Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter now, including display on the World Wide Web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Shen Gao April 7th, 2019

Cloud-based Active Learning System for Question Answering on Multiparty Dialogue

by

Shen Gao

Dr. Jinho Choi
Adviser

Department of Computer Science

Dr. Jinho Choi

Adviser

Dr. Ken Mandelberg

Committee Member

Dr. Shun Yan Cheung

Committee Member

Dr. Michael Carr

Committee Member

2019

Cloud-based Active Learning System for Question Answering on Multiparty Dialogue

By

Shen Gao

Dr. Jinho Choi

Adviser

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Sciences with Honors

Department of Computer Science

2019

Abstract

Cloud-based Active Learning System for Question Answering on Multiparty Dialogue
By Shen Gao

This thesis presents the design and architecture of an Active Learning system for Question Answering on Multiparty Dialogue. The goal of this system is to collect a robust Question Answering dataset and to improve the performance of the system on Question Answering challenges on Multiparty Dialogue. The system has an interactive web-based user interface which allows users to challenge the system with their own questions regarding a short passage of dialogues between multiple characters in a TV series. This system makes use of a state-of-art Machine Learning model to predict the answers to users' questions. In the same time, the system learns from users' responses and performs online update on the model. The system uses probability functions to guide user towards contributing data needed most for model improvement. The system is designed to handle heavy internet traffic by efficiently storing data and by carefully synchronizing the shared resources in the web system. The system has shown promising results in guiding users to contribute high quality data useful for model training.

Cloud-based Active Learning System for Question Answering on Multiparty Dialogue

By

Shen Gao

Dr. Jinho Choi

Adviser

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Sciences with Honors

Department of Computer Science

2019

Acknowledgements

First, I would like to thank my advisor, Dr. Jinho Choi for his support on this project. I joined Dr. Choi's lab in my junior year. Dr. Choi has not only taught me the fundamentals about Natural Language Processing when I was completely new to the field, but also guided me when I faced difficulties in this project. I truly appreciate the opportunity to work with Dr. Choi.

Secondly, I would like to thank my thesis committee members, Dr. Ken Mandelberg, Dr Shun Yan Cheung and Dr. Michael Carr for providing constructive advice on this thesis.

Lastly, I would like to thank my fellow researchers at Emory NLP lab: Zhengzhe Yang, Liyan Xu, Han He, Gary Lai, James Finch, Changmao Li, Jose Coves, Xinyi Jiang and Kate Li for the insights they provided during lab meetings, that made this thesis possible. I would also like to thank my girlfriend, Wenzhu Pan, who is a student at Goizueta Business School for the emotional support she provided during my research career.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Question Answering

The topic of Question Answering has been drawing the attention of the Natural Language Processing community for many years. Question Answering is a Computer Science discipline focused on building automated systems which are able to answer questions from humans in natural language [2]. Based on this concept, a question answering challenge usually starts by giving the machine a question and texts containing relevant information to the question in natural language. The machine is then responsible for providing an answer to the question raised.

Question Answering is crucial in improving the human-machine interaction, because in order to attempt the Question Answering challenge, the machine must not only be able to understand the question in natural language but

also understand the relevant context for forming an answer. Some of the most widely used applications of Question Answering system in the industry are the variety of personal voice assistants: Google Assistant, Apple Siri and Amazon Echo etc, which are able to answer questions based on the open-domain context of almost the entire internet.

1.2 Question Answering on Dialogue

There have been a number of attempts made in the Natural Language Processing community on Question Answering challenges with a variety of texts and questions. The specific classifications of the questions will be discussed in Chapter 2.1. The texts on the other hand, vary by size and types. Some data-sets simply provide a knowledge base containing quantities of passages as context, which requires the model to use Information Retrieval models to filter relevant passages to a specific question. Examples of this kind includes **Quasar-T** (Question Answering by Search and Reading) [4], which is based on Trivia question set with 100 passages collected for each question; **Search QA** [5], which is based on Jeopardy! question set with 50 web articles collected for each question. Other data-sets provide more specific passages, usually annotated from non-fiction writings or descriptive articles. Examples

of this kind include **SQuAD** (Stanford Question Answering Dataset) [8], which is annotated from Wikipedia articles.

Regardless of the corpus size, all of the passages selected by previous work feature non-fictional, descriptive or scholarly writings as relevant context to the questions raised. Very little work has been conducted on Question Answering with dialogue as context. Dialog data is significant, because not only it is available in large quantity, but also the quantity is growing rapidly. There were approximately 21.9 billion online messages sent each day in the year of 2017, an increase of 17% increase from an daily average of 18.7 billion from the year of 2016[10]. This resource only recently became available due to the rapid growth of information technology. The ultimate aim of this project is to collect a robust question answering data-set based on dialogue data as well as to explore ways of attempting the Question Answering challenge on this data set.

1.3 Active Learning

During the annotation phase of the project, there were some impediments, on which Chapter 2.2 will elaborate. In order to collect more data for a more robust data set and explore different ways to improve the performance of

model, a web-based online Active Learning system was developed in parallel to the conventional offline data collection and offline model tuning.

Active Learning is a sub-branch of Machine Learning in which the learning system will interactively query the user to obtain the desired data from the user[9, 11]. In the implementation of the Active Learning model for Dialogue Question Answering, the system will first analyze the performance of the Machine Learning model against a fixed set of test cases, then prompt the user who use the system to produce data that could potentially assist in the improvement of the model, and finally the system will periodically use the collected and targeted data to train and improve the model.

The main focus of the this thesis will be on the Active Learning system built to assist the Question Answering model, including its architecture and much of the engineering details.

1.4 Layout of the thesis

Chapter 2 will examine the rest of the background information needed for illustrating the Active Learning system. The chapter will begin by defining the specific types of questions we aimed to attempt, the data collection or annotation process for our data, the source and statistics of the data set

generated and the baseline Machine Learning model selected for the Active Learning system.

Chapter 3 will discuss the design and implementation of the Active Learning system. It will start by discussing the implementation details of the system as a web application and then move on to the system as a machine learning service.

Chapter 4 will summarize the work done in the deployment and testing of the system as an web application and then analyze the results and collected by the time writing this thesis.

Chapter 5 will provide a summary to this project and intended future work on the Active Learning system after the writing of this thesis.

Chapter 2

Background

This chapter focuses on the work done before the development of an Active Learning system. Section 2.1 will start by categorizing the common types of the questions with examples and defining the question types we picked to attempt in Dialogue Question Answering. Then Section 2.2 and Section 2.3 will discuss the results and process of the annotation phase of the project as well as its format and statistics. Finally, Section 2.4 will discuss the performance and methodology behind the baseline Question Answering model, BERT [3] (Bidirectional Encoder Representations from Transformers), that we picked for the Active Learning system.

2.1 QA question types

2.1.1 By Answer Formats

During the preliminary stages of the data collection, the project started with iterations of experimental annotations by the researchers who are part of the project to explore what potential questions are meaningful in the context of dialogues and what forms the reasonable answers to them take. By English grammar [7], questions can be categorized into "Yes/No" questions and "W" questions. The previous require a binary answer while the latter have open-ended answers.

Since the binary questions by definition only have a limited set of answers, no restrictions need to be applied on the question during data collection.

On the other hand, "W" questions can have answers of arbitrary length and content. We applied the following limitation on the answer to "W" questions: the answer must exists as a continuous span of texts inside the original passage. This procedure is common in most popular Question Answering data sets. Examples of data sets that follow this procedure include all of the data sets mentioned in Chapter 1: **Quasar-T** [4], **Search QA**, [5], **SQuAD**, [8].

2.1.2 By Degree of Inference

Even if the answer to different questions take the same format, the questions themselves can be vastly different in terms of the inference needed to resolve the question. Take the dialogue in Table 2.1 as an example:

[Scene : A kitchen somewhere . Monica is interviewing for a job] **Interviewer** Alright , let s see if you 're as good in person as you are on paper . Make me a salad . **Monica Geller** A salad ? Really I , I could do something a little more complicated if you like . **Interviewer** No , just a salad will be fine . **Monica Geller** You got it . **Interviewer** Now , I want you to tell me what you 're doing while you 're doing it . **Monica Geller** Alright , well I 'm tearing the lettuce . **Interviewer** Uh-huh . Is it dirty ? **Monica Geller** Oh - oh , no no do n't worry , I 'm gon na wash it . **Interviewer** Do n't , I like it dirty . **Monica Geller** That 's your call . **Interviewer** So , uh , what are you going to do next ? **Monica Geller** Well , I thought that I would cut up the tomatos . **Interviewer** Are they , uh , firm ? **Monica Geller** They'r alright . **Interviewer** You sure they have n't gone bad ? You 're sure they 're not very , very bad ? **Monica Geller** No really , they 're OK . **Interviewer** You gon na slice them up real nice ? **Monica Geller** Actually , I was gon na do them jullienne . **Interviewer** Aaaahhhhhhh . **Monica Geller** I 'm outa here .",
Q1 Does the interview includes making a salad? **Q2** Is the interviewer picky?

Table 2.1: Sample Dialogue from TV series Friends

In order to resolve **Q1**, one can use the textual similarity of the highlighted text to collect information on the fact that Monica is taking an interview about making a salad and therefore answer,"Yes". Very little inference is needed from the text to draw this conclusion. In attempting to resolve **Q2**, one can only infer from the fact that the interviewer kept asking about the state of the ingredients of the salad, making specifications on the salad and the impatient reactions from Monica to conclude that the interviewer is a picky person. Significant inference is needed and there is no direct text that has a similar meaning to picky.

Questions like **Q1**, which are answerable without inference from the contextual similarity between text inside original passage and those inside questions, are classified as **Explicit Questions**. Questions like **Q2**, which are answerable through inferences of the text with little or no textual similarities between the passage and the answer, are classified as **Implicit Questions**. This type of question in dialogue usually describes some kind of state of an entity inside the dialogue.

2.2 Annotation

The passages for dialogues are selected from the transcripts of famous TV series **Friends**. The goal of the annotation is to generate question-answer pairs from all episodes of **Friends**. Given the number of scenes inside the TV series and limited researchers we have for annotation, we decide to make use of the online crowd-sourcing platform, **Amazon mTurk** to gather support from cloud workers. Instead of using the whole episode which can be lengthy and hard to grasp, we broke down the episodes into a more reasonable unit of scenes, from which the questions need to be generated.

As far as question types go, we picked both "W" questions and binary questions as types for annotation. In terms of Explicit and Implicit Questions, we only picked implicit questions for annotation on binary questions. If we were to generate question answers pairs for "W" questions, we would need to provide the states to be selected for the entities inside the conversation, since the states describing an entity are often not available in the original passage. Finding a comprehensive set of adjectives describing the possible states is challenging. An incomplete set of states to choose from, posted in instructions to workers, would exert unwanted guidance on the workers. The workers could only generate questions based on the selections.

2.2.1 Amazon mTurk

Figure 2.1: Amazon Mechanical Turk Worflow

Amazon mTurk is a crowd sourcing platform which helps individuals or small businesses outsource jobs to a distributed workforce. To start with, the requester uses HTML and JavaScript to build a web template containing the answer fields to be collected, which will be displayed to the workers who attend the job. Then, a csv data file is used to populate the template with information describing a specific task. The requester also specifies the reward for each question completed. Additionally, the requester has the option of specifying criteria, which workers need to fulfill in order to work on the question including: native language, experience ... etc. Once the jobs are posted, the workers use the web interface to complete the tasks and the system collect results automatically. The requester finally has the chance of

performing quality control before paying the workers by rejecting the answers that do not follow the instructions. The quality control phase also concludes a batch of mTurk cycle.

2.2.2 Process of Annotations

The annotation process started with experimental annotations during which we used small batches of data for question-answer generation so that we could tweak the instructions and the interface for optimal results. Annotation for the two types of questions were run in parallel and I was primarily responsible for the binary questions. We developed two separate interfaces for the "W" questions and binary questions. The prior has span selection tool which allows user to select a continuous span for each of the questions. The latter allows user to choose from a drop-down box of answers Yes and No. One small detail worth noticing here is that we allowed users to select more than one span for each question. We also supplied each template with the data granulated into scenes from the transcript of the seasons of **Friends**. In order to provide a uniform distribution of question types, we asked workers to generate equal number of implicit and explicit questions for the binary case. Since each scene varies in the amount of information for types of "W" questions, we asked

annotators to generate 4-6 "W" question with no limitation on the question types. We also made use of the JavaScript functions to do preliminary quality assuring on the input data includes ensuring all questions are attempted and answered. We also proof read all of the responses from annotators to reject question answer pairs that obviously diverged from the context of **Friends** dialogue as supplemental quality control.

Following the question-answer generation phase, another phase of annotation, the verification phase was conducted to record the agreement of human performance on the types of questions. The goal of the verification phase was to ensure high agreement between human beings on the question generated. We developed two other similar annotation interfaces which allow different workers to attempt the questions created by workers in question-answer generation phase to record **Inter-Annotator Agreement**. The agreement statistics was computed by F-1 score evaluation metric[8] for "W" typed questions and exact-match percentages for the binary questions.

We experienced challenges during the annotation for both "W" questions and binary questions. For "W" questions, the initial batch showed that some question, were too ambiguous to answer. For binary questions, we noticed that the use of ambiguous pronouns in generated questions caused the confusion

between annotators. We also noticed that collected questions contain much of the wording from the original questions which would undermine the quality of data for training. This occurred despite the fact that instructions had been provided to require paraphrasing from original text.

We updated the instructions to disallow the use of pronouns, and to substitute pronouns with explicit names of entities. We also supplied plots from previous annotation on the scenes[13], which is paraphrased, descriptive text of the content of a conversation in a scene, to the annotators of binary questions so that they could directly combine the plot information with original text to form questions. We also tried to reduce the number of questions required to be generated from each scene, hoping for higher quality binary questions.

2.3 Friends Dataset

Despite the effort made to improve the agreement between question-answer generation phase and verification phase in binary questions, we could not obtain high-agreement data between the annotators in experimental phase. The following table summarizes the result of the agreement in binary questions.

ITA change Binary Annotations			
Round	Total EM	Implicit EM	Explicit EM
1	70.45%	70.45%	70.45%
2(P)	79.54%	72.72%	86.37%
2(NP)	77.08%	81.25%	72.91%
3	69.23%	74.35%	64.10%

Table 2.2: Inter-Annotator Agreement of Binary Question Annotations. **P** denotes with plots presented to annotators and **NP** denotes the scenes without plots presented the annotators; Question Number dropped from 4 to 3 per scene in round 3

Since the answer is strictly binary, a random guess would produce an expected value of agreement of 50%. We aimed that the human agreement to stabilize in high 80% to 90% for the data to be useful in training, which was never achieved. We also expected the agreement for explicit questions to be higher than that of implicit question since the latter would require more inference from the text to solve and should result in more divergence in the opinions of annotators. Since no high quality data was available for binary typed questions, we decided to drop those types of questions in the further

development of the project.

The annotations for "W" typed questions, on the other hand, showed promising results in experimental phase. The Inter-Annotator Agreement remained stable above 80% indicating the questions generated could be useful in the development of an Question Answering model.

ITA change in "W" annotations	
Round	F1
1	83.42%
2	83.99%
3	83.12%
4	88.17%

Table 2.3: Inter-Annotator Agreement of "W" Question Annotations.

With the promising results observed in Table 2.3, we proceeded to collect the **10,610** questions from **1,222** scenes. The overall Inter-Annotator Agreement remained at the level stayed at **81.82%**. The questions are further classified by their types as:

Type	Count
What	2,058
Where	1,896
Who	1,847
Why	1,688
How	1,628
When	1,493

Table 2.4: Count of "W" questions by type

2.4 Baseline Model

The Bidirectional Encoder Representations from Transformers (BERT) [3] is developed by Google AI for Question Answering. Unlike conventional models which usually follows End-to-End design, BERT incorporates a pre-trained layer learned from text data publicly available on the web capturing the general grammar and contextual information on a language. It also introduces a new architecture allowing bidirectional flow between layers of network (Figure 2.2). It have showed outstanding performance on many other question answering data sets and pushed the F-1 score and Exact Match to an unprecedented record of **87.4%** and **93.2%** in **sQuAd** data sets [8].

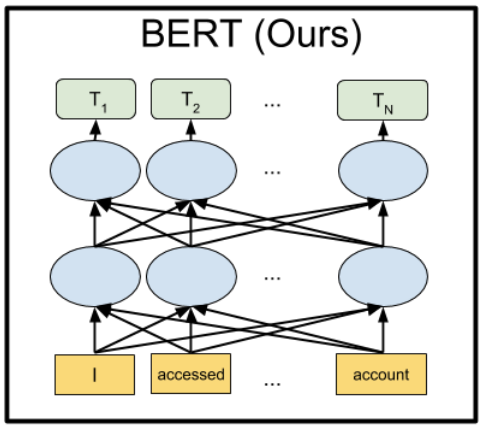

Figure 2.2: Bidirectional Flow in BERT; picture edited from [3]; The arrows indicate the information flow from one layer to the next. The green boxes at the top indicate the final contextualized representation of each input word

The code for BERT has been open sourced by Google and thus is convenient for integration with the system. Limited by the resource available, we were only able to train the BERT model with 12 layers with **80%** of the total annotation we collected, totaling **8535** questions. We also used the pre-training provided by Google since the pre-training process is time consuming and the computing resources needed are beyond our power. According to Google, the pre-training provided took 4 days on 4 to 16 cloud TPUs [3].

Chapter 3

Approach

In retrospect of the failure of collecting high quality annotations for the binary question types, we decide to explore unconventional way of model learning as well as data collecting. This chapter will focus on the design and development of the Active Learning system. The goal of the system is to guide users towards contributing to types of data needed by the model to improve, as well as actively learning from the responses of users online. Section 3.1 will explain the interaction design between the user and the system. Section 3.2 will discuss details on how the model guides user towards better input for the system. Section 3.3 will illustrate the web-application design details of the system as well as security and performance considerations. Section 3.4 will explain how the system is optimized for incorporating the BERT model into online production.

3.1 User Interaction

Traditional data collecting platforms like Amazon mTurk provide a platform where workers can provide contribution to the data set in exchange for monetary rewards. To the workers, on the other hand, the optimal strategy would be to minimize the amount of time spent on each task to maximize the reward. The quality assurance check from the requester serves as a bar raiser for the quality of the work. But for tasks such as those in verification phase where the worker is asked to attempt questions of which ground-truth answers are unknown to the requester, manually performing the quality assurance requires the same amount of work as the work to be done by the workers. Therefore, it is infeasible for the requester to perform quality control on tasks like these.

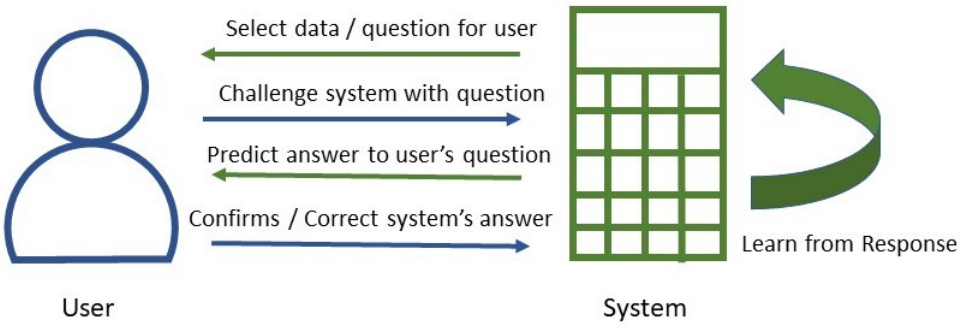

Figure 3.1: User Interaction with Active Learning Model

In our Active Learning System design, we try to explore a user-system relation which involves more interaction and hopefully will inspire the user with more spontaneity in contribution. We start with a model trained for answering "W" questions from scenes in **Friends**. The system selects scenes and question types for the user to work on. Then user can challenge the system with a question of the type specified. The system queries the model for an answer, and displays the answer to the user. The user then has the chance to confirm or correct the system's response. The collected responses are trained automatically by the system in batches to update the model. The user can witness the growth of the model by visiting the sites and asking the same type of questions, which, we hope, will encourage the user to contribute more on the system.

3.2 User Guidance

The Active Learning System provide guidance on user input by two means: selecting the **scene** and selecting **question** type for user to contribute to. For the scenes, the goal is to ensure an even distribution among all scenes, excluding ones that will be reserved for testing. For question types, the goal is to encourage user to contribute more on the question types that: **1.**

annotated the least; **2.** the model showed insufficient performance on. On the other hand, since continuously requesting the same scene and question type would result in bad user experience, the system make use of random generation with probability to ensure enough freshness in the user experience.

Since the data set has abundant number of scenes available, for scene selection, we use an uniform distribution for the scenes which has least count of annotation:

$$P(S_i) = \begin{cases} \frac{1}{N_{min}} & S_i = S_{min} \\ 0 & S_i \neq S_{min} \end{cases} \tag{3.1}$$

where S_i denotes the count of annotation of Scene i, S_{min} denotes the the min of all S_i and N_{min} denotes $\mid \{S_i \mid S_i = S_{min}\} \mid$.

For the question type selection, we start by defining the normalized question type count $\hat{Q}_{iN} = \frac{Q_{iN}}{Q_N}$, where Q_{iN} is the count of annotations on question type Q_i and Q_N is the total count of annotations.

$$f(Q) = 3 - (\hat{Q}_{iN} + Q_{iF1} + Q_{iEM}) \tag{3.2}$$

$$P(Q) = \frac{c-1}{e^3 - 1} \times (e^{f(Q)} - 1) + 1 \tag{3.3}$$

Here Q_{iF1} and Q_{iEM} denotes the F-1 score and Exact Match for question type Q_i and c is a scaling factor used to control the upper bound of the

probability of a single question type. Since Q_{iN}, Q_{iF1} and Q_{iEM} all have range $[0, 1]$, $f(Q)$ has range of $[0, 3]$. The exponential function will then have range of $[1, e^3]$. The factor c scales the probability linearly range to [1, c]. It denotes the ratio of probability of a question type with minimal statistics being chose and one of an answer with maximal statistics being chose. The exponential function guarantees the discrepancy between low stat and high stat will rise rapidly while c ensures a question type with perfect statistics still have $\frac{1}{c}$ chance of selected with respect to a question type with minimal statistics. The final probability is computed by normalizing $P(Q_i)$ for all question types.

In order to prevent the side effect that revealing the test case statistics in user guidance could cause biases in our results, a set of data disjoint with the **training data** and **test data** is separated and used for obtaining statistics for user guidance purpose only. Here Q_{iF1} and Q_{iEM} are the statistics from such **dev** data sets. The true exact match and F-1 scores of the **test data** are not visible to the system during user guidance.

3.3 Architecture

The implementation of the system as an web application follows the design pattern of Model-View-Controller[12]. In this design pattern, the Model is responsible for managing all of the data relevant to the web application. The View is responsible for organizing and displaying the data to the user. The Controller contains the logic needed for interactions between the model and view as well as serving as the handlers to the requests from front-end.

Many pieces of open-sourced software are incorporated into the build of the Active Learning System. The Model layer is implemented using standard mySQL database which stores all of information on the passages text, user responses and machine learning model statistics. The View part is implemented in HTML and JavaScript with the help of JQuery selectors. The Controller part is implemented in python using the Django Framework[14], which is a python based backend server framework. The Model and the View is networked through standard REST API calls, while the communication between controller and model is implemented by the convenience of Django Object-Relation Mapping(ORM)[14], which allows manipulation of database entries in python.

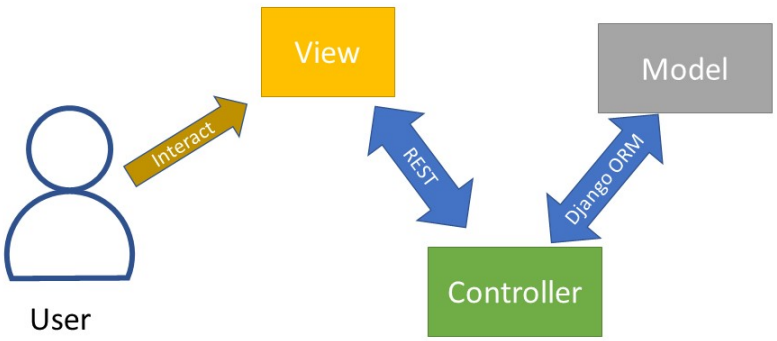

Figure 3.2: Model-View-Controller Design in the System

3.3.1 Database

The following database schema has been designed to support the operation of the Active Learning System.

In the Figure 3.3, all entities with solid contour represent a table and all with grey contour rep sent an abstract entity. Solid arrows represent Foreign Key Pointing while grey arrows represent Inheritance in python code.

In an attempt to further optimize the database performance, Indexing is applied to all fields related to serve a user request, which includes: **hash** field on **User Response**, which is needed when user correct the answer from

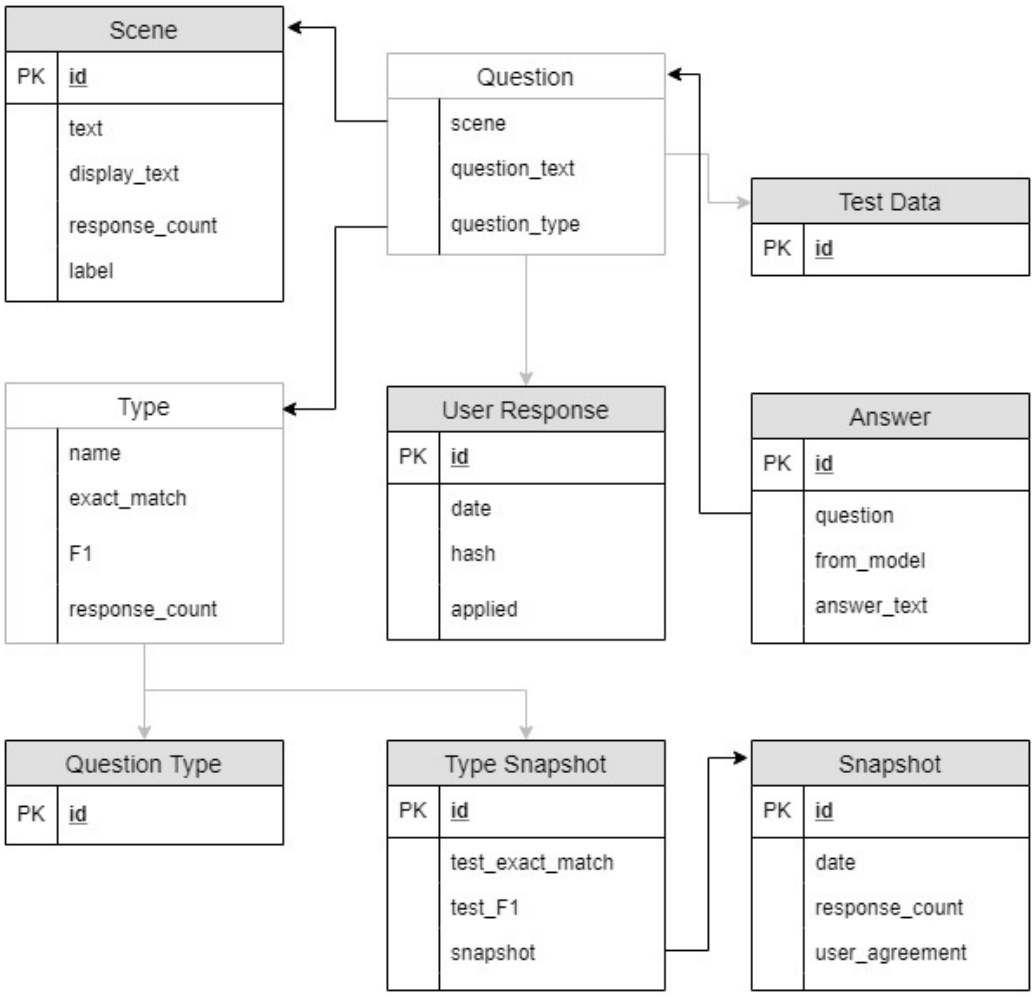

Figure 3.3: Database Schema for the Active Learning System

system. **response_count** field on **Scene**, which is needed when the system selects a scene to serve the user. Aside from database indexes, since updating index can be expensive, all writes into database are postponed after the controller has returned response to user by starting a separate thread just before the return the response.

In order to ensure consistency in the database during potential concurrency racing conditions, row-level pessimistic lock [1] is applied on the **Question Type** table and **Scene** table, since different users can respond in a racing condition and increment the count of the total number of a specific scene. The row lock prevents the dirty writes into the field of **count** and grantees the ready-after-write constraint on **count** fields, such that the user guidance module (Section 3.2) can use the latest available data on computing the probability for each scene and question type.

3.3.2 Application Programming Interface

Three Representational State Transfer (REST) based API calls are designed for communication between the front and back end. Since adding a user management system could discourage people from using the system as it would require all visitors to register and sign in, all interfaces are open to

public without authentication.

As soon as the front-end page loads, a GET call is sent to controller, requesting for the scene and question type data. To reduce the overhead in front end rendering, the HTML text of each scene that needs to be rendered is pre-computed and stored in **display_text** in **Scene** table. The controller follows the user guidance procedure discussed in Section 3.2 to pick a scene and question type then returns the scene display text and question type.

After the user finished composing a question, a second POST call is sent to controller, which contains the payload with the question submitted by the user. The question is verified by front-end to be consistent with the question type required by the first GET call. The controller will then create a **User Response** entry to store the question from the user. The controller then queries the BERT service for an answer to the question, which the controller will store as an **Answer** entry in the database with **from_model** bit turned on. Finally, the controller returns the prediction to the question and the view will highlight the answer span inside the selection area in a different color from user selection.

Once the user edits or confirms the answer, the third POST call is made from front-end to either to confirm model prediction or to correct model

prediction. The controller will then add user's correction as another **Answer** entry with **from_model** bit turned off. Note since the interfaces are open to public and no authentication is required, a malicious user could abuse the third POST call to change the responses from a previous user arbitrarily if the interface is not implemented with care. To resolve the potential loophole, the controller assigns each response with a unique identifier obtained by Django PBKDF2 Password Hasher [14] with input from current time stamp, scene id and a random salt in the second POST call and returns the hash with answer to user. In other words, the controller creates a password hash for each response which is hard to guess, given only the id of the scene. The id of the scene is the only piece of information public to the users. This ensures a malicious third party cannot obtain the hash by brute force attack without significant amount of computation. The hash is also needed in the payload of the third POST call along with the correction information to be used for identification purposes for the original response.

3.3.3 User Interface

The User Interface of the web application is straight-forward. The user starts interaction with the web application by reading a section containing an introduction of the website, then the user needs to pass a tutorial for using the span selection tool developed for answer selecting purpose (Figure 3.4). Once the user has completed the tutorial, the main interface (Figure 3.5) populates and the user can submit a question using the input fields on the right hand side. Since the scenes are selected at random and there are cases where a certain type of question is impossible to formulate in one scene, we provide a button for user to get another scene and question type pair. Once the user submits the question, the the system returns a prediction from the model, which is highlighted using a different color on the passage text on the left hand side. The user can finally use the span selection tool to change the answer span and submit the correction to the server.

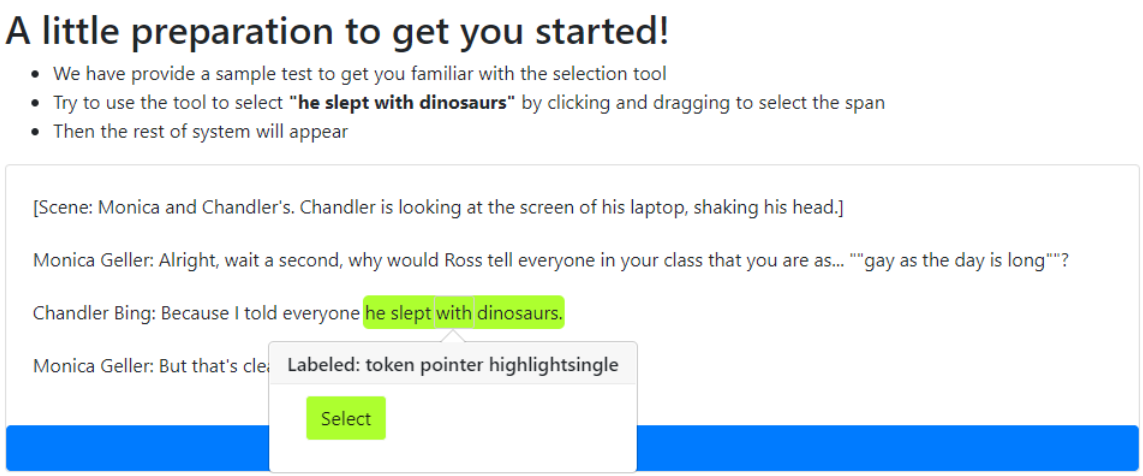

Figure 3.4: Span Selection Tool Tutorial

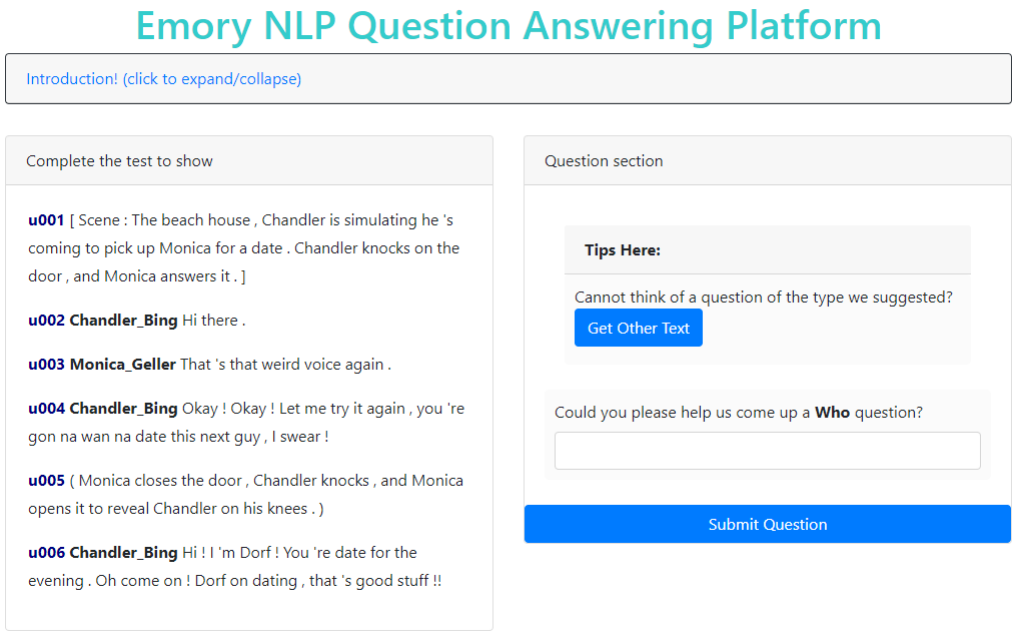

Figure 3.5: Main User Interface

3.4 BERT Service

The open-sourced code for BERT model provides interface for offline training and testing[3]. Our Active Learning system needs the BERT model to be integrated into a service which is efficient, concurrency safe and also supports online updating. In order to modularize the model into an online service, the integration development starts by timing the overheads incurred during the offline execution of the BERT model.

Overhead of BERT execution	
Task	Execution Time (ms)
Convert input into feature	45
Write feature into file	10
Load model from file	4474
Use model for prediction	6771
Store write results back into file	5
Total	11305

Table 3.1: Stats for Overhead of BERT execution, the data is an average of 50 experiments on a single scene with same question rounded up to the closest integer, experiment is conducted on machine with i7-6700HQ with CPU computation

As shown in Table 3.1, excluding the time actually taken by computation to predict from input, we can optimize performance by loading model into memory and eliminate the file operations to save **41%** time. After ensuring all data transition is done in memory and the model is pre-loaded into memory

on server start, the average execution time of same query handling reduced to **6946** ms on the same query for 50 times.

On the other hand, the service needs to be configured for concurrency handling. We encapsulate the BERT interface into worker objects with a dedicated model loaded for each worker. On start up, the system creates a list of workers populated with model data loaded from file system. A mutex array is created to synchronize on the race conditions. A semaphore is used to keep track of the total number of available workers. The controller starts a query into BERT service by acquiring the semaphore by 1, then it checks the mutex array for the available workers and acquire the mutex. Once the prediction is returned, the calling process releases the mutex and then releases the semaphore control.

The training for the BERT service is invoked by scheduled cron jobs at fixed time intervals. The cron job starts training by querying the database and checks if enough response for one training batch has been generated. The answers generated by the system itself are then filtered out and training only starts if the qualifying response has reached a fixed batch size. After the new model checkpoint is generated, the training process populates another array of workers of the new model and simply changes the pointer of the workers

array to point to the new array. The operation is concurrency safe since if other processes are conducting prediction when the pointer change happens, workers will not be destroyed as long as it is pointed to by some local pointer in the request handler process.

3.4.1 Snapshot

The snapshot of the model performance is conducted via a cron jobs scheduled at vacancy hours. The service is implemented via crontab feature of the Linux system and a server interface reserved for local routing only. The controller starts the snapshot by computing F-1 score between the user and the model answers to the questions raised by user. The answer origin can be distinguished by the state of the **from_model** field in **Answer** table. The controller then save the count of total responses and the F-1 score (in field **user_agreement**) into a new row in Table **Snapshot**. Finally, the controller acquires a new worker from BERT service to test against all test data and dev data stored in database respectively and save the statics into Table **Type Snapshot**. The controller also updates the statistics in **Question Type** if any of the statistics changed.

Chapter 4

Experiment

After the conclusion of the development of the Active Learning system, the system is deployed into an staging instance on Web server and tested for concurrency and latency measuring. The system is then opened to public and an advertising email was sent to the students at the Department of Computer Science at Emory University. Section 4.1 will discuss the testing conducted before the system is shipped to production as well as the environment on which the web application is hosted. Section 4.2 will discuss the results we have collected by the time this thesis is written.

4.1 Pre-Deployment

4.1.1 User Guidance

To ensure the functionality of the User Guidance (3.2) module, a test was conducted to verify the probability distribution of each question type. For our experiment, we picked the scaling factor to be 3, which means the question type with the minimal possible statistics will have 3 times the probability of being chosen by the system for annotation as the question type with maximal possible statistics. The initial statistics is obtained from the BERT model with 12 layers, trained with 80% of the annotation collected through Amazom mTurk (Section 2.4), against **dev** datasets (Section 3.2). The test is done by 1000 test calls to the GET interface which returns a question type for annotation.

Question Type	EM	F1	$P(Q_i)$	$P(\hat{Q}_i)$	Frequency
What	34.90 %	50.53%	10.95	17.1%	169
Where	58.94 %	69.15%	7.49	11.7%	115
When	43.61 %	59.00%	9.38	14.7 %	147
Who	48.02 %	53.26%	9.49	14.8%	149
Why	19.75 %	37.84%	14.14	22.1%	224
How	30.13 %	40.26%	12.56	19.6%	196

Table 4.1: Results for the user guidance testing over 1000 ; All Question Annotation Count is initialized to 0; $P(Q_i)$ denotes the output from the probability function; $P(\hat{Q}_i)$ is normalized true probability

Results in Table 4.1 confirm the functionality of the user guidance module, having worst-performed **Why** typed questions being selected **94.7**% more than **Where** typed questions, which is the question type our model performed the best on.

4.1.2 Environment

The Active Learning system as a web application is hosted on Amazon Web Services (AWS). Since the system requires the use of GPU for accelerated model prediction, P3-EC2.2xlarge instance is chosen to support prediction and training of the machine learning module. The instance features 1 NVIDIA Tesla V100 GPU, pairing 5,120 CUDA Cores and 640 Tensor Cores, 61 GB memory and 16 GB GPU Memory, which meets the minimum requirement for training the BERT model with 12 layers. It provides better performance in model prediction with respects to the statistics collected on personal computer using CPU prediction in Table 3.1.

4.1.3 Latency Test

To ensure user experience, the performance of each programming interface is measured. The results in Table 4.2 demonstrated reasonable latency for the data collecting purpose of our project.

Request Time of API interfaces	
API	Time(ms)
get-scene	235
predict	3573
post-correction	193

Table 4.2: Request Time of API interfaces in staging instance; results are in ms and obtained by average of 100 requests; predict is tested by test data from 100 independent scenes of different size

4.1.4 Concurrency Test

The concurrency test is conducted using Apache JMeter [6], which is a software capable of generating requests in concurrency to a destination server address. This test is to ensure the functionality of the database locks to prevent dirty writes into the system as illustrated in Section 3.3. The test contains 1000 requests sent to server writing responses to a single scene of a single type and the result count on the scene and question type stayed clean.

4.2 Result

4.2.1 Data Collecting & Model Improvement

By the time writing this thesis, the model has been shipped into production for 7 days. Only limited responses and progress have been made given the short time frame but the web service is going to continue online to collect more data. So far, 151 responses have been collected from users and we have

not yet observed progress in terms of model performance, this is because the data collected so far only constitutes 1.76% of the training data (Section 2.4) which is too insignificant for performance improvement.

On the other hand, progress has been shown in user guidance. Compared to the data collected in annotation phase, during which we did not provide any kind of the instructions on which type of question to generate and allowed the user to choose freely, the data collected in the Active Learning system is more focused on weak question types despise the fact that users can choose to refresh the types and scenes on the user interface (Section 3.3.3).

Type	EM	F1	C_{AN}	P_{AN}	C_{AL}	P_{AL}
What	34.90 %	50.53%	2,058	19.39%	29	19.20%
Where	58.94 %	69.15%	1,896	17.81%	13	8.61%
Who	48.02 %	53.26%	1,847	17.66%	18	11.92%
Why	19.75 %	37.84%	1,688	15.91%	40	26.49%
How	30.13 %	40.26%	1,628	15.34%	40	26.49%
When	43.61 %	59.00%	1,493	14.07%	10	6.62%

Table 4.3: Comparison between data collected in Annotation vs in Active Learning; C_{AN}: Count of questions from Annotation; P_{AN}: Percentage of questions from annotation; C_{AL}: Count of question from Active Learning; P_{AL}: Percentage of questions from Active Learning

As shown in Table 4.3, compared to questions collected from annotation, user guidance in Active Learning system helped data collection to focus on three weak question types: **What**, **Why** and **How**. The three types

total constitutes **72.18%** in Active Learning with respect to **50.64%** during annotation.

4.2.2 User-Model F-1

Another important statistic that was kept track of during the operation of the Active Learning system is the User-Model F1. This represents the similarity between the ground truth answer provided by the user and the original prediction from the system, which is used to display to the user. Since the goal of the system is to obtain data to learn from, and the model cannot improve from its own prediction, this score also measures the reverse of available responses meaningful for learning.

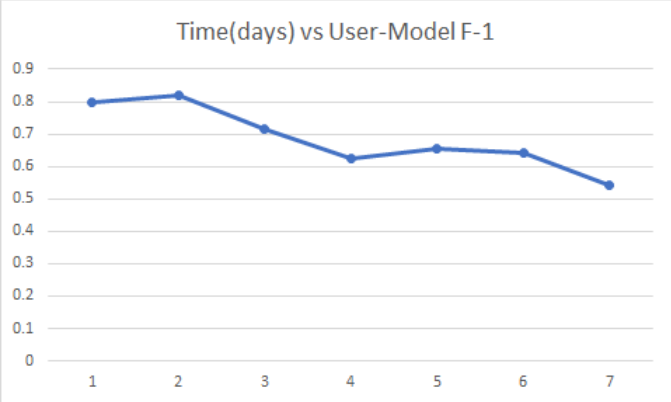

Figure 4.1: User-Model F1 change versus time

As shown in Figure 4.1, the user-model F1 score dropped over time, indicating more responses from user that the system can learn from were collected.

Chapter 5

Conclusion

This thesis presented the design and architecture of an Active Learning System implemented to assist the data collecting and development for Question Answering in dialogues. The challenge of collecting high quality training data (Section 2.2) inspired the development of such system which promotes more interaction in the data collecting for Question Answering data sets. The system at the same time uses probability functions (Section 3.2) to guide users towards contributing more to the data needed for model improvement. The system has shown results in guiding users towards annotating for the question types needed the model to improve. However, due to the limited quantity of the responses collected, we have not yet observed the improvement of model performance against the test cases.

For future work, since the progress of the system is currently limited by the response quantity, we intend to publicize the system towards larger

audiences including public online forums of Natural Language Processing and the Friends TV series to accelerate the data collection for this project. Once progress is made in terms of model performance, we can experiment with more types of user guidance and compare the effects of each to better summarize the methodology.

Bibliography

[1] Haki Benita and Haki Benita. How to manage concurrency in django models, Jul 2017. URL `https://medium.com/@hakibenita/how-to-manage-concurrency-in-django-models-b240fed4ee2`.

[2] Philipp Cimiano, Andrea Christina Unger, and John McCrae. *Ontology based interpretation of natural language*. Morgan & Claypool Publishers, 2014.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[4] Bhuwan Dhingra, Kathryn Mazaitis, and William W. Cohen. Quasar: Datasets for question answering by search and reading, 2017.

[5] Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Guney, Volkan

Cirik, and Kyunghyun Cho. Searchqa: A new q&a dataset augmented with context from a search engine, 2017.

[6] Bayo Erinle. Performance testing with jmeter test web applications using apache jmeter with practical, hands-on examples.

[7] Kitlum. Forming questions in english grammar. URL `https://www.fluentu.com/blog/english/questions-in-english-grammar/`.

[8] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[9] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender systems handbook*. Springer, 2015.

[10] Jeff Schultz. How much data is created on internet each day?, Oct 2017. URL `https://blog.microfocus.com/how-much-data-is-created-on-the-internet-each-day/`.

[11] Burr Settles. Active learning literature survey. *Computer Sciences Technical Report 1648*, Nov 2014. URL `http://pages.cs.wisc.edu/~bsettles/pub/settles.activelearning.pdf`.

[12] Artem Syromiatnikov and Danny Weyns. A journey through the land of

model-view-design patterns. *2014 IEEE/IFIP Conference on Software Architecture*, 2014. doi: 10.1109/wicsa.2014.13.

[13] Marilyn A. Walker, Heng Ji, and Amanda Stent, editors. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, 2018. Association for Computational Linguistics. ISBN 978-1-948087-27-8. URL https://aclanthology.info/volumes/proceedings-of-the-2018-conference-of-the-north-american-chapter-of-the-association-for-computational-linguistics-human-language-technologies-volume-1-long-papers.

[14] Thomas Walter. Das python-framework django. *Kompendium der Web-Programmierung X.media.press*, page 447461, 2008. doi: 10.1007/978-3-540-33135-3_22.