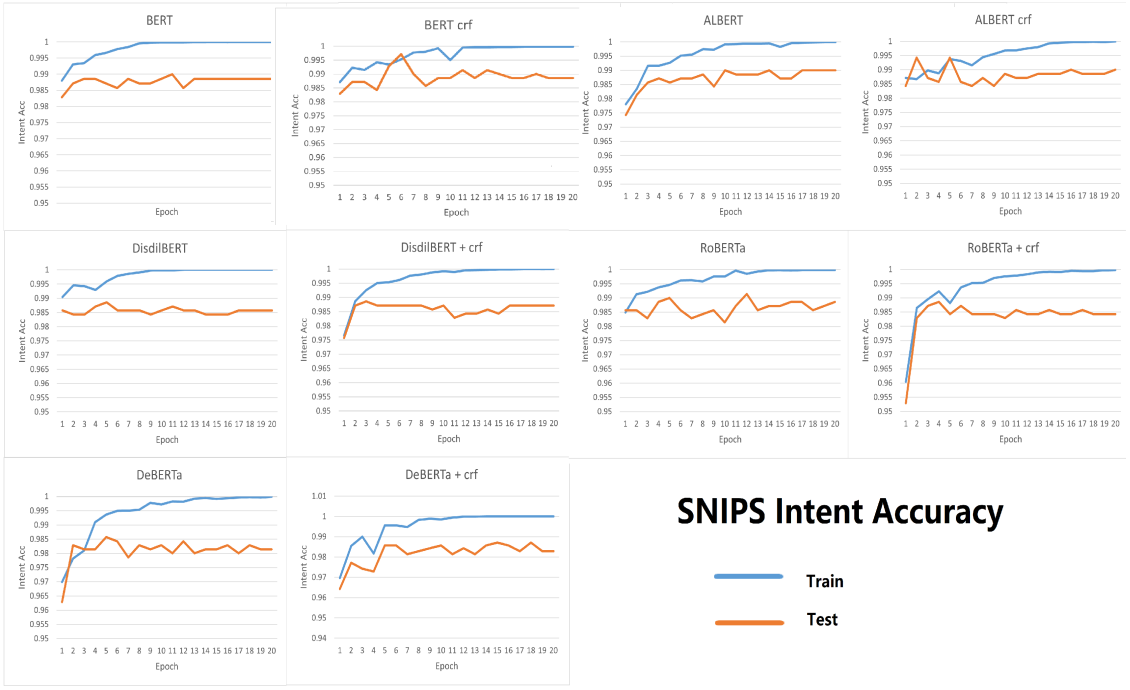

Figure 3.3: SNIPS Intent Accuracy Trends along Epoch

Figure 3.4: ATIS Loss Trends along Epoch

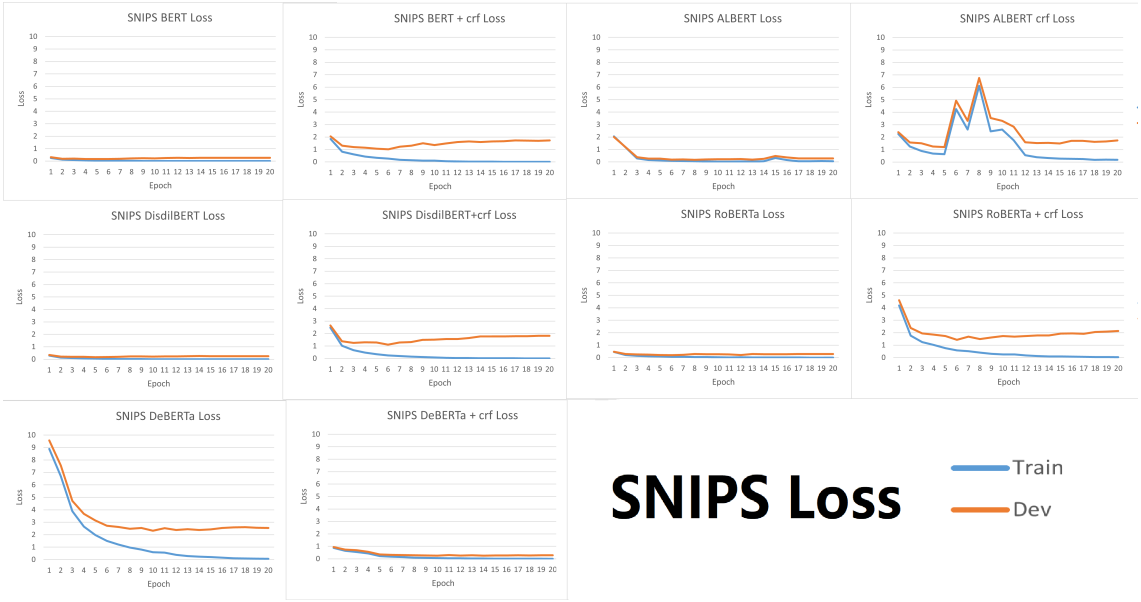

Figure 3.5: SNIPS Loss Trends along Epoch

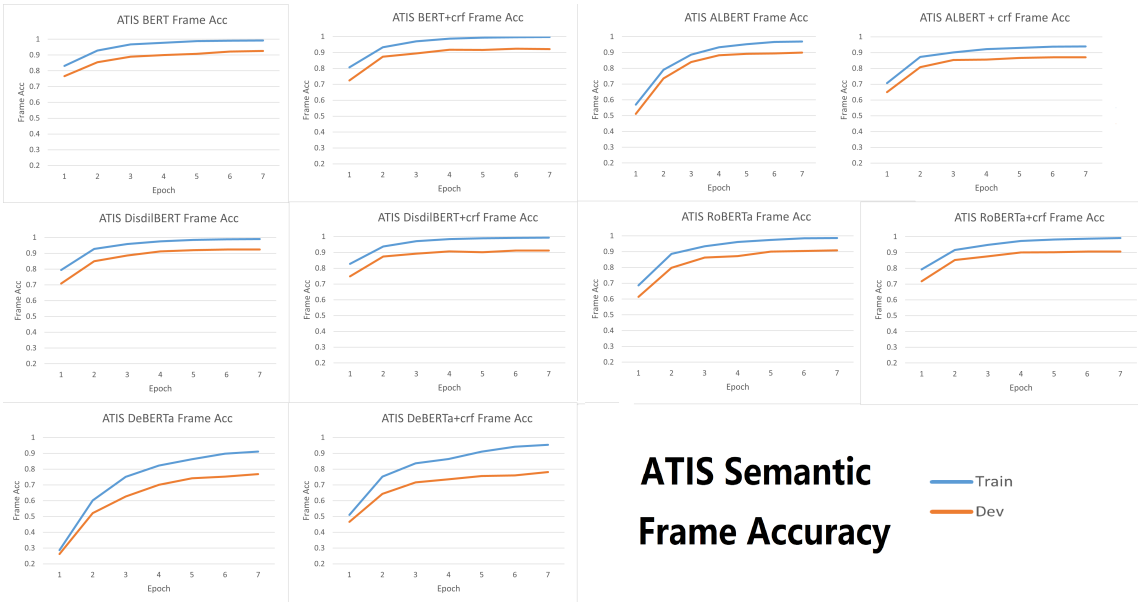

Figure 3.6: ATIS Semantic Frame Accuracy

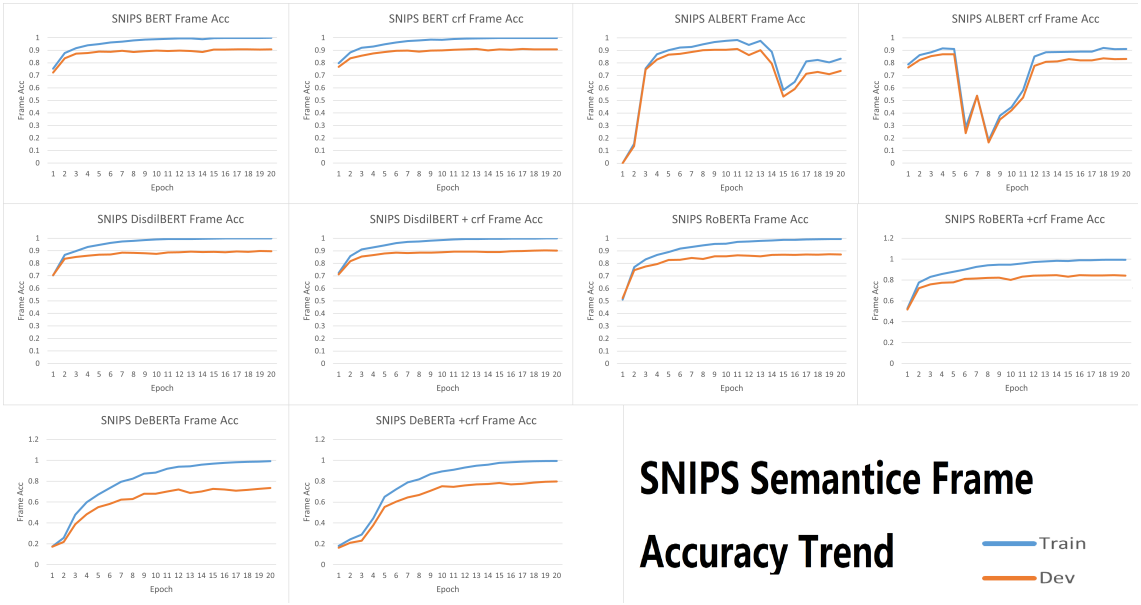

Figure 3.7: SNIPS Semantic Frame Accuracy

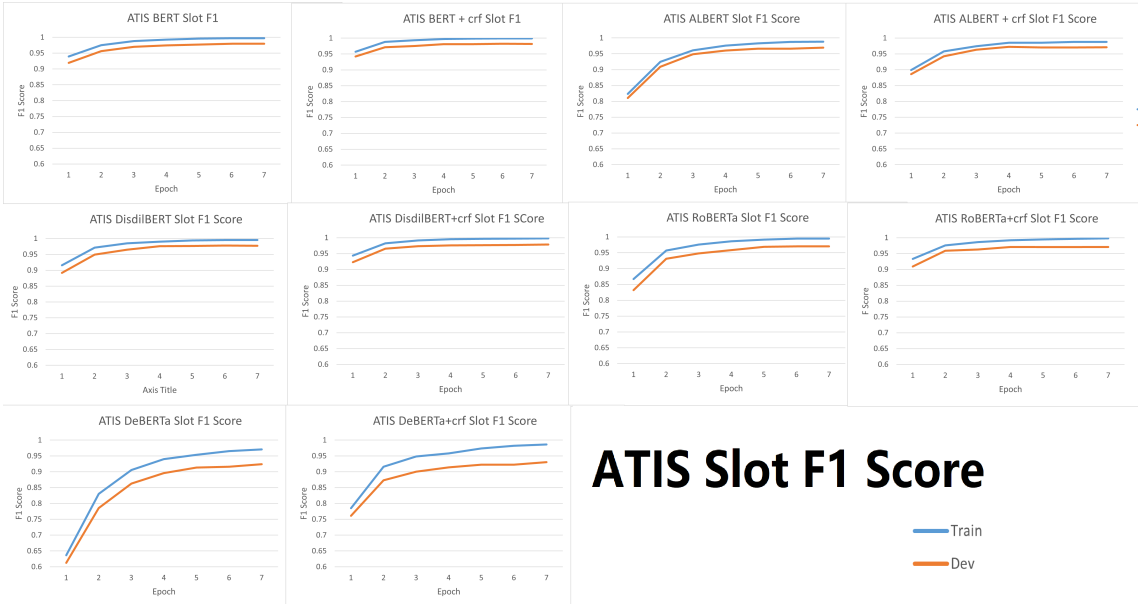

Figure 3.8: ATIS Slot F1 Score Trends along Epoch

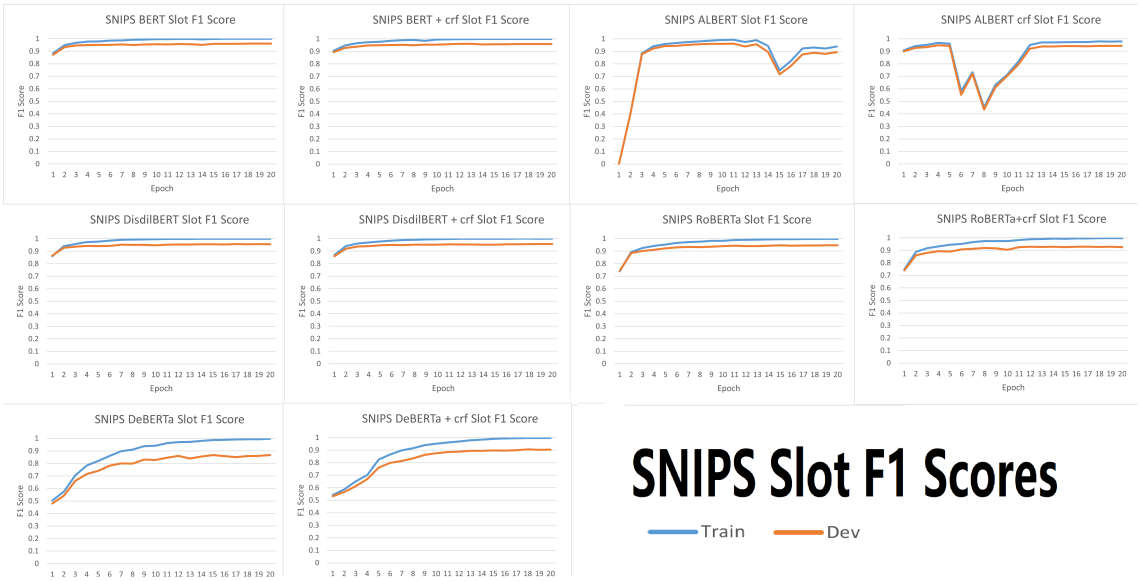

Figure 3.9: SNIPS Slot F1 Score Trends along Epoch

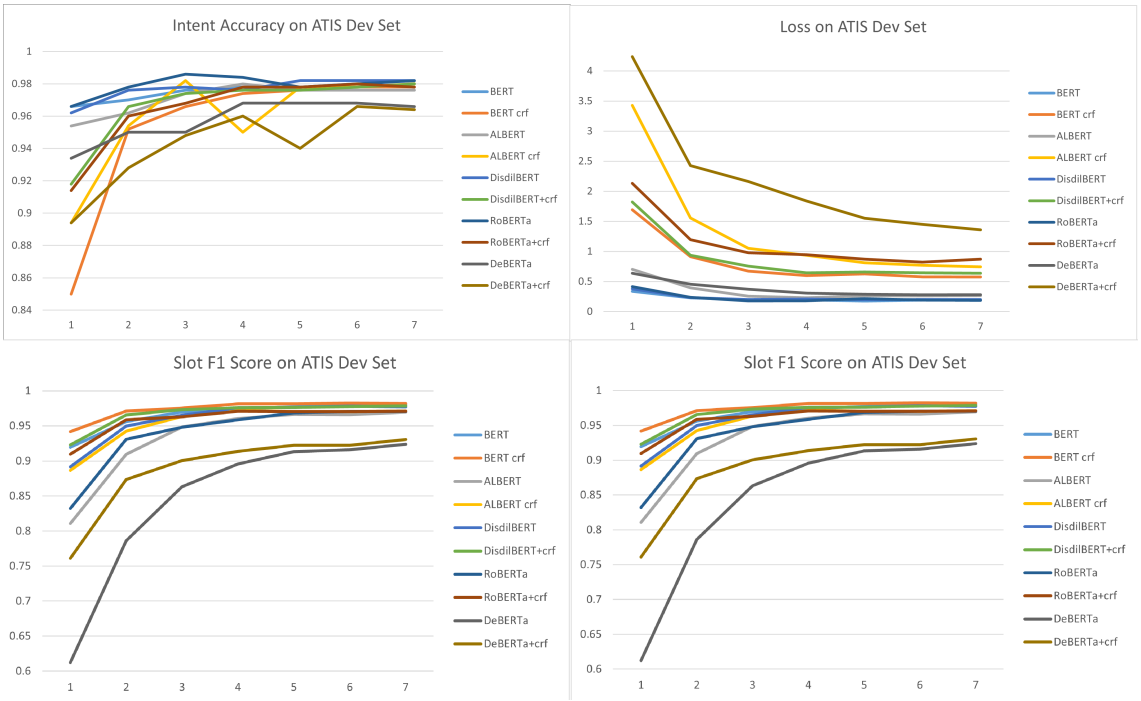

Figure 3.10: ATIS Model Comparison

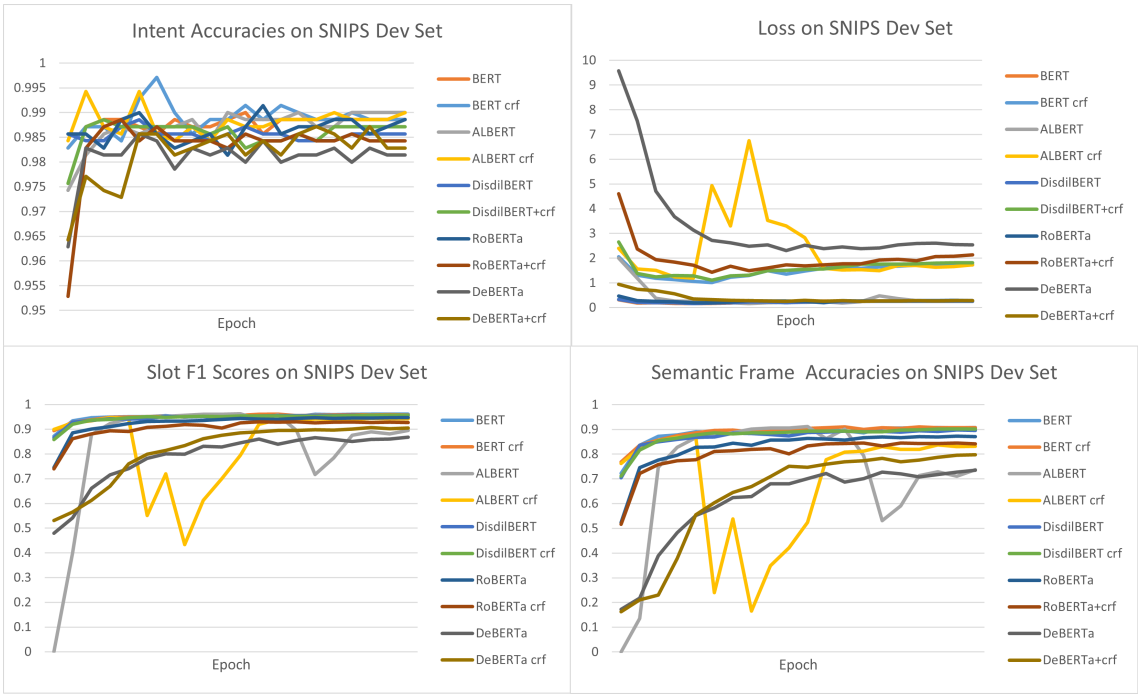

Figure 3.11: SNIPS Model Comparison

3.5.3 CRF and Model Performances

This section attempts to analyze the effect of CRF adoption on model performance. Table 3.8 calculates the increments of the evaluation scores after using CRF on each model, takes the average and size weighted average in groups of datasets, and summarizes the results. It is observed that the addition of CRF is able to improve slot filling F1 score on an average of 0.98% with a tiny degradation on intent accuracy of about 0.1% as a trade-off.

Dataset	Ave Δ Intent Acc	Ave Δ Slot F1	Ave Δ Frame
ATIS dev	-0.0024	0.0026	-0.0100
ATIS test	-0.0051	0.0021	-0.0052
SNIPS dev	0.0020	0.0224	-0.0099
SNIPS test	0.0011	0.0121	0.0202
Average	-0.0011	0.0098	-0.0012
Weighted Ave	-0.0013	0.0098	0.0009

Table 3.8: Average increment after adopting CRF

Our result on slot filling aligns with Chen et al. [7]'s conclusion on the effect of CRF. However, they do not mention the impact on the minor degradation of intent accuracy in comparison with that of rid of CRF. A preliminary intuitive hypothesis could be made to explain this phenomenon that CRF affects the total loss, namely the sum of the intent loss and slot filling loss, which in turn, through backpropagation, passes the influence to the intent classifier. In separate intent classifier training, backpropagation will only penalize when the intent prediction does not align with the ground truth, but this is not the case in our joint approach. For joint training, even though the intent is recognized precisely, backpropagation will also modify the intent-related parameters if the slot prediction is not 100% correct. This coherence could be understood as an overfitting-preventing mechanism that improves the consistency between the prediction of both tasks simultaneously. A degradation of intent accuracy and a slight rise of slot F1 score indicates that CRF helps to strengthen cohesion.

Another evidence in favor of the cohesion-strengthening characteristic of CRF is the experiment result of slot loss coefficients. Slot loss coefficient, as is discussed in section 3.5.4, controls the relevant weight of slot loss in a total loss, which in terms affects the change of parameters in the backpropagation process. Figures 3.12 to 3.17 demonstrate the changing trends of evaluation scores along with the increase of epochs. Comparing Figures 3.12 and 3.13, 3.14 and 3.15, 3.16 and 3.17, it is clear that the curves with CRF tend to be more smooth and closer to each other, which supports the hypothesis of CRF's cohesive influence. Besides, these results also indicate that the adoption assists in canceling the effects brought by the alternation of slot loss coefficient.

3.5.4 Slot Loss Coefficient and Model Performance

This section discusses the influence of the slot loss coefficient on model performance. The slot loss coefficient is a hyperparameter that determines the relative significance of slot loss in total loss. In the model implementation, we have:

$$Total\_loss = Intent\_loss + Slot\_loss\_coefficient * Slot\_loss.$$

Figures 3.18 to 3.20 display the maximum evaluation scores the model achieved

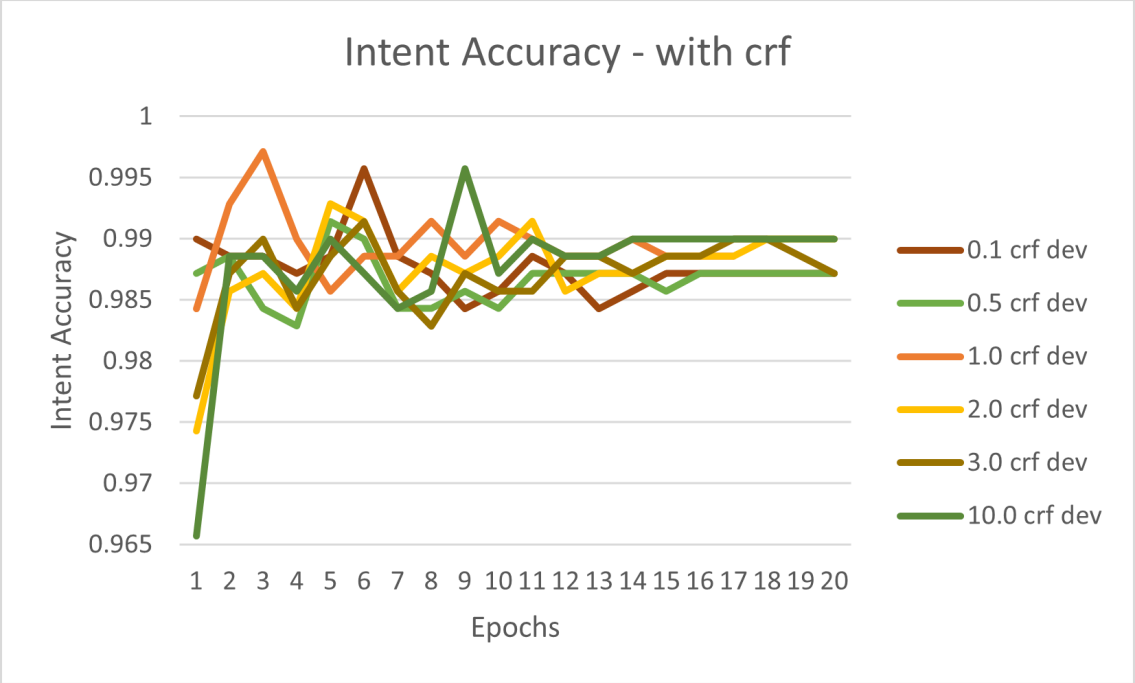

Figure 3.12: Slot Loss Coef Result: Intent Accuracy with CRF

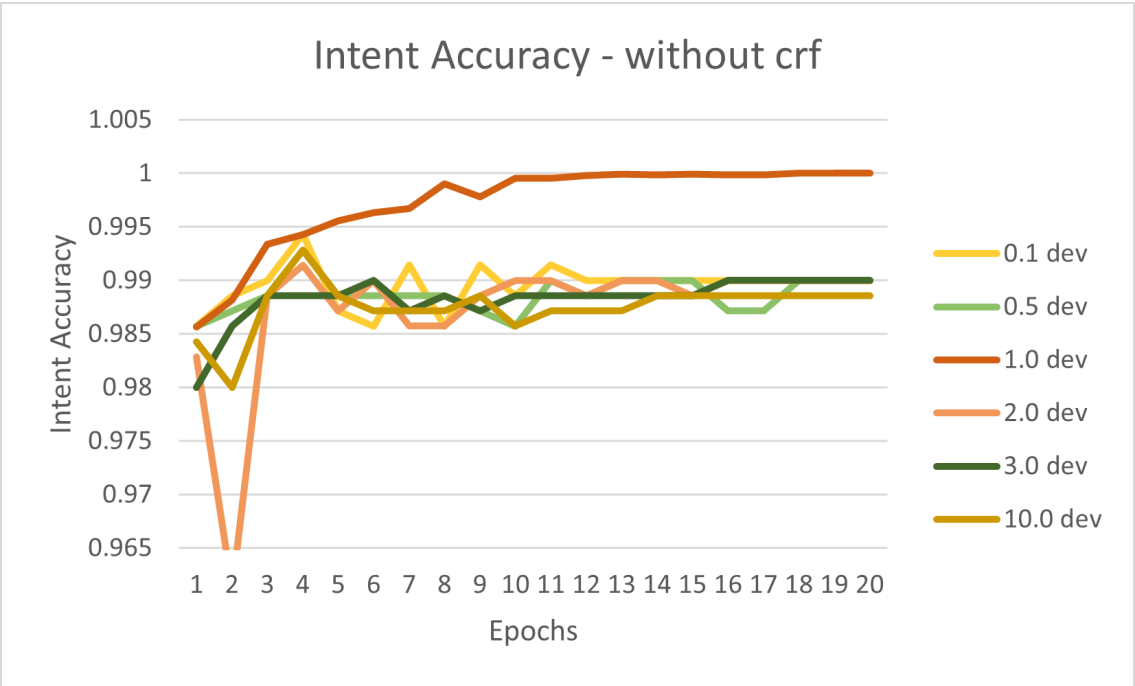

Figure 3.13: Slot Loss Coef Result: Intent Accuracy without CRF

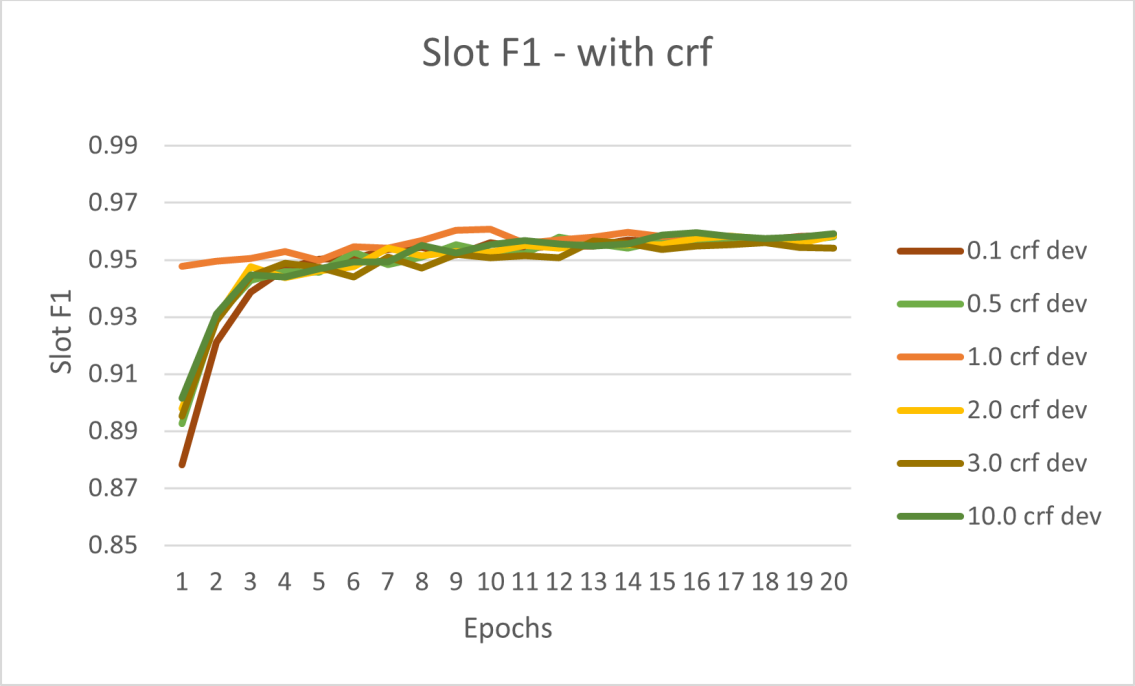

Figure 3.14: Slot Loss Coef Result: Slot F1 Score with CRF

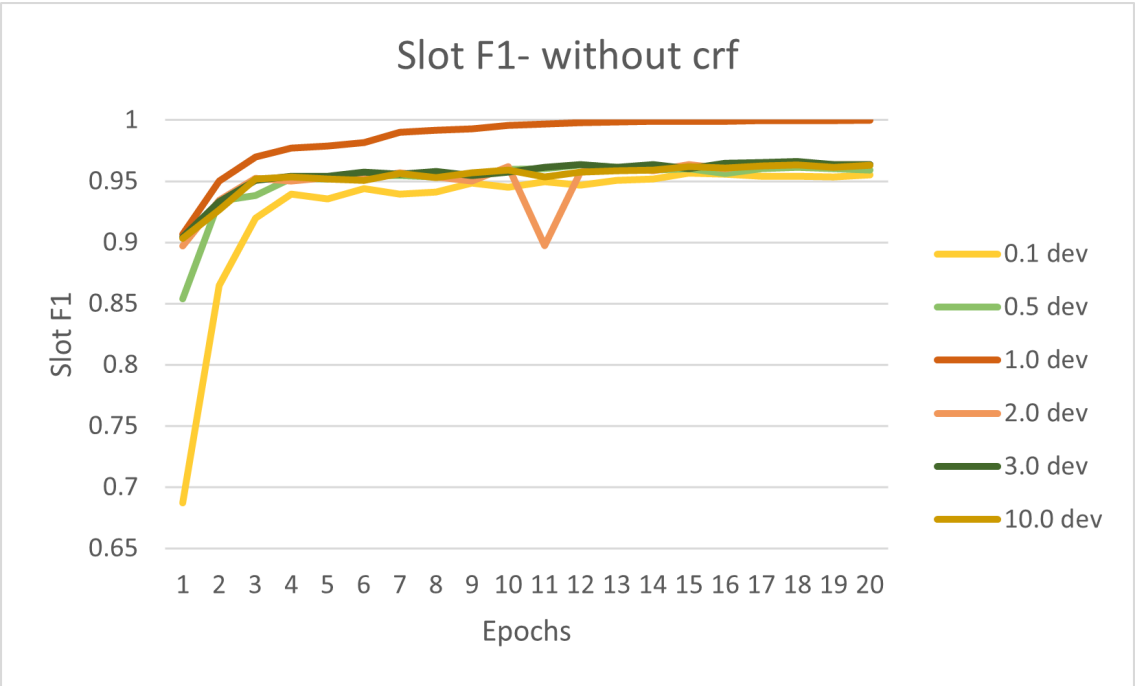

Figure 3.15: Slot Loss Coef Result: Slot F1 Score without CRF

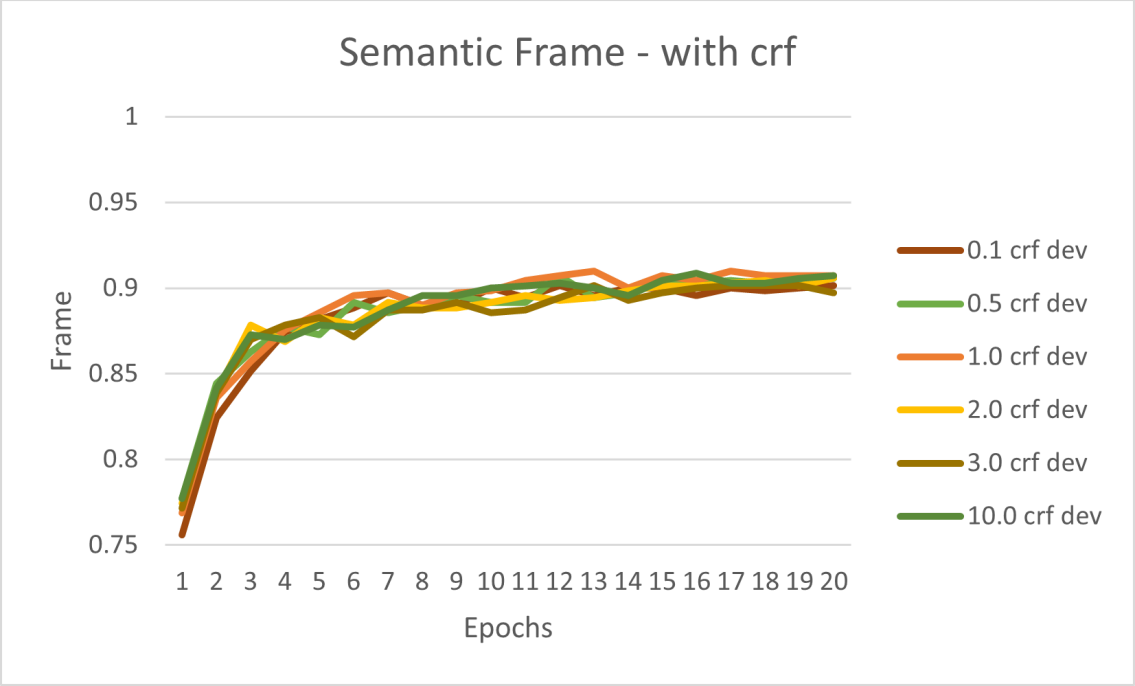

Figure 3.16: Slot Loss Coef Result: Semantic Frame Accuracy with CRF

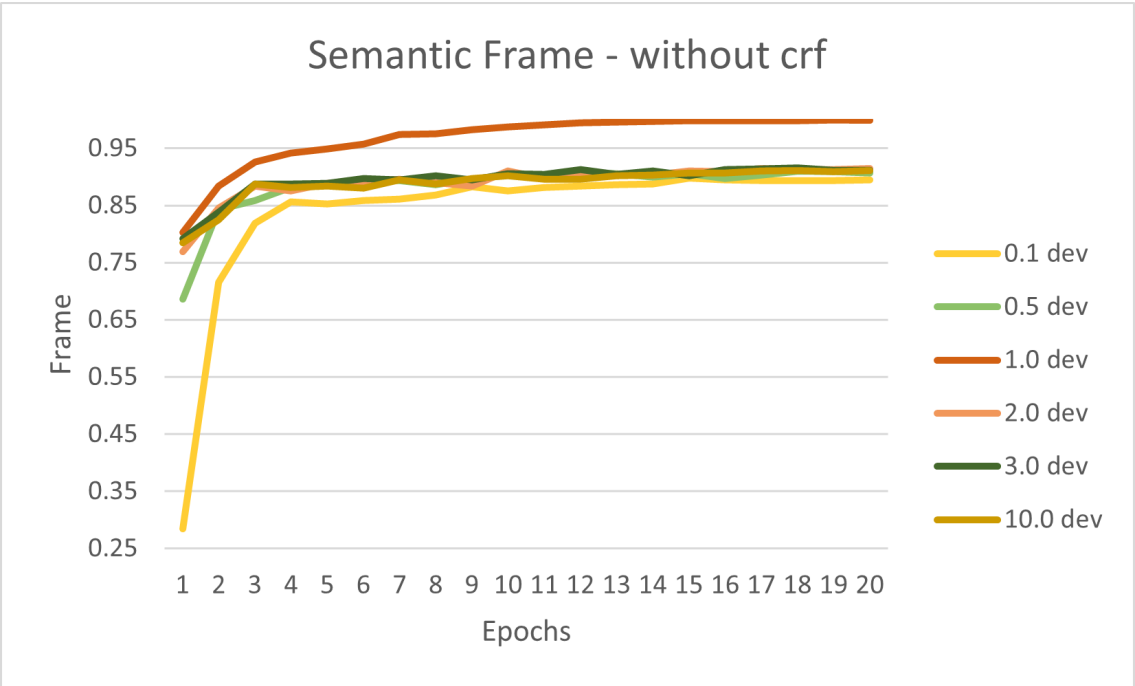

Figure 3.17: Slot Loss Coef Result: Semantic Frame Accuracy without CRF

with different slot loss coefficients. It is noticed that the differences caused by the alternation of clot loss coefficient in the model's best performance are not significant. Actually, the influence of the slot loss coefficient is reflected more in the training process. Comparing Figures 3.13, 3.15, and 3.17, we could find that the value of this coefficient affects the required training epochs to reach a relatively stable performance. Among all the options we examined, 1.0 outperforms its competitors in all three main indices, including intent accuracy, slot F1 score, and semantic frame accuracy.

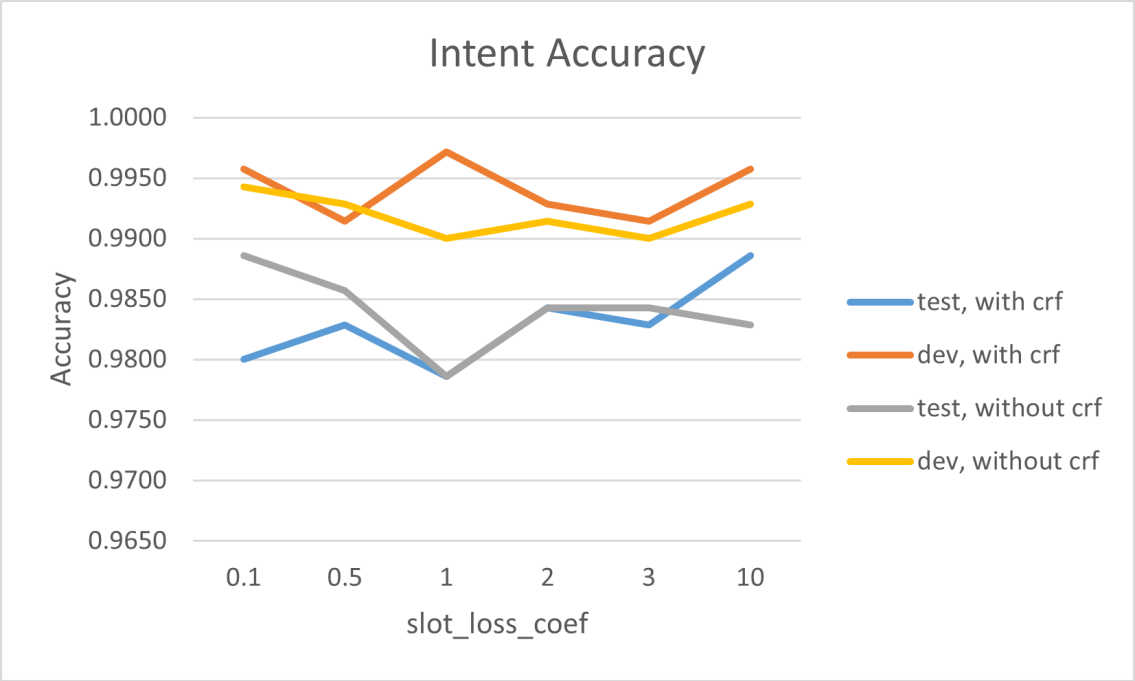

Figure 3.18: Slot Loss Coef Result: Maximum Intent Accuracy

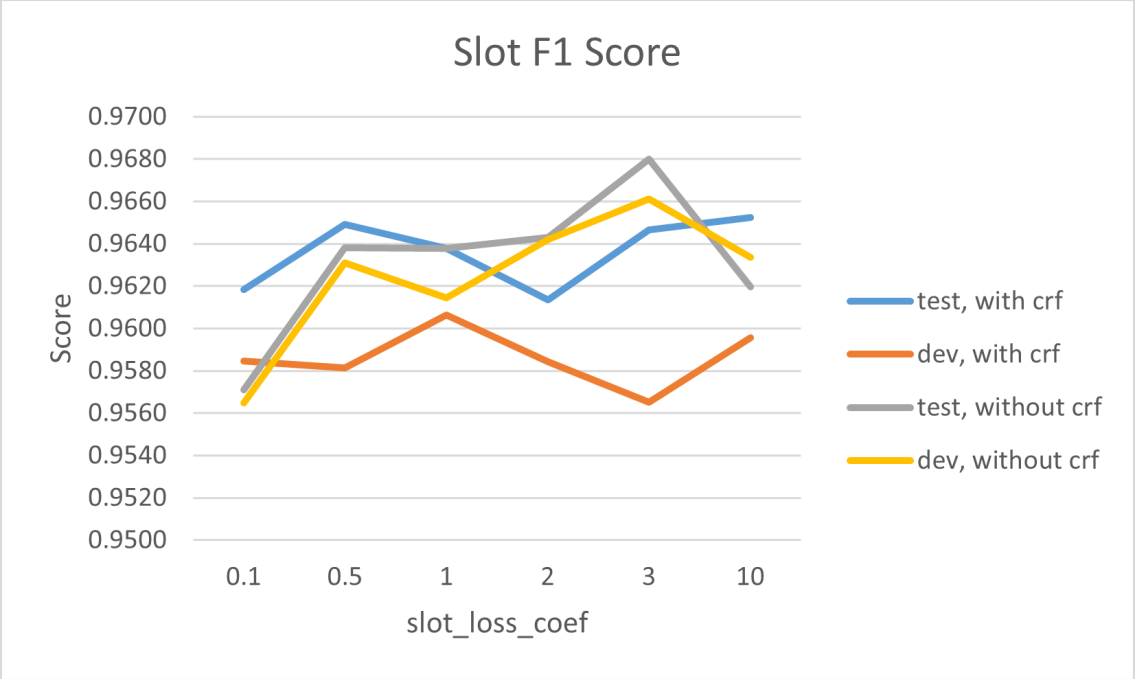

Figure 3.19: Slot Loss Coef Result: Maximum Slot F1 Score

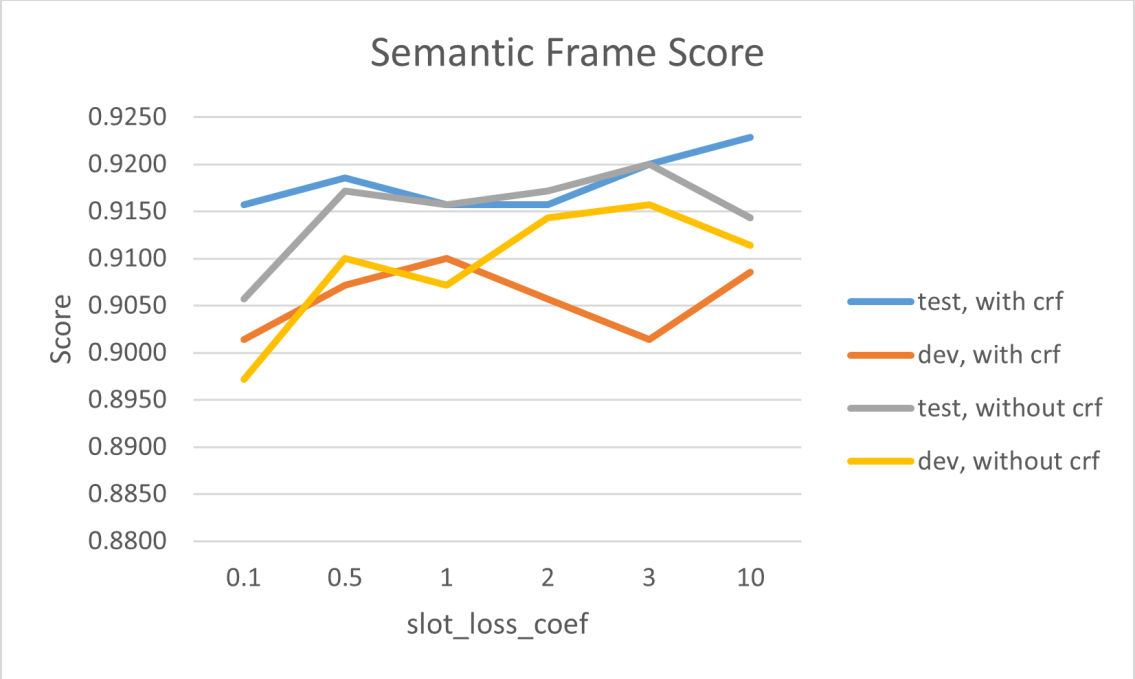

Figure 3.20: Slot Loss Coef Result: Maximum Semantic Frame Accuracy

3.6 Custom Dataset Design

As our project focuses on a specific application scenario that assists students in entertaining their class experience, it is necessary to collect data related to this domain for model training. Current intent and slot filling datasets such as ATIS and SNIPS do not express an intensive flavor on education-related tasks. Hence, we generate our own education-specific goal-oriented NLU dataset. In order to facilitate the data collection procedure, several rules that describe the specific criteria for data samples, dataset construction, and data topic domains are summarized according to the task description of NLU and the characteristics in the two aforementioned datasets.

3.6.1 Data Sample

Each data sample consists of the sentence string, the intent label, and the slot label sequence. Table 3.9 is an example of a qualified data sample.

3.6.2 Sentence String

A sentence string is an original text in natural language that reflects the speaker's intent and some critical and concrete information related to the intent. Here are some typical format requirements:

- The media is purely English.
- Punctuation marks are omitted, including comma (,), period (.), apostrophe (‘), exclamation (!), single quotes (‘ and ’), double quotes (“ and ”), question (?), semi-colon (;).

Sentence String	please let me know the weather forecast of stanislaus national forest far in nine months
Intent Label	GetWeather
Slot Label	O O O O O O O O B-geographic_poi I-geographic_poi I-geographic_poi B-spatial_relation B-timeRange I-timeRange I-timeRange

Table 3.9: Example of a qualified data sample

- All letters are converted into small letters.
- Words are separated by space.

3.6.3 Intent label

The intent label identifies the user's intent of a sentence string. Several sentence strings may share the same intent label. Each sentence string is matched with only one intent label. For example, the sentences in the list below share the same intent label 'PlayMusic':

- listen to westbam alumb allergic on google music
- play the song little robin redbreast
- can you play me some eighties music by adele

For the purpose of standardization, some format requirements of intent labels are expected for the custom dataset:

- Use Pascal Case to join multiple words. For instance, 'BookRoom, 'GetWeather'.
- Punctuation marks are illegal characters.
- Each label should be matched with only one intent and each intent is only represented by one label (1-to-1 matching).
- Several sentence strings may share the same intent label, but each sentence string is matched with only one intent label.
- Each intent label corresponds with a template of slot label sequence (Details please refer to Slot label sequence section).

stanislaus	forest	weather	in	nine	months
B-geographic_poi	I-geographic_poi	O	O	B-timeRange	I-timeRange

Table 3.10: Example of slot label and sentence string matching

3.6.4 Slot Label Sequence

The slot label sequence of a data sample represents the critical information necessary to accomplish the task assigned by the user. The format requirements of slot label sequence are listed below:

- The length of the sequence is equivalent to the number of words of its corresponding sentence. Table 3.10 is an illustration of this matching.
- Each intent label indicates a template of slot label sequence. The templates across different intent labels are usually different. The template does not limit the sequence order, but it regulates the possible labels that appeared in the sequence. Table 3.11 describes the rules of label construction.
- Inside a slot sequence, labels are separated by space.
- The 'XXX' section in Table 3.11 is formatted with small letter cases. If multiple words are involved, join them by '_'. For example, 'music_item' for a 2-word sub-label and 'artist' for a single-word slot label.
- The label 'I-XXX' only appears after 'B-xxx'. For example, consider a slot 'destination'. When the destination is 'London', only B-destination will appear in the sequence. Whereas, when the destination is 'Hong Kong', the subsequence [B-destination, I-destination] will appear in the slot sequence.

3.6.5 Dataset Preliminary Files

Our custom dataset is supposed to contain two preliminary files for the model to identify the intent and slot filling labels, namely Intent_label.txt and Slot_label.txt. Intent_label.txt consists of all the intent labels in a .txt file, with each line containing one label and the first line occupied by the label 'UNK'. Slot_label.txt lists all the

Label Identifier	Meaning
O	Trivial word to complete the task.
B-XXX	The beginning word of the critical information XXX. Here XXX is dependent on the intent (similar representation for the row below).
I-XXX	The word(s) that consists of the critical information XXX but is not the beginning word. I-XXX only appears after B-XXX, where XXX represents the exact slot identifier string. However, I-XXX is not necessary if this slot is matched with one word only.

Table 3.11: Slot label naming criteria

slot labels, with each line containing one label and the first three lines being 'PAD', 'UNK', and 'O', respectively.

3.6.6 Dataset Structure

The dataset should be partitioned into three folders, namely 'train', 'dev', and 'test', respectively. Each subset contains three files: seq.in for sentence strings, seq.out for slot label sequences, and label for intent labels of the corresponding sentences. Different records are separated by line breaks.

3.7 Custom Dataset Implementation

The data distribution satisfies the criteria that the portion of the same topic (intent label) in train/dev/test sets is equal, and the proportion of different intents in a dataset is equal. This constraint is to preserve a similar variance of data across different subsets so that the evaluation on dev and test sets is able to objectively reflect what the model has learned from the train set.

Based on real scenarios that often occur in an online education environment, we recognize four different tasks. In Table 3.12, we list all four topics that appear in our custom dataset and a data sample of each topic. The size and data distribution of each subset is displayed in Table 3.13.

Intent Label	Sentence String and Slot Sequence
BookRoom	help me book room b503 from 14:00 to 16:00 on this wednesday O O O O B-room_name O B-room_begin_time O B-room_end_time O B-room_date I-room_date
CheckGrade	show me my grade of cs2402 assignment 1 O O O O O B-grade_code B-grade_name I-grade_name
CourseInfo	is there any syllabus for cb2300 O O O B-course_info_target O B-course_info_code
ContactInfo	how can i contact prof chen O O O O B-teacher_name I-teacher_name

Table 3.12: Data topic examples

subset	BookRoom	CheckGrade	CourseInfo	ContactInfo	Total
train	25	25	25	25	100
dev	5	5	5	5	20
test	5	5	5	5	20

Table 3.13: Example of slot label and sentence string matching

3.8 Model Adaption on Custom Dataset

Two approaches to adapting the NLU model to the custom dataset are recognized. First, an NLU model can be trained on a larger dataset such as ATIS or SNIPS, and then fine-tuned on the custom dataset. Second, it can also be trained directly on the custom dataset, as BERT itself is a pre-trained model already. The first method is a common technique in transfer learning, which assumes the former larger dataset training and the latter smaller one share some parameters [22]. However, since the output domains of the two are not comparable in our implementation, the fine-tuning method requires some modification to the model structure. The second method, on the contrary, is straightforward and less complicated but without the warranty of satisfactory performance.

Our deployment of the NLU model is trained on the second method and fortunately shows satisfactory results. We take BERT as the pre-trained model, equip it with CRF, and train it for five epochs with a slot loss coefficient equal to 1.0. The first method, due to the time constraint of the project, is not examined, but can be a possible further direction of research.

4 Software Design, Implementation and Test

4.1 Software Design

Section 2.2.6 briefly depicts the architecture of our goal-oriented dialogue system. In this section, a more accurate picture is sketched.

Our chatbot prototype consists of four components: a joint-trained NLU model, which is based on Chen et al. [7]'s previous work, a cross-platform UI using Flutter, a rule-based RG unit, and a Record Maintenance (RM) unit for query history storage. As the model design is shown in the Methodology section, we focus more on the other three parts in the following paragraphs. Figure 4.1 is a system diagram reflecting the elements' connections and interactions. Notice that the RG unit is split into *Respond Decision* and *Respond Generation* to clarify how intent and slot information is involved in the generation of responses.

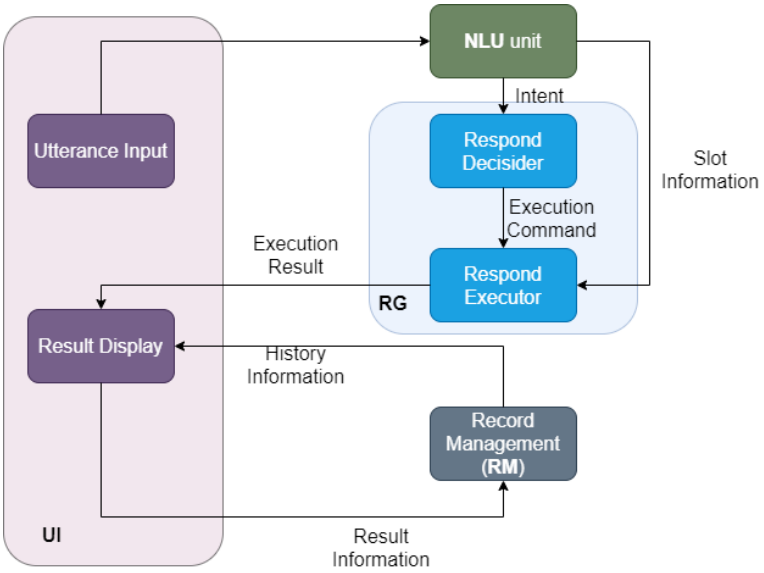

Figure 4.1: Chatbot System Diagram

4.1.1 Use Case Diagram

Though the main task of this project concentrates on goal-oriented NLU, to produce an application, some auxiliary functions are incremented to the software design

to facilitate the user, especially the record storing and error handling functions. Figure 4.2 below shows the user cases involved in the interaction.

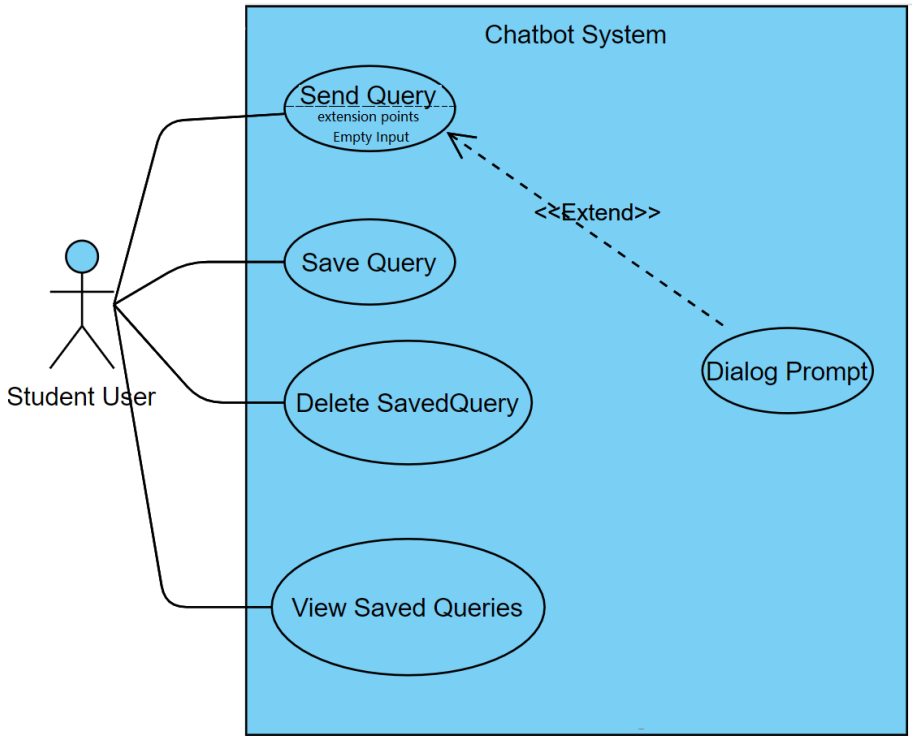

Figure 4.2: User Case Diagram of Chatbot

Case *Send Query* is the core function of the chatbot. It is responsible for receiving the user's utterance, extracting the intent and slots contained, conducting the recognized task, and finally returning the execution result. This function involves the interactions of all the components in our design. When the user attempts to send an empty input, this case is extended to case *Dialog Prompt*, which pops up a dialog box alerting for the empty input and simultaneously prevents the blank message from sending out.

The rest cases are related to records management. *Save Query* allows a chatbot response result to be stored in a checklist, followed by its mirror case *Delete Saved Query*. Case *View Saved Queries* facilitates the browsing of all saved queries.

4.1.2 Sequence Diagrams

Figure 4.3 describes the sequence of sending a message. Figure 4.4 illustrates the actions involved in saved records management. Figure 4.5 displays the work flow of the back-end system.

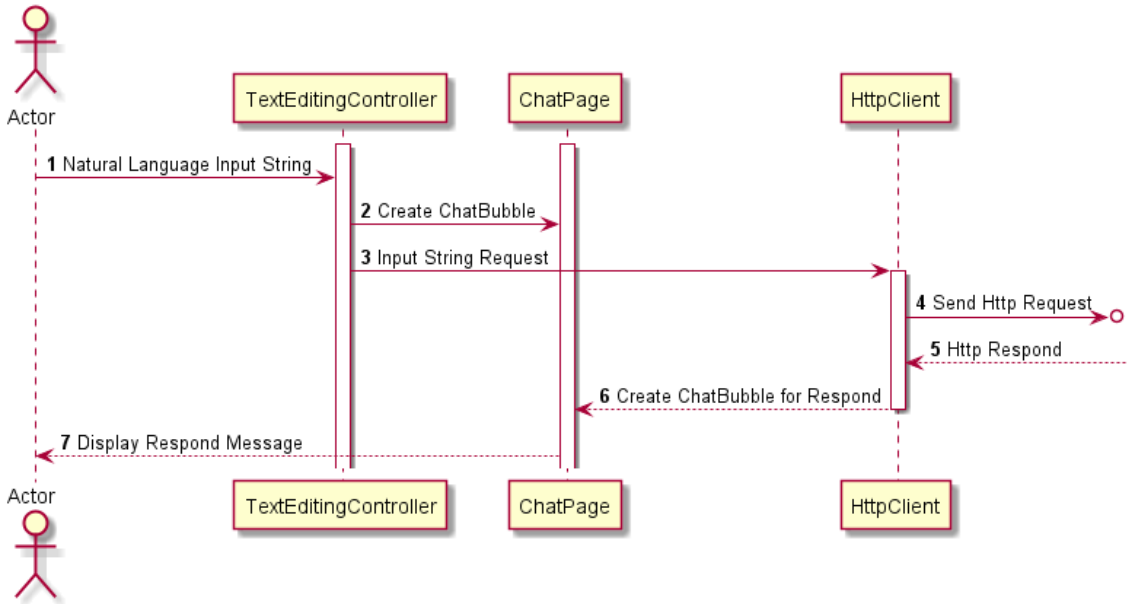

Figure 4.3: Sequence Diagram for Message Sending

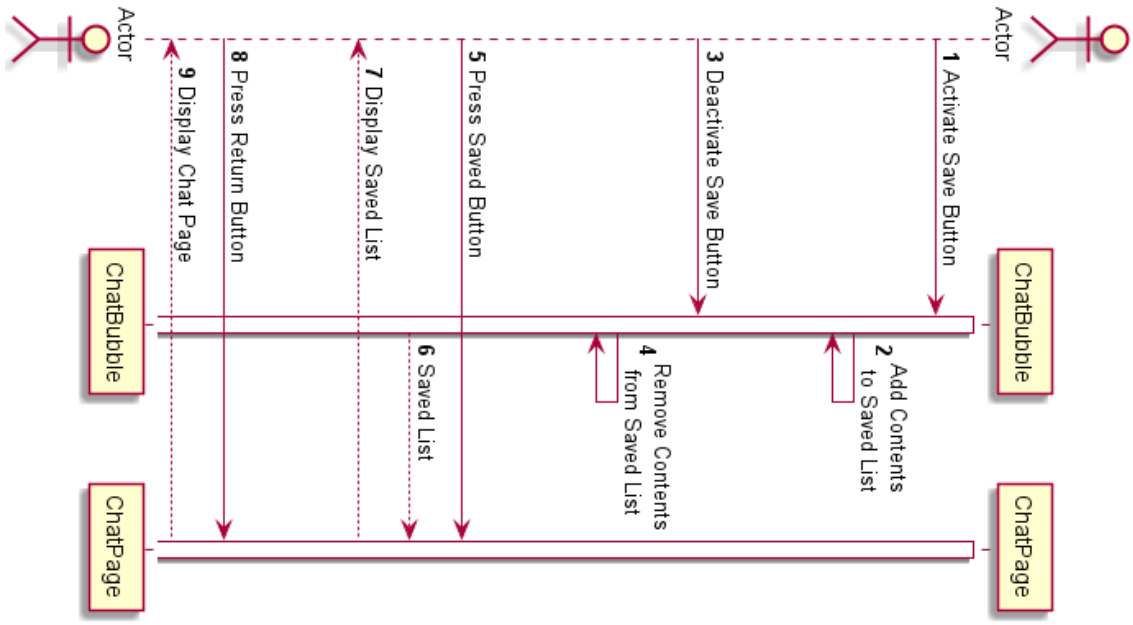

Figure 4.4: Sequence Diagram for Saved Records Management

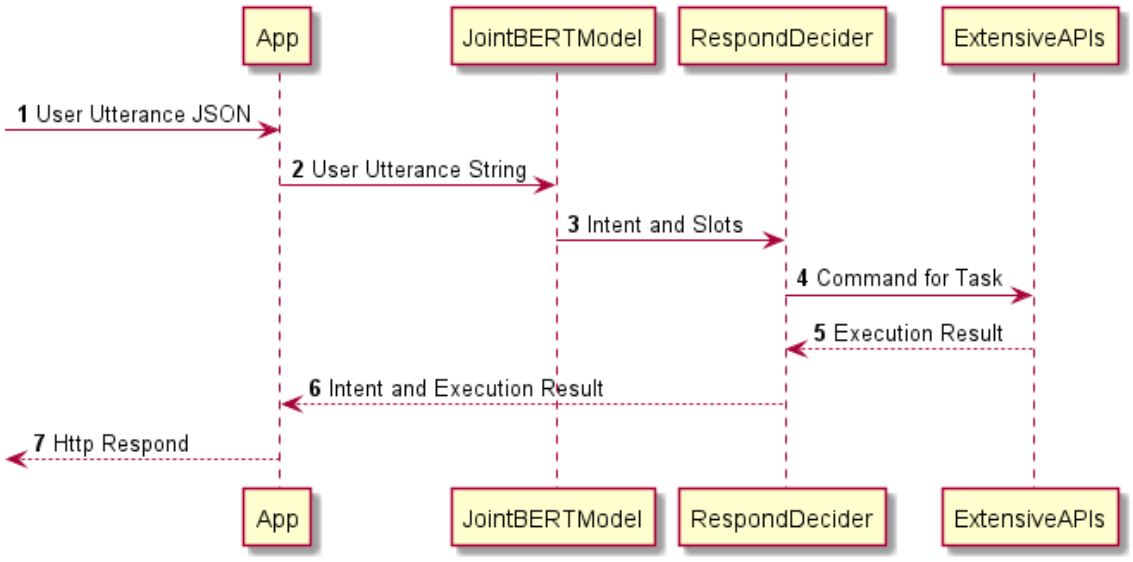

Figure 4.5: Sequence Diagram for Back-end

4.1.3 Cross-Platform UI

As described in the Introduction section, we expect to provide a cross-platform product independent of the Operating Systems (OS). However, in traditional software development, OS constraints the programming languages. For instance, many mobile applications have iOS and Android versions because of the distinct mobile device environments. Fortunately, Flutter appears with its broad support from Android, iOS to website front-end development. This project adopts Flutter to construct a cross-platform UI.

4.1.4 Rule-Based RG Unit

As shown in Figure 4.1, the rule-based RG unit receives the intent and slot information extracted from the NLU unit and conducts the assigned task. This procedure is elaborated into a three-step process. To begin with, the *Respond Generator* receives the intent and takes execution commands accordingly. Next, the command, together with the slot information, is passed to the *Respond Executor* to actually operate the task. Lastly, as the execution result is returned, it is then displayed to the user via UI. *Respond generator* is implemented in the backend system in a rule-based manner, and *Respond Executor* is designed as dummy functions that return a fixed value, which can be easily extended and integrated with different outside systems.

4.1.5 Record Management (RM) Unit

The Record Management (RM) unit is put forward to ameliorate the user experience of goal-oriented dialogue systems in a conversation-like environment. A conversation-based interface always displays dialogues in a reversed chronological fashion, which means the latest messages are shown first, and earlier ones can be accessed via up-scrolling. Nevertheless, for goal-oriented queries, history information may need to be recalled from time to time.

The RM unit is implemented with three major functionalities: add a record to a list, display a saved list, and delete a record from the list. In our project, RM

utilizes volatile local storage in users' front-end devices. Moreover, a back-end database for saved history may be a possible future work direction since it can take advantage of the cross-platform timbre of the product and thus further improve the competitiveness of this system.

4.2 Front-end Implementation

In this section, the implementation of the front-end functionalities is displayed, covering the function units of UI and RM. The class diagram is generated by an Android Studio plugin *PlantUML integration* automatically based on our developed front-end codes.

4.2.1 Class Diagram

Figure 4.6 describes the dependencies of classes in the front-end design. Packages with prefix *flutter::* is provided by Flutter, whereas those with *code::* is developed for the project. In Flutter, elements whose states may change during execution should extend class *StatefulWdiget* and implement the possible changes in its corresponding *State* classes. On the contrary, static elements should extend *StatelessWidget*.

At the bottom of the front-end structure are the chat bubbles. The abstract class *Bubble* is implemented by two concrete classes *LoadBubble* and *ChatBubble*. The former appears when the system is waiting for a response from the system, while the latter displays the chat history of the user and the system. All the bubbles are contained in *ChatPage*. The stateless *MyApp* is the entry to execute the application. On executing, it creates a window that contains a *ChatPage* object to provide interactions and services.

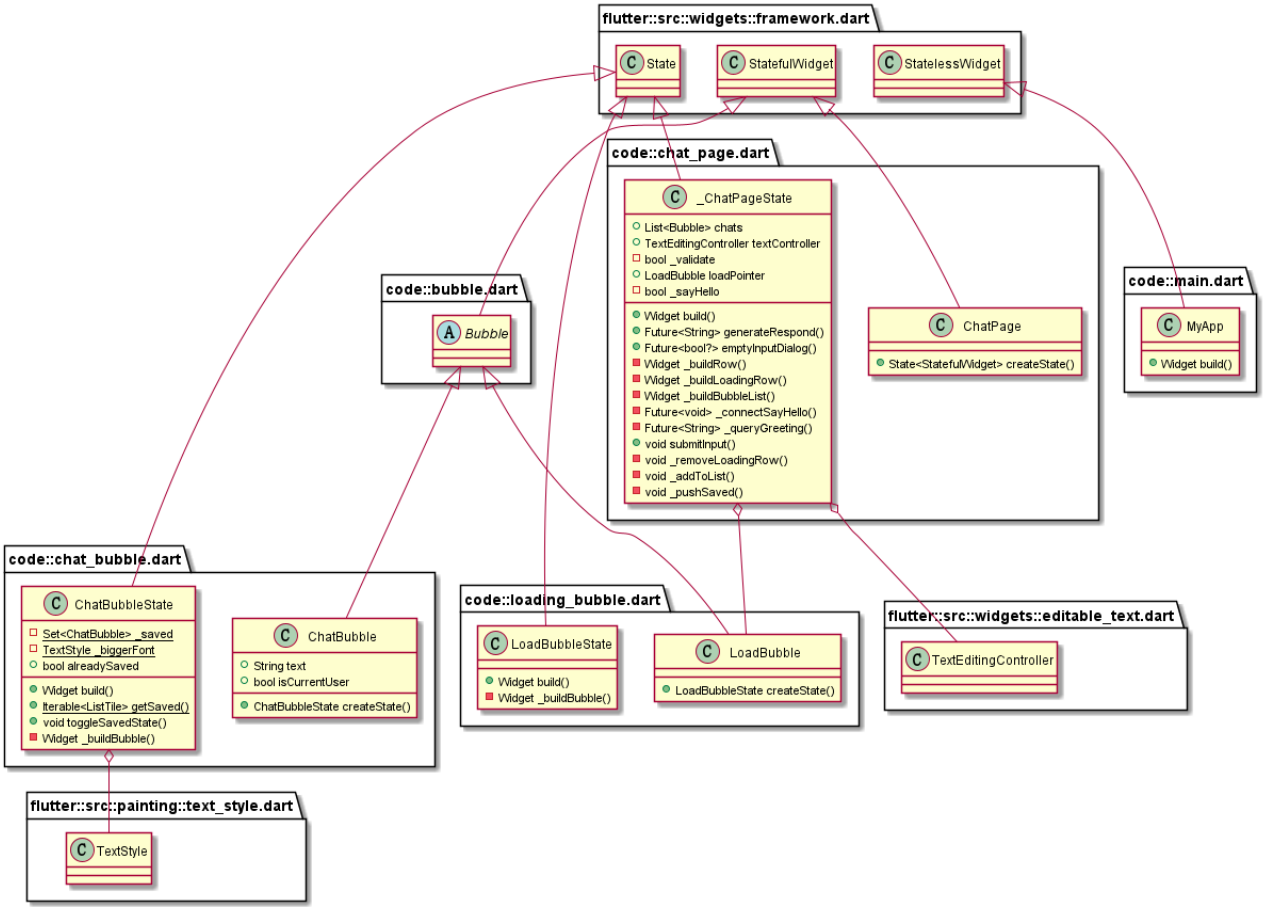

Figure 4.6: Front-end Class Diagram

4.2.2 UI Demonstration

This section demonstrates the front-end interface and functionalities. Figure 4.7 shows the appearance of the chat page and saved history page without saved records. Figure 4.8 shows the cases when two records are marked as favorites. Figure 4.9 displays the loading bubble that indicates the system is waiting for a response from the server.

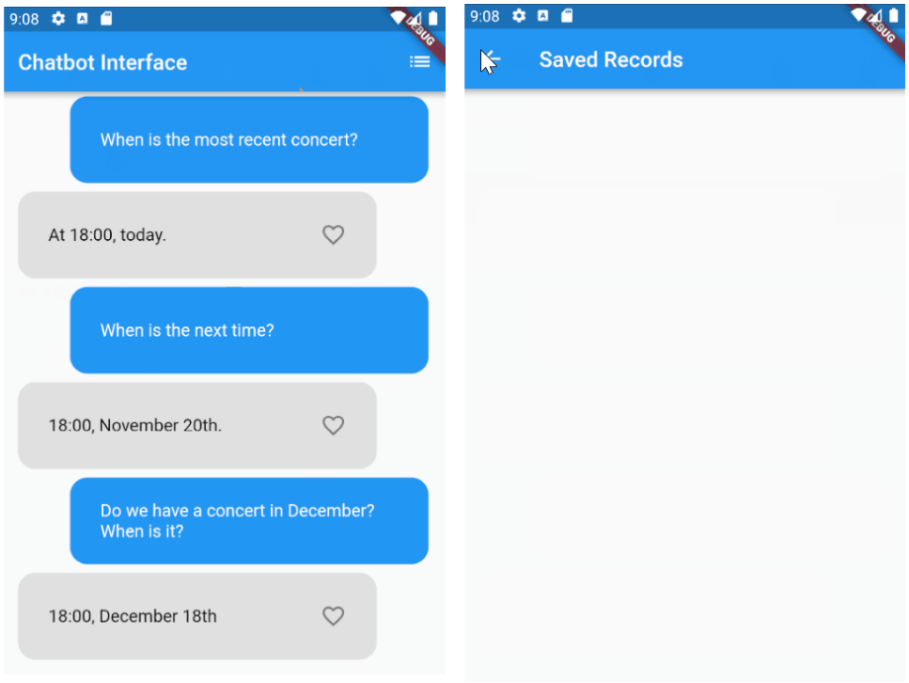

Figure 4.7: Chat page and history page without saved records

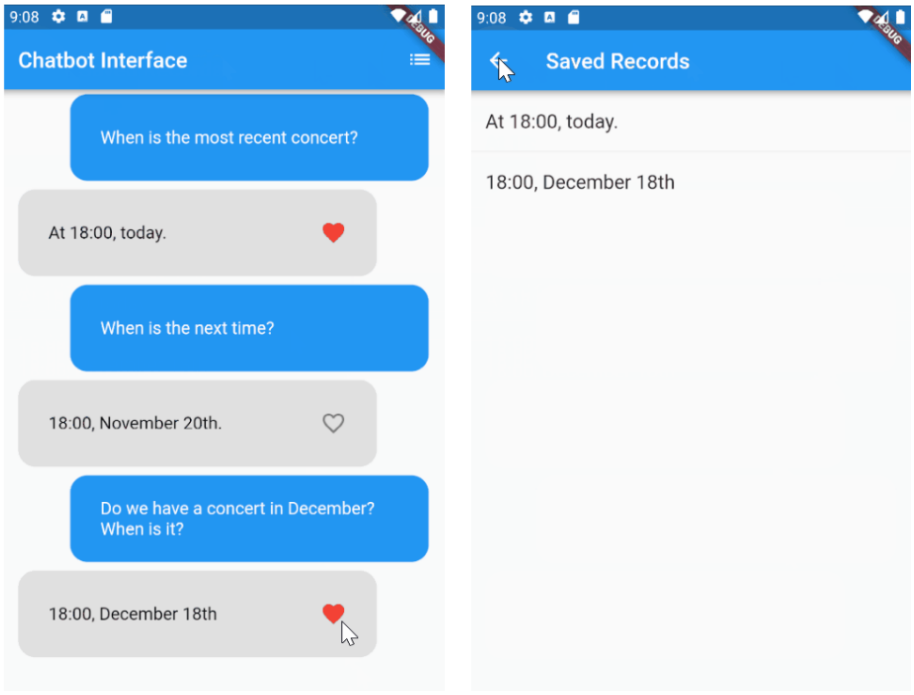

Figure 4.8: Chat page and history page with saved records

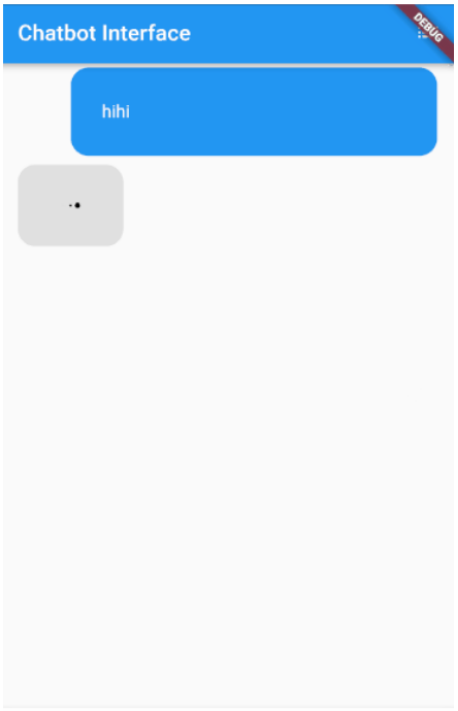

Figure 4.9: Loading bubble waiting for server respond

4.3 Back-end Implementation

This section discusses the back-end implementation. Built on *Flask*, the back-end system consists of the following parts: Application Programming Interfaces (APIs) receiving the front-end requests, an NLU predictor that extracts the meaning of the query, a respond decider that makes decisions according to the intent, and APIs for extensive systems to conduct the identified tasks.

4.3.1 APIs for Front-end

hello This API will be called when the application is wakes up. It only accepts the GET method and returns a greeting message. The purpose of this API is to check the network connection.

respond Every time the user sends a message to the chatbot, this API is called. It only accepts the POST method and returns the intent label, the slots, and the execution result of the task.

4.3.2 NLU Predictor

The predictor integrates the joint BERT model trained on the custom education-focused dataset. The code implementation inherits the main idea of Chen et al. [7]'s model prediction implementation, but with necessary adaption to the integration of other units in the system.

4.3.3 Respond Decider

The rule-based response decider takes the output from the NLU predictor. It decides task execution, which external systems will conduct via the calling of extensive APIs introduced in section 4.3.4.

4.3.4 APIs for Extensive Systems

The actual conduction of each task that matches the four predefined intends in the custom dataset requires the involvement of extensive systems. As this is not

the most critical aspect of the project, and the implementation may need further support from other platforms such as canvas, venue booking system, etc., some APIs are designed to better integrate these external systems in the future.

4.4 Test Design and Results

This section discusses the test procedure during the software development, including the front-end test for Flutter and the system test, which mainly examine the back-end and integration.

4.4.1 Flutter UI Test for Front-end

Flutter encourages a bottom-up testing strategy by offering test guidance, starting with unit tests. As an Object-Oriented Programming (OOP) language, the test cases are encapsulated into a test class. Hierarchical test case composition is also possible via the *group* function. All the test-related processes mentioned are supported by the *test* package.

Coverage tests are designed and conducted to test the front-end code implementation systematically. The result is illustrated in figure 4.10, of 100% files and 98% lines tested. The rest 2% lines that are not covered are related to network connection handling. As the Flutter test environment only supports a fake connect code 400, these lines cannot be covered under the situation. However, they are covered in the manual system test, and the test results show the network connection section works properly as our expectation.

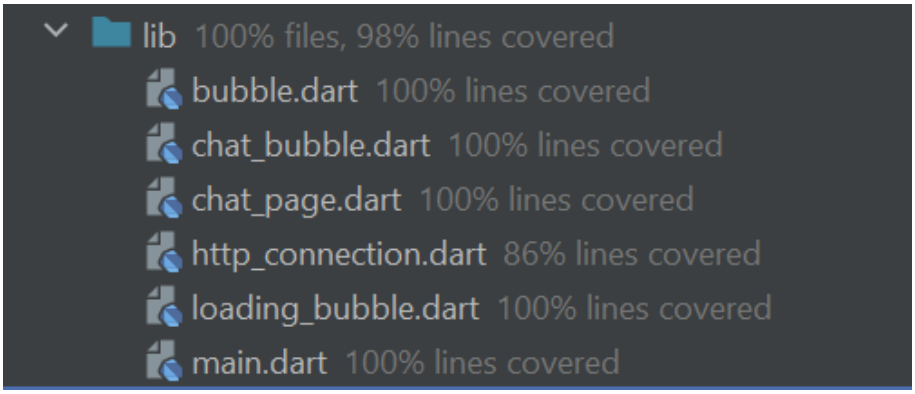

Figure 4.10: Coverage Test Result

4.4.2 Manual System Test

As the back-end code contains the NLU model, the test output is hard to predict, and therefore designing an automatic test process could be a challenging task. Consequently, we examine the performance of the system manually. For each of the four intents in the custom dataset, a test case is shown in Figures 4.11 to 4.14.

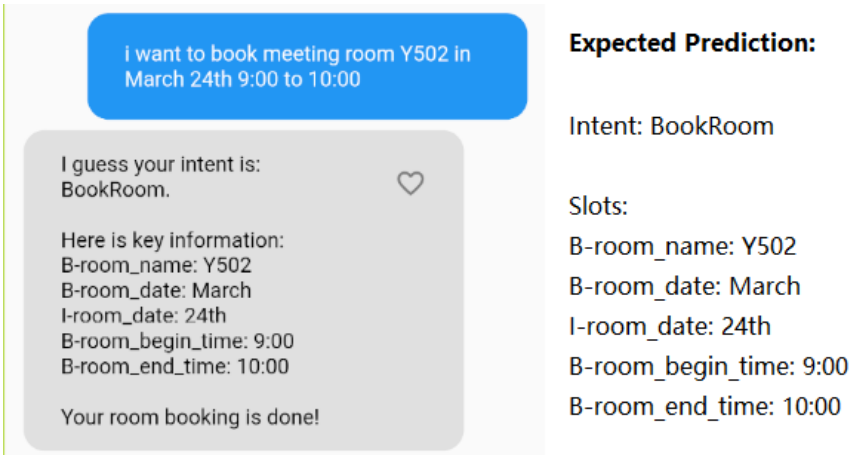

Figure 4.11: Test Result: BookRoom

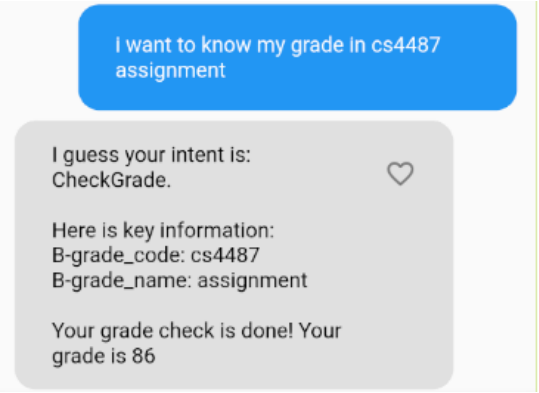

Figure 4.12: Test Result: CheckGrade

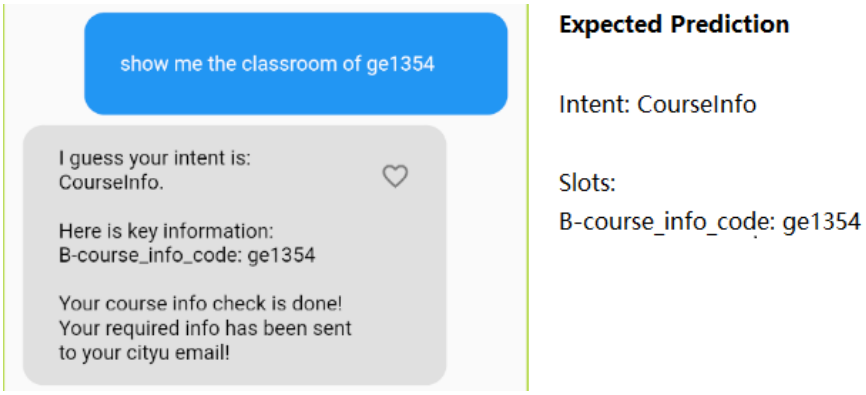

Figure 4.13: Test Result: CourseInfo

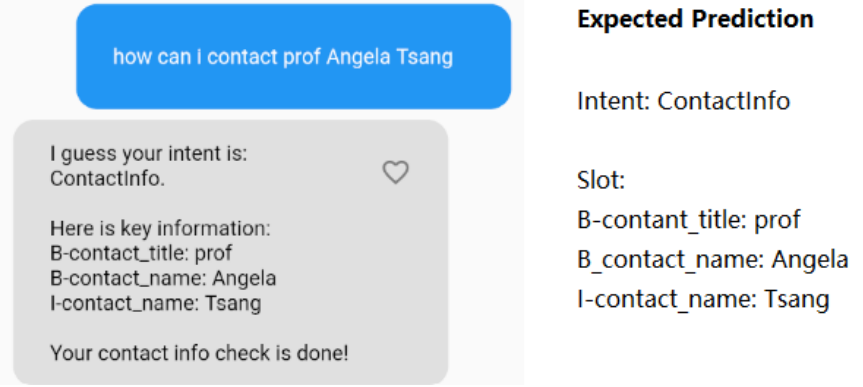

Figure 4.14: Test Result: ContactInfo

5 Conclusion

5.1 Achievement Summary and Critical Review

This project identifies the blank of an education-specific task-oriented speech dialog system in the market, proposes a chatbot solution equipped with a SOTA joint BERT model, analyzes the effect on the performance of some critical components in the model, constructs a custom education-related task-oriented NLU dataset, designed, implemented, tested and deployed the chatbot system with a cross-platform front-end interface and an extension-friendly back-end system.

One of the most serious difficulties that impeded the progress of the project was the blank of an education-focused dataset. Data used for training is directly linked to what intents and slots can be recognized by the model and, in turn, influences the tasks that the system is supposed to conduct. Though there exist some mature and large-scale datasets such as ATIS and SNIPS for researchers to experiment with their models, it seems that a dataset does not have sufficient motivation to be generated in such format with a specific purpose for education-related intents off-the-shelf so our project can directly utilize it. To solve this problem, we summarize the criteria and format to prepare such a dataset and collect data accordingly based on relevant real-life experiences. Training on this custom dataset proves the feasibility of this solution.

The most time-consuming stage of this project locates in the model experiments on various hyperparameters. In this project, some critical aspects we tested are the alternation of BERT variants adopted in the model, the optimal number of training epochs, the effect of CRF, and the influence of the slot loss coefficient. The design and analysis of the experiments rely on an incisive understanding of the mechanism of neural networks, the mathematical representation of the NLU task, as well as an insight to convert excessive digits in the outcomes into meaningful diagrams, charts, and conclusions. This procedure is painstaking, but it also provides us a precious opportunity to systematically improve the ability to learn from previous academic work, design experiments, and analyze the results.

5.2 Suggestion for Extension of Project

Education-focused NLU-based speech dialog system, or chatbot, is a cross-disciplinary topic involving AI, NLP, Human-Computer Interaction (HCI), and many other areas. Our project, though with creativity and innovation highlighted in the dataset and model structure, has incredible potential to become a mature and competitive product that benefits its stakeholders, including teachers, students, and online education platforms. Three main potential valuable directions are identified to extend this project, namely data collection, software development, and predicting performance.

In terms of data collection, questionnaires can be designed to collect the education-focused task-oriented NLU data. With an adequate survey of the user group, the application scenarios of our chatbot system can be enriched, and in turn, the user experience will be enhanced.

From the perspective of software development, a back-end database for saved records may be an alternation for the current front-end volatile saved list. We believe that the former option, combined with the cross-platform characteristics of the application, will bring the user more flexibility. Besides, records management could also be optimized by incrementing other functionalities such as multiple deletions, saved tags, priority marks, etc.

Knowledge Graph (KG), a graph structure that reflects the relationships between entities [23], can be integrated with the RG unit to increase the response speed of the system. For example, the contact information of staff can be stored in a KG and then linked to the chatbot's GR unit so that when a *ContactInfo* task is realized, the RG unit can search for the data required directly in the KG to save time in proceeding the query.

References

[1] A. Khanna, B. Pandey, K. Vashishta, K. Kalia, B. Pradeepkumar, and T. Das, "A study of today's ai through chatbots and rediscovery of machine intelligence," *International Journal of u-and e-Service, Science and Technology*, vol. 8, no. 7, pp. 277–284, 2015.

[2] B. A. Shawar and E. Atwell, "Chatbots: are they really useful?" in *Ldv forum*, vol. 22, no. 1, 2007, pp. 29–49.

[3] E. Adamopoulou and L. Moussiades, "An overview of chatbot technology," in *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, 2020, pp. 373–383.

[4] L. C. Klopfenstein, S. Delpriori, S. Malatini, and A. Bogliolo, "The rise of bots: A survey of conversational interfaces, patterns, and paradigms," in *Proceedings of the 2017 conference on designing interactive systems*, 2017, pp. 555–565.

[5] A. Maedche, C. Legner, A. Benlian, B. Berger, H. Gimpel, T. Hess, O. Hinz, S. Morana, and M. Söllner, "Ai-based digital assistants: opportunities, threats, and research perspectives. bus inf syst eng 61: 535–544," 2019.

[6] D. Khashabi, *Reasoning-Driven Question-Answering for Natural Language Understanding*. University of Pennsylvania, 2019.

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

[8] J. Zhou, H. Wu, Z. Lin, G. Li, and Y. Zhang, "Dialogue state tracking with multi-level fusion of predicted dialogue states and conversations," *arXiv preprint arXiv:2107.05168*, 2021.

[9] D. Guo, G. Tur, W.-t. Yih, and G. Zweig, "Joint semantic utterance classification and slot filling with recursive neural networks," in *2014 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2014, pp. 554–559.

[10] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[11] C. Raymond and G. Riccardi, "Generative and discriminative algorithms for spoken language understanding," in *Interspeech 2007-8th Annual Conference of the International Speech Communication Association*, 2007.

[12] M. Jeong and G. G. Lee, "Triangular-chain conditional random fields," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, no. 7, pp. 1287–1302, 2008.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[15] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[16] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pre-training approach," *arXiv preprint arXiv:1907.11692*, 2019.

[17] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," *arXiv preprint arXiv:2006.03654*, 2020.

[18] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[19] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," 2020.

[20] C. Zhang, Y. Li, N. Du, W. Fan, and P. S. Yu, "Joint slot filling and intent detection via capsule neural networks," *arXiv preprint arXiv:1812.09471*, 2018.

[21] W. Xu, B. Haider, and S. Mansour, "End-to-end slot alignment and recognition for cross-lingual nlu," *arXiv preprint arXiv:2004.14353*, 2020.

[22] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[23] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, "A review of relational machine learning for knowledge graphs," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 11–33, 2015.

Appendices

A Monthly Logs

A.1 October

1st – 3rd: Search and read relevant research works for chatbot architecture, current solutions, and application scenarios.
4th – 8th: Compose Introduction section for FYP interim report.

9th – 13th: Search and read relevant research work for Natural Language Understanding (NLU) existing solutions and its application in Chatbots.
13th – 17th:

- Compose Literature Review section for FYP interim report.
- Revise the structure of the written report.

18th - 20th:

- Survey on cross-platform user interface existing solutions.
- Set environment for Flutter, a front-end cross-platform language based on Dart.
- Set environment for Joint Intent Classification and Slot Filling BERT Model (based on Chen et al., 2019).

21st – 24th:

- Learn to write code in Flutter and Dart.
- Try to run Chen et al.'s code and reproduce the result of classification and slot filling tags on provided test sets.

25th – 28th:

- Analyze the joint model structure proposed in Qian et al.'s paper.
- Read the code for the joint loss function implementation.

29th – 30th: Review monthly work and meet with supervisor.

A.2 November

1st – 3rd: Design the System Diagram for the proposed software.
4th – 6th: Analyze requirements for the users and sketch the use case diagram.

7th – 8th: Based on the understanding of Qian et al.'s model, analyze the strengths and weaknesses.
9th – 10th: Design experiments based on the analysis:

- On the selection of pre-trained models,
- On the manner to combine the loss function.

11th – 12th: Compose the Methodology section of Interim Report I.
13th – 14th: Compose the Software Design section of Interim Report I.

15th:

- Make tentative schedule for the following stage,
- Revise Interim Report I.

16th – 18th: Design the 'add to history list' function in the user interface.
19th: Test for the 'add to history list' function.

20th – 25th: Design the conversation page in the user interface.

26th – 30th: Test for the conversation page.

A.3 December

1st – 5th: Refactor the class diagram of front-end to add loading bubble animation effectively.
6th – 8th: Implement loading bubble animation when the system is waiting for response.
9th – 12th: Integrate the animation to the system.

13th – 15th: Repeat the experiment of joint BERT model and draw charts for analysis.
16th – 18th: Learn to use the back-end Python library Flask and create a small demo.
19th – 20th:

- Create front-end functions to send and receive messages to and from the back-end system.
- Prepare presentation and slides for progress report to the supervisor.

21st – 23rd: Implement the POST API in the back end to receive messages from the front-end.
24th – 26th: Connect both ends with the POST method and test.

27th - 29th: Integrate the history record list to the system.

31th – 31st: Write test cases to do a thorough and systematic test.

A.4 January

1st – 7th: Do experiments of pre-trained models on ATIS dataset.
8th – 14th: Do experiments of pre-trained models on SNIPS datasets.

15th – 17th: Process the data obtained from the experiments and conduct analysis.

18th – 21st: Design coverage tests for front-end code.

22nd – 25th: Create charts, tables and pictures for experiment results.

26th – 28th: Compose Interim Report II.

29th: Meet with supervisor to discuss the work in the nest stage.

30th – 31st: Revise and amend the Interim Report II for submission.

A.5 February

1st – 5th: Design the experiments on the effect of slot filling loss coefficient to performance of the model.
6th – 12th: Conduct the first group of experiments (with crf) and collect the data obtained during training.
13th – 18th: Conduct the second group of experiments (without crf) and collect the data obtained during training.
19th – 20th: Analyze the experiment data and make charts to illustrate the findings.

21st – 23rd: Design the custom dataset labels and file structures.

24th – 28th: Create data items for the custom dataset according to the predefined data labels and file structures.

A.6 March

1st - 3rd: Format the dataset files and train the model on the custom dataset.
4th - 10th: Integrate the trained model to the back-end system.

11th - 15th: Implement the RG unit based on the intent labels of the custom dataset.

16th - 18th: System test for back-end and network connection.

19th - 24th: Prepare diagrams, charts, pictures and compose draft for the final report.
25th - 31st: Revise and modify the final report. Prepare other documentation and files for final submission.