GROUP PROJECT REPORT

TOPIC: Conversational Chatbot

MEMBER: DO Van Quyet, TRINH Van Hoan, DO Thuy Trang
{vqdo, vhtrinh, ttdo}@connect.ust.hk

1. INTRODUCTION

Recently, **chatbots have emerged** as a great tool for automotive communication. Essentially, a chatbot is a computer program that conducts conversations with people, given their text or voice input, without any help from humans [1]. In the past, a traditional chatbot relied on rule-based approaches, by querying the input on a database and returning the corresponding answer if there is a match with a template utterance. With the development of Machine Learning/Deep Learning/Natural Language Processing, modern chatbots are equipped with State-of-the-Art language models, making them more **powerful in generating human-like conversation and be customizable by training on domain-specific data**. Because of that power and customizability, nowadays chatbots are used in various industries and use cases, ranging from commercial use such as customer care, and business Q&A, to social common goods such as public healthcare and psychological support [2].

Thus, with the future goal of making a mental health therapy chatbot for the public, in this project, we settle our first step toward the goal. We first choose an existing conversational dataset [3], analyze the data using Spark, then use it to build a conversational chatbot fueled by both rule-based and AI models. Finally, we deploy it on AWS EC2 instances as a web application (SaaS) for public use.

2. APPLICATION FLOW

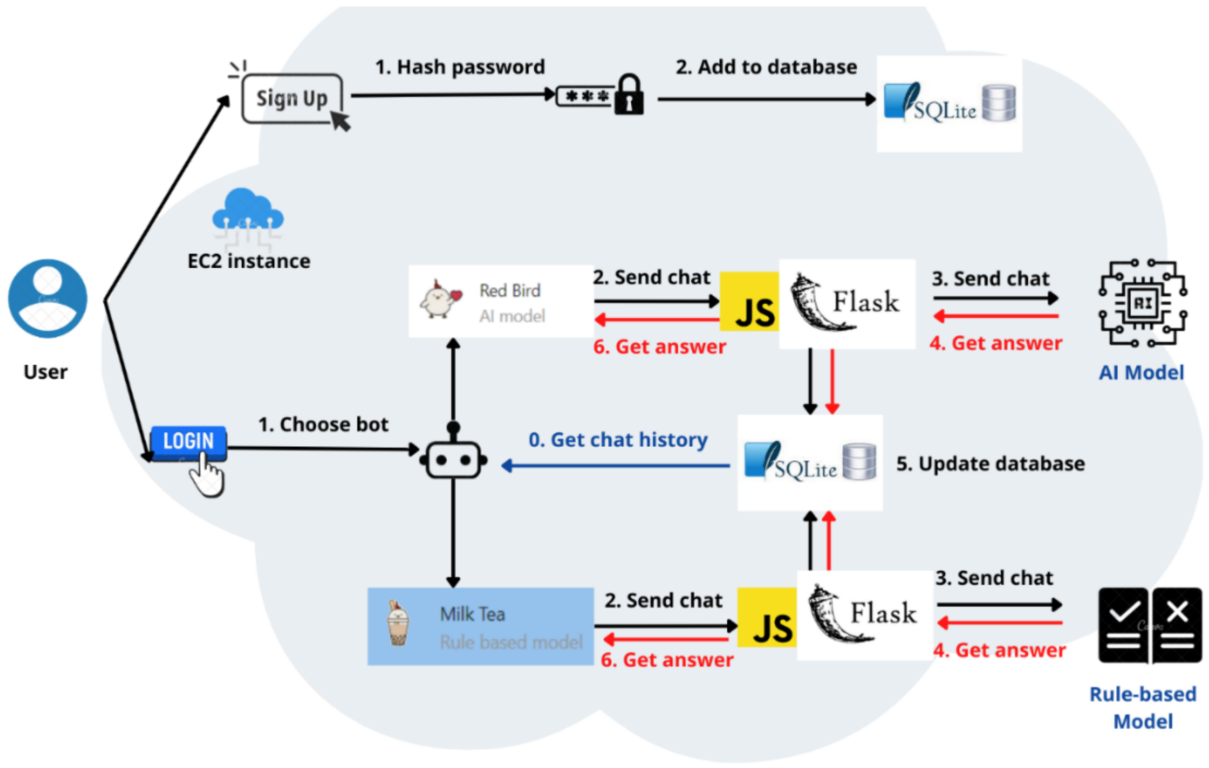


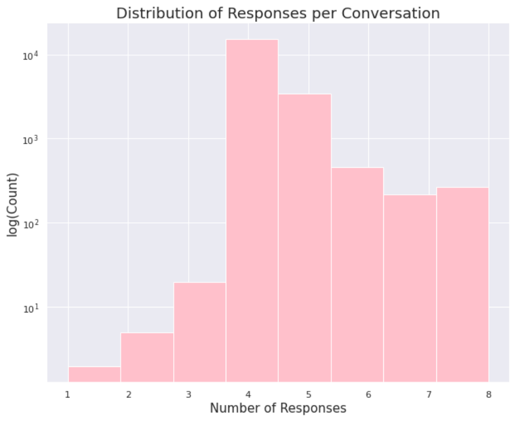
Figure 1. Our application flow

3. DATA ANALYSIS

We performed data analysis on Databricks with PySpark. The data is presented as a CSV with 84169 rows and 8 columns. Each row contains the text of an utterance, along with its information. The 8 columns are described below:

- **conv_id, utterance_idx, speaker_idx**: the IDs of Conversation, Utterance, and Speaker respectively
- **context**: an adjective describing the text
- **prompt**: a Prompt based on a given Context, with which the Speaker wrote the first sentence
- **utterance**: the text data of the Utterance
- **selfeval, tags**: other information that we don't use.

In general, most conversations have a few utterances (15136 out of 19533 conversations have 4 responses in figure 2).



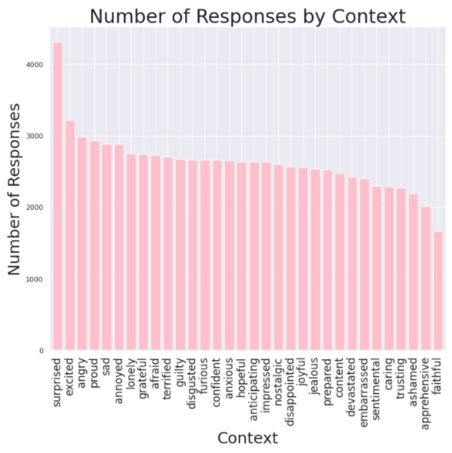


Figure 2. Conversations' lengths (left). No. responses for each context (right).

We can use **context** as the center of our analysis. There are 32 of them, ranging from 'suprised', 'excited', to 'faithful'. In figure 2, we can see that the numbers of responses are evenly distributed across all contexts, with the exception of "surprised", which seems to frequently have longer conversations.

There are 810 **speakers** in total, most of them are active in several contexts (578 speakers participate in <= 15 contexts), and some of them engage in almost all topics (115 speakers engage in >= 30 context)

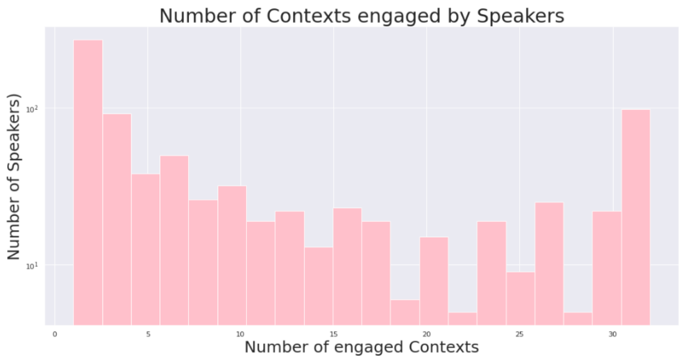


Figure 3. No. contexts engaged by speakers

For the textual data, in general, most **utterances** are short. The average length is 14 words, with 69146 out 84169 utterances are less than 20 words long.

4. MODEL

The core engine of our chatbot consists of two sub-models, a (Deep Learning) AI model and a rule-based model.

4.1. AI Model

The AI Model we chose is **DialoGPT**. The model is developed by Microsoft Research, finetuned from the base pre-trained language model GPT2. Inheriting attention mechanisms and the architecture of the Transformers model's decoder, GPT2 can first well encode the tree structure of textual data and then autoregressively generate the next words for the input text with a certain level of cohesion.

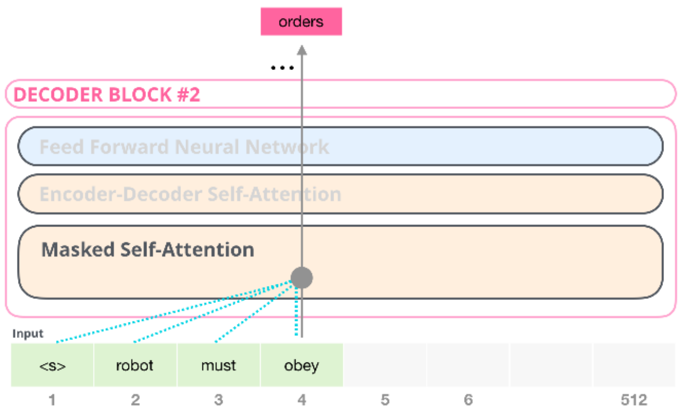


Figure 4. Autoregressive Text Generation by GPT2
(Source: https://jalammar.github.io/images/xlnet/transformer-decoder-block-self-attention-2.png)

However, GPT2 is pre-trained on conventional text data which often does not represent the way people orally interact and communicate. Thus, DialoGPT, to integrate a more conversationally interactive tone to the generation of GPT2, draws on 147M multi-turn conversations extracted from Reddit discussion threads. The evaluation of DialoGPT shows that responses generated by this model are diverse, contain information specific to the source prompt, and to some extent achieve the human-level quality in a single turn Turing test. That makes DialoGPT a suitable choice for our chatbot.

We plan to further finetune the model on the chosen dataset, however, due to the limited time and resources, we only use the checkpoint of DialoGPT-small provided by Huggingface Transformers [5]. **As suggested** in the generation pipeline of DialoGPT, we **include chat history along with the user's new input to generate the output**.

4.2. Rule-based Model

Besides the Language Model, we also employ a rule-based model, with the **database in the form of key "template input" and value "corresponding output" in a row**. The database is extracted from conversations in the chosen dataset and consists of 64k rows. When users give a new input, for each key we calculate a score that represents the similarity of the key (template input) and the new (actual) input, then we take top-5 keys with the highest score. After that, we sample 1 from 5 corresponding values of those keys, to diversify the model output.

In this project, instead of working with AIML (Artificial Intelligence Markup Language, an XML schema) database as traditional rule-based chatbots, we **utilize DataFrame for concurrent querying on the database**, **using the Pandas package in Python**. Also, we use **BLEU-3** (NLTK implementation) as the scorer, with a **hypothesis as the user's input and references as the chat history and template input**. This BLEU expression would yield a high score when n-grams of hypothesis frequently occur among references. Thus, in our case, template inputs with the top score will align but not need to be largely similar to the newest user's input, which is suitable to naturally develop the conversation. Also, the act of placing the chat history and template input on the same side of the BLEU scorer is to avoid top-score template inputs being too similar to one of the previous utterances, since they have been drawing from the same source.

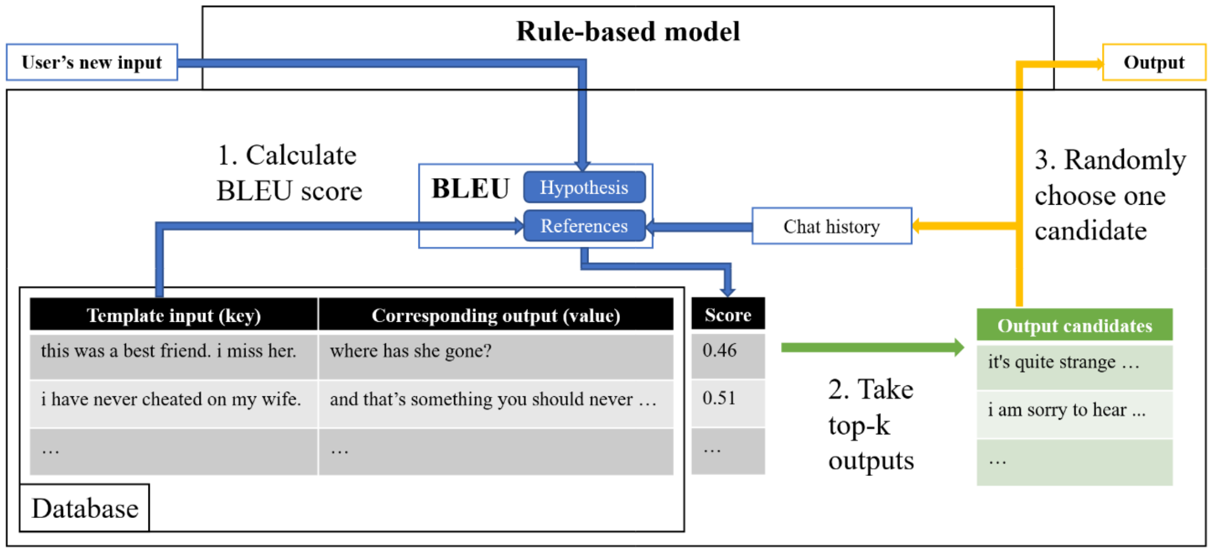


Figure 5. Anatomy of our rule-based model.

The **default setting** for both sub-models is using only the last bot output as the chat history. Our preliminary result shows that the response generated by both models still has a big gap with human response, especially in terms of content relevance, thus needs improving further.

5. WEB DEVELOPMENT

5.1. Database structure

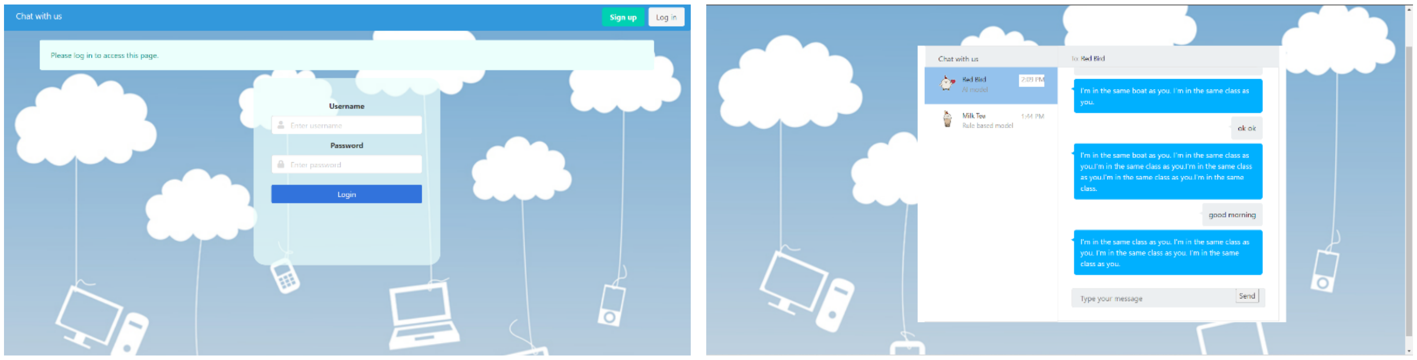
Our SQLite database contains two tables: users and chats. Users table has two attributes username and password. The username needs to be unique, and the password is hashed before it is stored in our database. This method increases the security of our server and prevents the hacker from accessing our source code and stealing the password of users.
The chats table has five attributes: content, create_at, user_id, message_from_bot, and with_ML_model. When users log in to their account, the backend will find all messages that have user_id same as the ID of the current user. The attribute with_ML_model is to differentiate the conversations with Red Bird (using the AI model) and Milk Tea (using the rule-based model). The attribute message_from_bot helps the frontend define whether chat from user or bot for betting presentation. The create_at attribute is the time that chats are created, therefore, when users log in to their account again, all their chats are displayed in the time order.

5.2. Backend Description

We built the backend using Flask for better communication with our ML model and rule-based model written in Python. We used the SQLite Relational Database type because we considered that our data is consistent and share the same structure, so SQL would run more effectively and save time when querying chat or username. The backend communicates with the frontend using two types of routers: auth for the user to login/sign up and chats for the user to send/get the message.

5.3. Frontend Design

We designed the frontend using a simplistic design with basic login and registration page and the chatroom page. It aims to give users a relaxed and fresh feeling when chatting with chatbots. Bootstrap and and bulma-css libraries are used.



I'm in the same boat as you. I'm in the same class as you.

ok ok

I'm in the same boat as you. I'm in the same class as you.I'm in the same class as you.I'm in the same class as you.I'm in the same class as you.I'm in the same class.

good morning

I'm in the same class as you. I'm in the same class as you. I'm in the same class as you. I'm in the same class as you.

Figure 6. Login and chatroom page

6. DEPLOYMENT AND EVALUATION

We provide **detailed instructions in our Github repository** to deploy our application on a local machine or an AWS EC2 instance. It is encouraged to install all dependencies in a new virtual environment. The web interface has been shown in section 5.

To evaluate our application, we considered two tests – response quality and response time – on different models and different EC2 instances. However, because the qualitative judgment on the bot's response requires human resources to proceed while it is less relevant to cloud computing than response time measurement, in this project we only focus on the response time. We programmatically record the response time of our chatbot on each configuration, then compare them to draw some insights. Notice that for the rule-based model, querying on the whole database would be time-consuming, thus we add an option to perform the query on a random part of the database (hence called "partial query").

Models / EC2 instances	AI	Rule-based	Rule-based (partial query, *)
t2.small	n/a	n/a	n/a
t3a.medium	0.7678	21.2373	1.9823
m5.large	0.7679	9.6870	1.0672
t3.medium (oregon)	0.9078	17.6857	1.8099

Table 1. Response time (in second) with respect to different configurations. *partial query on 5000 rows

We launch three instances of different sizes (small, medium, and large) in N. Virginia, and a medium instance in Oregon for comparison. Unfortunately, to install PyTorch which is necessary for our AI model, a computing instance needs to have more than 3GB of memory [6], thus we encounter the same installation error in t2.small and cannot deploy our app. For other instances, for each model type, we run three times and take the median to report in the result table. We observe that there is no significant difference in response time of two medium instances located in different regions. Rule-based models running on the large instance perform twice faster as expected since the large instance has twice more memory than the medium. However, the AI model's response time in both instances is approximately equal, suggesting that the workload of the AI model is roughly fixed, solely relied on CPU power, and does not benefit from larger memory.

7. CONCLUSION

In this project, we **successfully built** a chatbot application **and deployed** it on AWS EC2 instances for public use. There are several aspects that need our attention, especially the core model. **In the future**, we will **further finetune the AI model** for better and domain-specific generation and study a better scorer of the Rule-based model to select the most suitable output. Additionally, we may **employ "Stress Test"** [7], which is commonly used in industry, to test our system with thousands of concurrent connections or more. Nonetheless, we will try to utilize GPU power from AWS Elastic Inference for faster AI model inference [8].

All of the code and data are uploaded to this Github repository:
https://github.com/hkust-comp4651-22s/project-conversational-chatbot.

REFERENCE

[1] The History Of Chatbots - From ELIZA to ALEXA - Onlim Blog

[2] Karim the AI delivers psychological support to Syrian refugees | Artificial intelligence (AI) | The Guardian.

[3] https://github.com/facebookresearch/EmpatheticDialogues

[4] DialoGPT - Microsoft Research.

[5] microsoft/DialoGPT-medium · Hugging Face

[6] Cannot install pytorch on AWS Ubuntu 18.04 even with 3 GiB of ram - Ask Ubuntu

[7] Performing a Stress Test on Web Application? - Stack Overflow

[8] Reducing Machine Learning Inference Cost for PyTorch Models - AWS Online Tech Talks - YouTube