

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

Project Chat-Bot:

Information retrieval with concept ontology (NTUFOOD-BOT)

Lim Zhi Yang

Supervisor: Associate Professor Chng Eng Siong

Examiner:

School of Computer Science and Engineering 2018

Abstract

The proliferation of conversation agents is becoming prevalent due to the myriad of benefits it brings about. This has led to a revolutionised way of consumers interacting with machines. Thus, in this paper, the development of chatbot using Dialogflow as intents and queries extractor will be evaluated. The focus of this chatbot is on the parsing of complex queries and returning the total cost and summary of the food and drinks ordered back to the users in NTU North Spine Canteen A. In addition, information of the system's architecture and modules used to implement the system for the Front-end and Back-end will be discussed. Lastly, optimisation of the system of the whole process will be visited.

Acknowledgements

<To be added>

Contents

Acknowledgements	3
1. Introduction.....	6
1.1 Background	6
1.2 Aims and Objectives	8
1.3 Scope.....	8
1.4 Report Organisation	8
2. Literature Review.....	9
2.1 Overview of Chatbot.....	9
2.2 Types of Development.....	9
2.2.1 Rule-Based Approach Development.....	9
2.2.2 Machine Learning Approach Development	10
2.3 Comparison Between Various Chatbot Development Platforms	12
2.3.1 Dialogflow	15
2.4. Design and Implementation Considerations	18
2.4.1 Pre-Processing of Data	18
3. Proposed Methodology and System Specifications	20
3.1. Chatbot Development Platform	20
3.2 Chatbot Publishing Platform.....	21
3.3 System Architecture.....	21
3.4 Complex Queries.....	23
4. Implementation	25

4.1 Server-side	25
4.1.1 Environment Set-up	25
4.1.2 Server Setup and Routing Mechanisms	25
4.1.3 Program Control Flow	26
4.1.3 Named Entity Recognition (NER)	26
4.1.4 Hosting a Server for Spacy Model	32
4.1.5 Spell Checker Module	33
4.1.6 Parser for Food Information	35
4.2 Front-End	35
4.2.1 Facebook Messenger	35
4.3 Database	36
5. System Demonstration	37
5.1 Overview	37
6. Conclusions and Future Work	38
6.1. Conclusion	38
6.2. Future Work	38
References	39

1. Introduction

1.1 Background

With the rapid advancement of technology, automation technology has now restructured several business landscapes, supercharging performance for their operations [4]. This has brought about disruptive implications to businesses, transforming the way decisions are made and how processes can be optimised. Inevitably, businesses are pressured to make modifications to their business models or adopt entirely new models in view of the industrial revolution.

By leveraging on automation technologies, administrative and or repetitive matters for several processes over a wide range of domains can be automated [3]. This frees up financial and human resources that can be better allocated in other areas of operations which proposes revenue and profit gains. Recent findings have shown the integration of automation technology into business models, have led to increased competitiveness and more customized control for customers. This is evident in New York where check-out times are seen faster, and unexpected operational improvement are observed since 2008 with the presence of self-checkout kiosks in fast food restaurants [5]. Out of the multiple applications of AI technologies, chatbots are ranked the most popular in demand.

Chatbots are defined as computer programs or a service, powered by rules and sometimes artificial intelligence, that users interact via a chat interface. Chatbots provide human-like technology that mimic the way a human would converse with the user. Thus, Chatbots are deployed by businesses to answer user's enquiries for frequently asked questions as they are available 24 hours a day without requiring much overseeing by humans.

With the increased adoption of mobile Internet and messaging platforms in the society, Chatbots are usually deployed on mobile messaging platforms [6]. Leveraging on the mobile

messaging platforms, the extent of reach of the chatbots would be enormous. According to Business Insider (2016), approximately 3 billion people worldwide use mobile messaging applications such as Facebook Messenger, WeChat, Skype, Telegram, Slack, Viber, and Kik. For many users of these services, natural language is expected in online interactions, making automated processes using natural language a business opportunity with huge potential. Recognising this opportunity, global technology companies such as Facebook and Microsoft have offered tremendous support in providing resources for chatbot developments. As of 2016, Chatbots is seen growing rapidly with 11,000 Facebook Messenger chatbots being developed [1] and 80% of businesses are envisaged to have some sort of chatbot automation implemented by 2020 [2] with the initiatives by the technology giants.

The introduction of automation technology into the industries in particularly, Food and Beverages Industry, has revamped the way food and beverages are ordered. Many restaurants chains are seen jumping onto the bandwagon by utilising chatbots as one of their food ordering platform. This can be seen in a few restaurants such as Wingstop [8], Pizza Hut, Burger King, Taco Bell and Whole **Foods** Market [9]. After the implementation of the Chatbot, boost in profit and revenue can be observed in the restaurants [8]. As such, implementation of food ordering chatbots can be done in schools, to speed up the rate at which food and beverages are ordered. This will provide students with more time to eat their meals and lesser time spent on queueing for the food given that students have short breaks in between their lessons.

1.2 Aims and Objectives

Humans usually speak informally when they are ordering food. This makes their queries informal and possess different characteristics as compared to formal English language. Thus, the objective of this project is to develop a chatbot that can process both formal and informal queries pertaining to the ordering of food and drinks in Canteen A (North Spine).

1.3 Scope

This scope of this project covers the comparison of the performance between the various chatbot

development platforms, developing a full-stack prototype chatbot, how entities can be recognised and extracted based on user's queries, multiple platforms for the user interfaces (HTML, Facebook Messenger). Lastly, this project evaluates the speed of how fast replies can be processed and returned to the user.

1.4 Report Organisation

This report consists of 6 Chapters. The first chapter provides the background information of this Final Year Project. The second chapter discusses about various existing techniques and system designs for Chatbot Development. The third chapter explains the proposed solution and methodology. The fourth chapter discusses the implementation details of the system, followed by the elaboration on the system demonstration in the fifth chapter. Finally, the six chapter closes the report with conclusion and future works of this project.

2. Literature Review

This section covers the overview of how chatbot functions. In addition, the design and implementation options to it will be covered.

2.1 Overview of Chatbot

The main function of a Chatbot is to take in inputs from users and return an appropriate response back to them. Chatbots based its conversational ability on Natural Language Processing (NLP) and Machine Learning (ML) concepts.

2.2 Types of Development

A chatbot can be built in two ways, mainly rule-based approach and machine learning.

2.2.1 Rule-Based Approach Development

Rule-based approach requires the chatbots to adhere to certain specific rules and provide a dedicated response to the specific input. This approach is reflected in Fig.1, below.

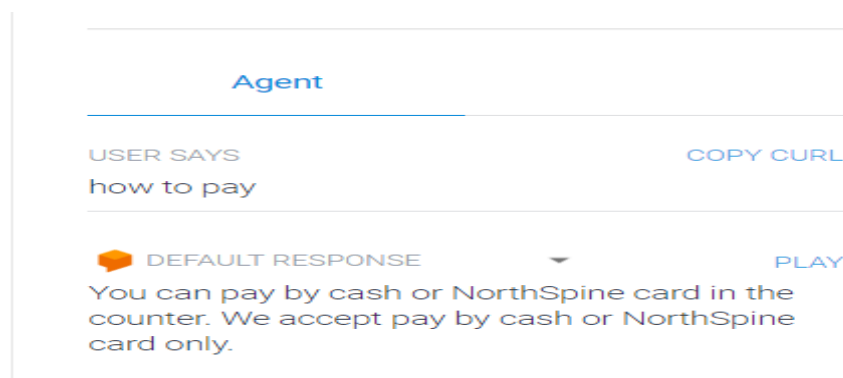


Fig. 1, Response to Rule-Based Approach

In the above diagram, it is observed that the user asked, “How to pay”. After receiving this input from the user, the chatbot automatically answer “You can pay by cash or NorthSpine card in the counter. We accept pay by cash or NorthSpine card only.”. This is a typical approach used in creating question-answer bots as seen in the above example since the chatbot is hardcoded to provide this specific answer to the specific input that the user queried. Using back the same example mentioned above, should the input changed to “How to pay now?” “The chatbot would mismatch incorrect intents to the input as it deems the input as a different data from what it was trained with. This arose due to a surplus of word input into the chatbot. Thus, inference can be drawn from it whereby should there be a lack or surplus of words input into the chatbot, the chatbot will not able to accurately allocate the correct intent to the input.

Rule-based chatbots requires minimal financial resources to implement and can be developed easily in a short period of time with minimal difficulty should the requirements be relatively simple. However, this type of approach renders the chatbot to not possess human intelligence since they cannot understand the intent or context of the queries by the users. This leads to the developer having to programme a myriad of answers to accommodate to the various ways of which questions can be asked, which is not feasible due to the infinite possibilities of how the queries can be phrased.

2.2.2 Machine Learning Approach Development

Machine learning approach allows chatbots to learn as they go rather than following specific hardcoded examples as discussed in section 2.1.1 of this report.

With the advancements in machine learning and natural language processing domains, algorithms pertaining to neural networks now allow the chatbots to draw valid assumptions based on the patterns of the large amount of data that the chatbot was trained with. After

which, the chatbot will map it to a concept such as a semantics, or the intent of the input.

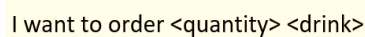
Take for example, the user queried:

A yellow rectangular box containing the text "USER SAYS" in a small, grey, sans-serif font, followed by "i want to order 1 bubble tea" in a larger, black, sans-serif font.

USER SAYS
i want to order 1 bubble tea

Fig. 2, Example of a User Query

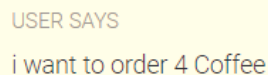
After being exposed to multiple queries of the same pattern as seen in Fig.2,

A yellow rectangular box containing the text "I want to order <quantity> <drink>" in a black, sans-serif font.

I want to order <quantity> <drink>

Fig. 3, Sentence Pattern of User Query

Upon future queries of similar pattern seen in Fig.3,

A yellow rectangular box containing the text "USER SAYS" in a small, grey, sans-serif font, followed by "i want to order 4 Coffee" in a larger, black, sans-serif font.

USER SAYS
i want to order 4 Coffee

Fig. 4, Example of a Different User Query

When the user queries the input in Fig.4, the chatbot will be able to deduce the user is trying to order 4 cups of the drink, Coffee, even if it previously do not know Coffee is a form of drink and 4 is a cardinal number.

Intuitively to humans, one can detect the pattern after exposing to multiple similar sentence structures. However, this is not the same for machines. Training a chatbot to recognise such patterns are time-consuming and labour intensive. Without exposing sufficient amount of sample sentences that comes along with labels within the sentences, there is this possibility of the machine misclassifying the structure of the sentence. Hence, there lies a need for the developer to come up with not ten or hundred samples but rather, ten-thousands, hundred-thousands or millions number of sentences that are varied in the way they are phrased. One

can imagine the amount of resources spent on labelling the sentences and for training the model of the chatbot just to ensure the accuracy of the pattern recognition. This type of approach is classified as a “Black Box” approach since the developers do not possess any knowledge of the internal workings of the chatbot and only can observe the inputs and outputs of it.

2.3 Comparison Between Various Chatbot Development Platforms

A chatbot entails a publishing and development platform. A chatbot publishing platform is the medium through which the user can access the user interface. It is usually on a webpage or through other messaging platforms such as Facebook Messenger, Slack, Telegram, Discord or Kik. Whereas a chatbot development platform is the tool that aids in the infusion of functions into the chatbot. Some of the examples of the chatbot development platform includes Microsoft bot Frameworks, Wit.ai, Dialogflow, IBM’s Watsons. These development platforms assist in creating intelligence in a chatbot by leveraging on ML and NLP concepts. A comparison of the various development platforms can be observed in the table below.

Bot Name	Features	Programming languages / Apps / Integration	Technical details	Project Link
IBM Watson Conversation Service	Built on a neural network (one billion Wikipedia words). Has three main components: Intents, Entities, Dialog	Node SDK Java SDK Python SDK iOS SDK Unity SDK	-	https://www.ibm.com/watson/ai-assistant/
wit.ai	Allows to use: Entities	Node.js client Python client	Is available for developers to use	https://wit.ai/

	<p>Intents</p> <p>Context</p> <p>Actions</p> <p>Natural Language Process (NLP)</p>	<p>Ruby client</p> <p>On other platforms:</p> <p>HTTP API</p>	<p>with iOS, Android, Windows Phone, Raspberry Pi, Python, C and Rust.</p> <p>JavaScript plugin.</p>	
rasa NLU	<p>intent classification</p> <p>entity extraction</p>	<p>HTTP api</p> <p>Python</p>	<p>You can use rasa as a drop-in replacement for wit, LUIS, or Dialogflow</p>	<p>https://rasa.com/</p>
Dialogflow	<p>DIALOGFLOW matches the query to the most suitable intent based on information contained in the intent (examples, entities used for annotations, contexts, parameters, events) and the agent's machine learning model.</p> <p>DIALOGFLOW transforms the query text into actionable data and returns output data as a JSON response object.</p>	<p>SDKs:</p> <p>Android</p> <p>iOS</p> <p>Cordova</p> <p>HTML</p> <p>JavaScript</p> <p>Node.js</p> <p>.NET</p> <p>Unity</p> <p>Xamarin</p> <p>C++</p> <p>Python</p> <p>Ruby</p> <p>PHP (community supported)</p> <p>Epson Moverio</p> <p>Botkit</p> <p>Java</p>	-	<p>https://dialogflow.com/</p>

	Leverage predefined knowledge packages collected over several years.			
Microsoft Bot Framework	<p>Understands the user's intent.</p> <p>To give your bot more human-like senses, you can incorporate LUIS for natural language understanding, Cortana for voice, and the Bing APIs for search.</p>	<p>Bot Builder SDK (.NET SDK and Node.js SDK.)</p> <p>Bot Connector</p> <p>Developer Portal</p> <p>Bot Directory</p>	<p>The framework provides the Direct Line REST API, which you can use to host your bot in an app or website.</p>	<p>https://docs.microsoft.com/en-us/azure/bot-service/?view=azure-bot-service-3.0</p>
Microsoft Language Understanding Intelligent Service (LUIS)	<p>Uses intents and entities.</p> <p>All LUIS applications are centered around a domain-specific topic or content related.</p> <p>Active learning.</p> <p>You can use pre-existing, world-class, pre-built</p>	<p>C# SDK</p> <p>Python SDK</p> <p>Node JS SDK</p> <p>Android SDK</p>	<p>LUIS offers a set of programmatic REST APIs that can be used by developers to automate the application creation process.</p>	<p>https://www.luis.ai</p>

	models from Bing and Cortana. Deploy models to an HTTP endpoint with one click. LUIS returns easy-to- use JSON.			
--	---	--	--	--

As there are multiple chatbot development platforms available in the market, this report will only be focusing on Dialogflow, as that is the platform of which the proposed solution of this report was developed in. For more information on the comparison of the Natural Language Understanding services, one can refer to the project links stated in the table above or, refer to an article on the evaluation of services for conversational question answering system [10].

2.3.1 Dialogflow

Dialogflow is previously known as Api.ai and is a platform catered for intent-based chatbots. Dialogflow supports more languages than Microsoft Language Understanding Intelligent Service (LUIS), RASA NLU and IBM Watsons Conversation Service. Dialogflow would match the user's query based on the intents and the machine learning model it was trained with. To accurately match to the correct intent, Dialogflow would consider the entities labelled annotation, context, parameters and events as seen in Fig. 5 before deciding which intent it would classify the input under. Dialogflow takes in the input as string format and transform this data into JSON response object upon output as reflected in Fig. 6.

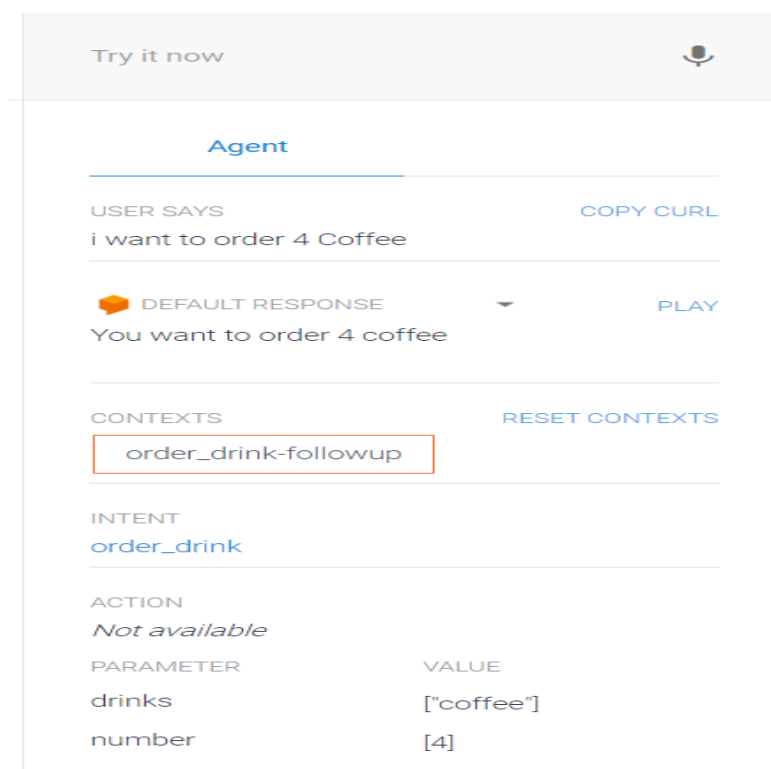


Fig. 5, An Example of Values Identified in Dialogflow

Diagnostic info

```
41  }
42  },
43  "outputContexts": [
44  {
45    "name": "projects/foodbotntu/agent/sessions/3eddef91-7c52-b086-e7d3-505f6790474b/contexts
46    /order_drink-followup",
47    "lifespanCount": 2,
48    "parameters": {
49      "drinks": [
50        "coffee"
51      ],
52      "drinks.original": "Coffee",
53      "number": [
54        4
55      ],
56      "number.original": "4"
57    }
58  },
59  ],
60  "intent": {
61    "name": "projects/foodbotntu/agent/intents/580ecbfd-10e5-4603-95dd-70a2da925546",
62    "displayName": "order_drink"
63  },
64  "intentDetectionConfidence": 0.59,
65  "diagnosticInfo": {
66    "webhook_latency_ms": 83
67  },
68  "languageCode": "en"
69  },
70  "webhookStatus": {
71    "code": 5,
72    "message": "Webhook call failed. Error: 404 Not Found"
73  }
74  }
```

Fig. 6, Diagnostic Info of Dialogflow

It is worthy to take note of way Dialogflow presents the data to the user in the user interface.

The user experience will diminish given the current version of the Dialogflow since

Dialogflow is not able to format its output with line breaks when integrated on webpage with its web demo settings. This can be observed in Fig. 7 where the responses are not aligned.

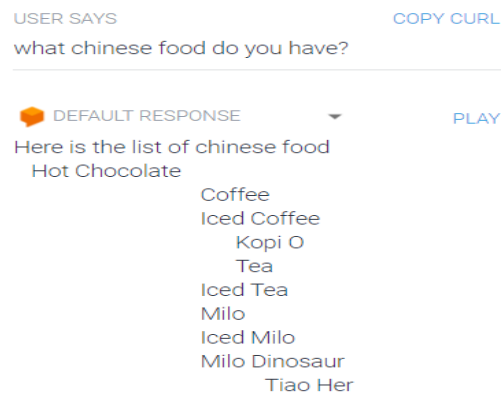


Fig. 7, Misalignment of spacing

2.4. Design and Implementation Considerations

With reference to section 2.2, conclusion about the limitations of the different development techniques can be made. Particularly, a lot of improvements can still be made for the areas of NLP and ML as many matters remains ambiguous. Both approaches require intensive manual labour to come up with tons of data to facilitate in training the model of the chatbot. The situation is further exacerbated by the fact of having to allocate resources to label the entities in the data for the ML approach. On top of that, both processes will need a long duration of time to train the model since the training of the model is based on a neural network approach. To heighten the accuracy of detection of entities, a few processing can be done to the input data.

2.4.1 Pre-Processing of Data

Data will come in “dirty” upon input from the users. Thus, to achieve the highest efficiency and accuracy for information extraction, data cleaning can be done. Data cleaning done aptly will help to significantly reduce the data dimensionality of term space. This translates to better information extraction efficiency as there are lesser data for processing for intent classification.

2.4.1.1 Case of Data

To facilitate in the data retrieval from databases like SQL server, there is a need to pay attention to the case of the data in whether it is in upper or lower case. This is important since databases like SQL server, are case-sensitivity. Hence, to avoid error from arising upon data retrieval, all the data can be changed to lower case before passing on to the databases.

2.4.1.2 Special Characters, Punctuations, Symbols and Multiple Spaces

Due to the fact that users may accidentally type or insert special characters, symbols, punctuations and multiple spaces into the query, such regular expressions are removed. This will ensure the query only contains information that are vital to the recognition of the entity types.

2.4.1.3 Word forms of numbers

Numbers exist in integer and word forms. This can be confusing to the machine as the machine may perceive the word forms of numbers to be a normal English word instead of as a quantity that the user wishes to use for his or her order. Thus, a dictionary of numbers was created that maps the word forms of the numbers to the integer value of the numbers. This will help to transform all the word forms of numbers to integers values which can be readily used later for reflecting the quantity of the food.

2.4.1.4 Singularity of the Input Item

To ensure the correct named entities is recognised by the chatbot's model, the singularity of the input should be consistent where all plural nouns should be changed to singular nouns. This hence eliminate any ambiguity of any potential wrong entities connotated with the correct noun.

2.4.1.5 Stopwords

Stopwords are frequently used words by humans. It comprises of examples such as, "I", "and", "you", "he". Such words prove not to be useful in the analysis of data since they do not add on to the uniqueness of the pattern of the data. Thus, these words could be removed to avoid redundant computations for intent classification. To suit the topic of food ordering, a customised list of stopwords was utilised. This list of stopwords contains the common stopwords like, "I", "and", "you", "he", "she", "on" as mentioned above and, common terms used in food ordering like "plates", "cup", "wish", "want" and 'order'.

2.4.1.6 Stemming and Lemmatisation

Stemming refers to the crude heuristic process that removes the ends of words to reduce the word to its stem or base form. The rules on how to remove the ends of words includes following rules that arise from derivational affixes. Stemming might reduce the inflected word to illogical forms that do not exist in the dictionary.

Lemmatization is the process of removing inflectional endings to return the base or dictionary form of a word, which is known as the lemma. If confronted with the token “ponies”, stemming will reduce the token to “poni” whereas lemmatisation will reduce the word to “pony”, which is the singular form of ponies.

A conclusion of lemmatisation taking into consideration the context of the sentence can be drawn. Similarly, a conclusion of stemming not considering the context of the sentence can be concluded as the word form after the process confirmed the hypothesis.

2.4.1.7 Spelling Correction

Lastly, any misspelled words should be replaced with a correct word. Online dictionary could be applied here to increase the replacement success rate.

3. Proposed Methodology and System Specifications

3.1. Chatbot Development Platform

In this project, Dialogflow is chosen as the chatbot development platform. Dialogflow is being chosen as the development platform since it is intelligent enough to understand what the user is referring to with its internal mechanism that permutes the query to correctly map the intents.

In addition, it supports an array of messaging platforms for deployment. As of now, it supports 16 platforms which allows an extensive amount of people to be reached easily. This

feature is further commendable with its one-click integration, which eases the job of the chatbot developer.

Lastly, it has built-in analytics tool that allow the chatbot owner to track the performance of the chatbot. Its integrated analytics tool can showcase usage patterns, latency issues, high and low-performing intents. This provides chatbots owners to be able to debug and improve the performance of their chatbot with minimal effort.

3.2 Chatbot Publishing Platform

With the large number of consumers utilising the Facebook Messenger application, Facebook Messenger is chosen as the publishing platform for the chatbot. This helps users to be able to access the chatbot easily and enhances the interaction with Facebook. On top of that, it will allow smooth integration with Dialogflow since Facebook Messenger is one of the listed messaging platforms that was supported.

3.3 System Architecture

The system comprises of 5 component layers. Fig. 8 shows the system architecture diagram which is, a conceptual model that depicts the structure, behaviour and relationship between the components of the system. The first layer is the User Interface layer, which is the front end where users will be able to interact with the chatbot by clicking and inserting inputs. Outputs back to the users will be reflected in this layer as well.

The next layer is the abstraction layer, Dialogflow, which is the interface that contains the data required for the next layer, the business layer. At the abstraction layer, the queries of the users and intents will be extracted at this layer and passed into the next layer which is the Python server.

The Python server is the business logic layer that contains the algorithms responsible for the parsing of the data from the abstraction layer. The data is passed from the abstract layer to the business logic layer by webhooking into the server with intents identified in Dialogflow.

Depending on the intents being identified, the next layer, the dependency layer, may or may not be called. The dependency layer contains Spacy libraries and API that does processing for queries that are classified as complex queries, which will be mentioned in the section 3.4 of this report. The dependency layer will only be called upon when the intent identified in Dialogflow is “Complexquery”. Should the dependency layer be called, a JSON object will be passed back to the server (business logic layer). Following which, the server will send a query request to the last layer, the persistence layer.

Upon receiving the query from the business logic layer, data as requested in the query will be retrieved from the MySQL database. This retrieved data would be passed back to the server, subsequently, Dialogflow and ultimately, Facebook Messenger.

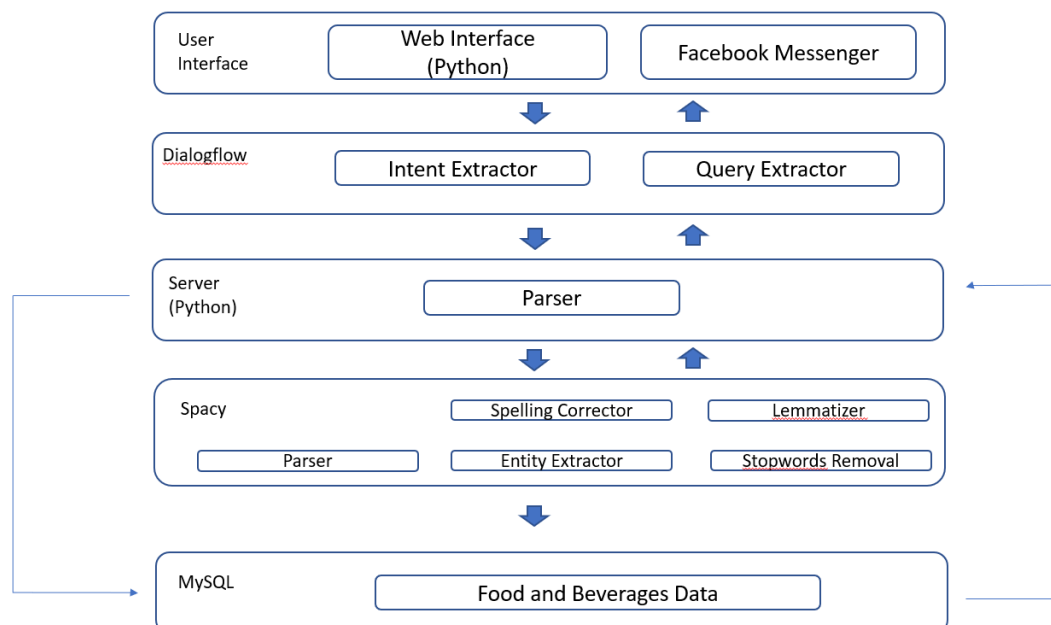


Fig. 8, System Architecture Diagram

3.4 Complex Queries

The focus of this project is on handling complex queries that is queried by the users. Complex queries are not simple queries that makes sense to the machine. Instead, complex queries are queries that can possibly contain special symbols or characters, Upper and lower-case text within a sentence, multiple white spaces, punctuations and multiple customised order of food. An example of a simple query can be seen in Fig. 9.

Simple Query:

"I want 2 plates of large Chicken Rice"

Fig. 9, An Example of Simple Query by User

On the other hand, complex queries can be of any of the following below in Fig.10.

Complex Queries:

e.g. "I want 2 small chicken rice, 3 medium fishball noodles and 4 large spaghetti"

e.g. "I want 2 plates of small chicken rice, 3 bowls of medium fishball noodles and 4 large plates of spaghetti"

e.g. "I want 2 small chicken rice, 3 medium fishball noodles and 4 large spaghetti"

e.g. "I want 2 !@@## small chicken rice, 3 !@#\$\$% medium fishball noodles and 4 l%^arge spag!@hetti!"

e.g. "2 small chicken rice, 3 medium fishball noodles, 4 large spaghetti"

e.g. "2 small chicken rice 3 medium fishball noodles 4 large spaghetti"

Fig. 10, Examples of Complex Queries by Users

It is highly possible for humans to send a complex query to the chatbot since they may misspell some of the words or even accidentally pressed other alphabets, symbols, characters or spacing. It is not possible to know exactly what sentences the users would send to the

chatbot due to the inherent incomprehensible nature of humans. Taking into consideration this point, a solution is formulated to tackle this issue.

4. Implementation

4.1 Server-side

The server side is implemented using Python. Python is a powerful high-level, object-oriented programming language used since 1991. Python utilises human readable language which makes it easy to develop and debug the program. Python developers do not have to worry about the conversion to machine language since python has in-built functionalities to automatically convert the high level interpreted language to machine language for them. This thus saves on memory resources for the python scripts. In addition, its object-oriented feature allows us to solve complex problems easily by creating objects to divide the complex problems into smaller manageable sets. With such appealing features, python proves to be a great language for developing servers.

4.1.1 Environment Set-up

To enable webhook feature in Dialogflow, the server need to be ran with a secured connection, HTTPS and the URL must be publicly accessible. To fulfil both requirements, domain name and SSL certificates are necessary for setting up a public server. Hence, an alternative to this would be to use Ngrok. Ngrok exposes local servers behind firewalls to the public internet over secured tunnels. This creates a public HTTPS URL for a website instantly by running the program on our local developing machine.

4.1.2 Server Setup and Routing Mechanisms

Creating a server in Python is relatively simple as Python has a framework called Flask. Flask is a relatively new framework that was released in recent years, 2010. Flask can be easily installed with Python Package Index (PPI). Flask application comes with an app.py, that is an instance of Flask and the application's root. An HTTP server will be established upon running the "flask run" command in the command line. This server is created based on the object defined in app.py. The setting up of the routing is done in app.py as well where

@app.route decorator specifies the points of interactions the server has with external applications.

<to be added after implementation>

4.1.3 Program Control Flow

Fig. 11 depicts the control flow diagram of the server-side implementation.

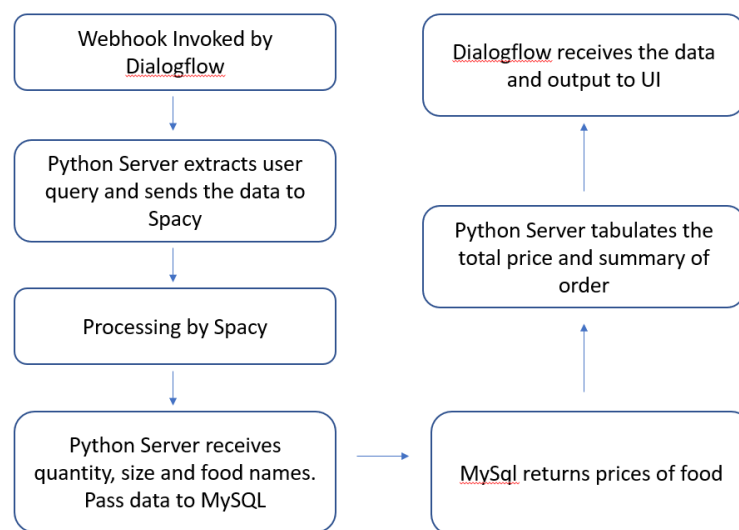


Fig. 11, Control Flow Diagram for Server

4.1.3 Named Entity Recognition (NER)

Named Entity Recognition is a process that annotates entities for a sequence of words. One of the state of the art NER model includes Spacy. Spacy is an exceptionally efficient system for NER in Python as it can recognise a wide range of named or numerical entities, which includes food names, quantity and size. In addition to this, it provides the ability to add on new arbitrary classes to the existing trained model.

Spacy outperforms other state of the art models in many domains. For NER, Spacy has a higher F-Score than Stanford NER model. This translates to better accuracy when recognising entities from sentences [11]. At the same time, Spacy significantly outperforms NLTK in word tokenization and

Part-of-Speech (POS) tagging as evident in Fig. 12 due to the different algorithms for both models. [12].

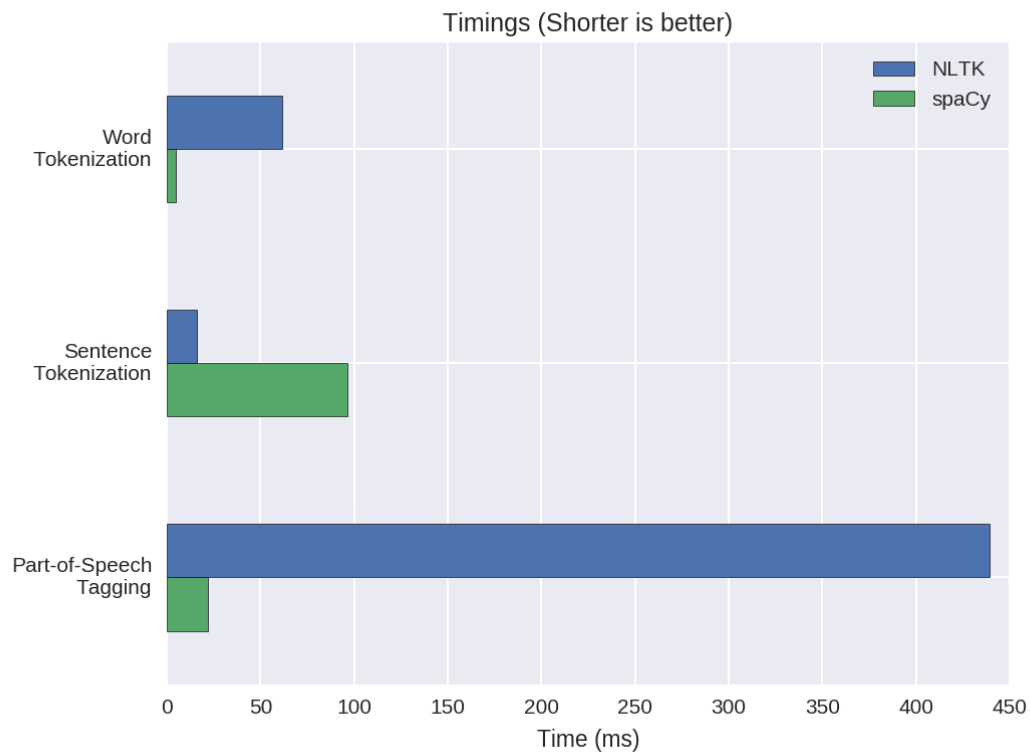


Fig. 12, Comparison of Performance Between NLTK and Spacy

4.1.3.1 Training Data for NER

To begin the process of using Spacy to train the model for NER, the training data for the model have to be manually created and labelled. An example of the training data for the model can be seen in Fig. 13.

```

def fooddatatraining():
    for i in food:
        x = i
        TRAIN_DATA = (
            (x + " is a cuisine", {
                'entities': [(0, len(x), 'FOOD')]
            }),
            ("2 " + x, {
                'entities': [(0, 1, 'QUANTITY'),
                             (2, 2 + len(x), 'FOOD')]
            }),
            ("4 large " + x, {
                'entities': [(0, 1, 'QUANTITY'),
                             (2, 7, 'SIZE'),
                             (8, 8 + len(x), 'FOOD')]
            }),
            ("3 " + x + " 3 bak kut teh", {
                'entities': [(0, 1, 'QUANTITY'),
                             (2, 2 + len(x), 'FOOD'),
                             (len(x) + 3, len(x) + 4, 'QUANTITY'),
                             (len(x) + 5, len(x) + 16, 'FOOD')]
            }),
            ("3 medium " + x + " 5 small chicken rice", {
                'entities': [(0, 1, 'QUANTITY'),
                             (2, 8, 'SIZE'),
                             (9, 9 + len(x), 'FOOD'),
                             (len(x) + 10, len(x) + 11, 'QUANTITY'),
                             (len(x) + 12, len(x) + 17, 'SIZE'),
                             (len(x) + 18, len(x) + 30, 'FOOD')]
            }),
            ("2 small " + x + " 4 large biryani 5 large chicken rice", {
                'entities': [(0, 1, 'QUANTITY'),
                             (2, 7, 'SIZE'),
                             (8, 8 + len(x), 'FOOD'),
                             (9 + len(x), 10 + len(x), 'QUANTITY'),
                             (11 + len(x), 16 + len(x), 'SIZE'),
                             (17 + len(x), 24 + len(x), 'FOOD'),
                             (25 + len(x), 26 + len(x), 'QUANTITY')]
            })
        )

```

Fig. 13, Portion of Code used for Training Food Items

Due to the fact that many Singapore delicacies are not named after dictionary English words such as Singapore famous delicacies, “Char Kway Teow”, “Laksa” and “Bak Kut Teh”, there lies a need to create a NER model from scratch. To achieve a high accuracy of the NER, millions of data have to be constructed and used to train this model. However, due to resources constraints such as insufficient memory size, lack of resources and long training time, the current model was trained using 1047 number of sentences which leads to a lower but, acceptable accuracy of the Spacy model. The number 1047, is the total number of sentences used for training the model over 26 epochs. This number is derived from looping the items contained in each category as seen in Fig. 14 into the different sentence patterns seen in Fig. 15.

food = ("char kway teow", "biryani", "oyster omelette", "chicken rice", "pasta", "satay", "hokkien prawn mee", "fried carrot cake",
 "hokkien char mee", "curry fish head", "noodle", "spaghetti with sausage", "beef", "rojak", "aglio olio chicken cutlet",
 "roti prata", "spaghetti with chicken chop", "aglio olio chicken chop", "bee hoon", "spaghetti with chicken cutlet",
 "bun bo hue", "spaghetti with ham and mushroom", "pho", "mee siam", "dim sum", "wanton mee", "pho bo", "laksa", "chicken burger",
 "bak chor mee", "crabs", "nasi lemak", "fish burger", "bak kut teh", "fried hokkien mee", "fish n chips")

drinks = ("milk tea", "lao hor", "soy bean", "coffee", "kopi", "beer gao", "milo", "tea", "teh", "teh bing", "milo bing",
 "iced milo", "iced coffee", "iced milk tea")

sides = ("fries", "mash potato", "toast cube", "salad", "soup", "spaghetti")

Fig. 14, Categories of Items

- 1) <ITEM>
- 2) <QUANTITY><ITEM>
- 3) <QUANTITY><SIZE><ITEM>
- 4) <QUANTITY><ITEM><QUANTITY><ITEM>
- 5) <QUANTITY><SIZE><ITEM><QUANTITY><SIZE><ITEM>
- 6) <QUANTITY><SIZE><ITEM><QUANTITY><SIZE><ITEM><QUANTITY><SIZE><ITEM>

Fig. 15. Sentence Patterns for training the data

The spacy model is trained based on a combination of different sentence patterns as seen in Fig. 15. For items that are annotated with "ITEM" as part of the pattern, it means all the items are trained regardless of whether it is a food, drink or side dish with the possible permutations. Elaborating on that, the third pattern in Fig. 15 can be "1 small milk tea" which has sentence pattern of same entity type <QUANTITY><SIZE><DRINKS>. However, using a single sentence pattern alone may prove to be unrealistic since it is not a good representation of what actual examples may look like. Thus, patterns that incorporate other entities types were included in the training process. Some of the patterns that consist of the different entities types can also be observed in Fig. 15.

One of the sentence pattern that varies in terms of characteristics from the third sentence pattern would be the sixth pattern. The sixth sentence pattern consists of a hybrid of different entity types with the example of "2 small chicken rice 4 large coffee 5 medium fries" which follows

<QUANTITY><SIZE><FOOD><QUANTITY><SIZE><DRINKS><QUANTITY><SIZE><SIDES>. This pattern is evident in Fig. 16, which shows a clear demonstration of what was being described.

```
("1 " + x + " 3 soup 4 biryani 5 kopi", {
    'entities': [(0, 1, 'QUANTITY'),
                 (2, 2+len(x), 'FOOD'),
                 (len(x) + 3, len(x) + 4, 'QUANTITY'),
                 (len(x) + 5, len(x) + 9, 'SIDES'),
                 (len(x) + 10, len(x) + 11, 'QUANTITY'),
                 (len(x) + 12, len(x) + 19, 'FOOD'),
                 (len(x) + 20, len(x) + 21, 'QUANTITY'),
                 (len(x) + 22, len(x) + 26, 'DRINKS')]
}),

("2 small " + x + " 4 large coffee 5 medium fries", {
    'entities': [(0, 1, 'QUANTITY'),
                 (2, 7, 'SIZE'),
                 (8, 8+len(x), 'FOOD'),
                 (9+len(x), 10+len(x), 'QUANTITY'),
                 (11+len(x), 16+len(x), 'SIZE'),
                 (17+len(x), 23+len(x), 'DRINKS'),
                 (24+len(x), 25+len(x), 'QUANTITY'),
                 (26+len(x), 32+len(x), 'SIZE'),
                 (33+len(x), 38+len(x), 'SIDES')]
}),
```

Fig. 16, patterns with different entity types

With the inclusion of the other entities types, the model would be more accurate as it is able to handle more unpredictable inputs from the users. This will hence prepare the model to be readier for the different possible ways the users may phrase their food orders.

To further elaborate on the training process, reference to Fig. 13 can be made. Fig. 13 shows the usage of a string with a variable “x”, for training. In this case, the variable “x” denotes the name of the item. There are 3 main categories of item being ordered when consumers buy their food. The 3 main categories are Food, Drinks and Sides. Thus, to achieve acceptable volume for the training, “x” was substituted with different item names for the same sentence pattern as seen in Fig. 13. In Fig. 13, variable “x” is the name of the item found in the “Food” category. When we are doing the training for the food data, each food item is substituted into each sentence pattern. Upon finishing the substitution for all the food names for that sentence pattern, this whole substitution process will be done the same for the other remaining types of sentence patterns as observed in Fig. 15.

Similarly, the substituting process will apply to the other 2 categories, drinks and sides as well. Fig.

17 is a demonstration of how some of the drinks data were being trained and Fig. 18 of how some of the sides data were being trained.

```
def drinksdatatraining():
    for i in drinks:
        x = i
        TRAIN_DATA = (
            (x + " taste good", {
                'entities': [(0, len(x), 'DRINKS')]
            }),
            ("1 " + x, {
                'entities': [(0, 1, 'QUANTITY'),
                           (2, 2 + len(x), 'DRINKS')]
            }),
            ("2 medium " + x + " please", {
                'entities': [(0, 1, 'QUANTITY'),
                           (2, 8, 'SIZE'),
                           (9, 9 + len(x), 'DRINKS')]
            }),
            ("1 small " + x + " 6 large spaghetti chicken chop", {
                'entities': [(0, 1, 'QUANTITY'),
                           (2, 7, 'SIZE'),
                           (8, 8 + len(x), 'DRINKS'),
                           (9 + len(x), 10 + len(x), 'QUANTITY'),
                           (11 + len(x), 16 + len(x), 'SIZE'),
                           (17 + len(x), 39 + len(x), 'FOOD')]
            }),
            ("1 small " + x + " 3 large coffee 5 medium soy bean", {
                'entities': [(0, 1, 'QUANTITY'),
                           (2, 7, 'SIZE'),
                           (8, 8 + len(x), 'DRINKS'),
                           (9 + len(x), 10 + len(x), 'QUANTITY'),
                           (11 + len(x), 16 + len(x), 'SIZE'),
                           (17 + len(x), 23 + len(x), 'DRINKS'),
                           (24 + len(x), 25 + len(x), 'QUANTITY'),
                           (26 + len(x), 32 + len(x), 'SIZE'),
                           (33 + len(x), 41 + len(x), 'DRINKS')]
            }),
            ("3 medium " + x + " 5 small milo", {
                'entities': [(0, 1, 'QUANTITY'),
                           (2, 8, 'SIZE'),
                           (9, 9 + len(x), 'DRINKS'),
                           (len(x) + 10, len(x) + 11, 'QUANTITY'),
                           (len(x) + 12, len(x) + 17, 'SIZE'),
```

Fig. 17, Training Food Data

```

def sidesdatatraining():
    for i in sides:
        x = i
        TRAIN_DATA = (
            (x + " taste good", {
                'entities': [(0, len(x), 'SIDES')]
            }),
            ("3 " + x, {
                'entities': [(0, 1, 'QUANTITY'),
                             (2, 2 + len(x), 'SIDES')]
            }),
            ("1 small " + x + " please", {
                'entities': [(0, 1, 'QUANTITY'),
                             (2, 7, 'SIZE'),
                             (8, 8+len(x), 'SIDES')]
            }),
            ("two large spaghetti 4 small fries 5 large mash potato", {
                'entities': [(0, 3, 'QUANTITY'),
                             (4, 9, 'SIZE'),
                             (10, 19, 'SIDES'),
                             (20, 21, 'QUANTITY'),
                             (22, 27, 'SIZE'),
                             (28, 33, 'SIDES'),
                             (34, 35, 'QUANTITY'),
                             (35, 41, 'SIZE'),
                             (41, 53, 'SIDES')]
            }),
            ("2 large " + x, {
                'entities': [(0, 1, 'QUANTITY'),
                             (2, 7, 'SIZE'),
                             (8, 8 + len(x), 'SIDES')]
            }),
            ("8 " + x + " 9 salad", {
                'entities': [(0, 1, 'QUANTITY'),
                             (2, 2 + len(x), 'SIDES'),
                             (len(x) + 3, len(x) + 4, 'QUANTITY'),
                             (len(x) + 5, len(x) + 10, 'SIDES')]
            }),
            ("3 small " + x + " 5 medium toast cube 9 large lao hor", {
                'entities': [(0, 1, 'QUANTITY'),

```

Fig. 18, Training Drinks Data

4.1.4 Hosting a Server for Spacy Model

Upon testing the trained Spacy model, the result returned was great, format is correct but there is a huge flaw when loading this model. This model requires a long duration of 18 seconds to load, process the query and return the data. Naturally, this will not be acceptable by any users as no user would want to wait 18 seconds for a reply. This flaw dramatically decreases the user experience which will result to low user satisfaction. Thus, to tackle this bottleneck problem, Spacy Server API can be used. The Spacy Server API loads the Spacy model in a local hosted server in a separate terminal process. This API keeps the model constantly online listening with a server and client protocol. Therefore, the model is continuously receptive to inputs and do not have to be loaded

with each run. The server-client solution for the data communication is based on the Remote Procedure Call-based Services (RPC).

Caching is done on top of the call-based services using the Least Recently Used (LRU) scheme. This exponentially decreases the loading speed as seen in Fig. 19 with loading speed decreasing from 18 seconds to approximately 0.0479 seconds. This implies the improvement of Spacy's performance by 429 times.

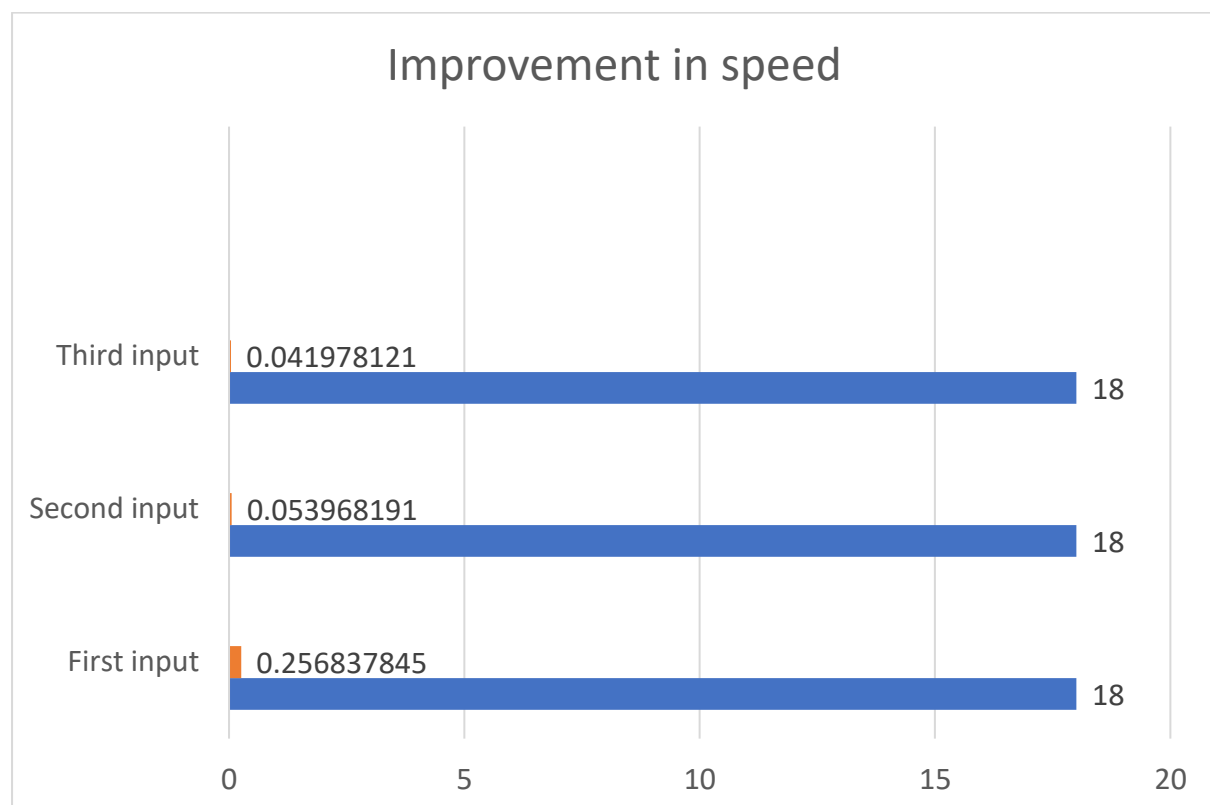


Fig. 19, Comparison of old run time with new run time after using spacy api

4.1.5 Spell Checker Module

A spelling checker or auto correct module is implemented on the server side to pre-process the user's queries. SymSpell library is used for this implementation instead of the well known PyEnchant since PyEnchant is not being updated anymore. SymSpell is a spelling correction module that uses Symmetric Delete spelling correction algorithm. This algorithm, opposite to other algorithms, only requires deletes and not transposes, replaces and inserts. Transposes, replaces and inserts of the

input term are transformed into deletes of the dictionary term. Replaces and inserts are expensive computation operations and language dependent: e.g. Chinese has 70,000 Unicode Han characters. Thus, the fast speed of this library comes from the inexpensive delete-only edit candidate generation and pre-calculation. Based on Fig.18, SymSpell is 1,870 times faster than BK-tree with the following conditions (dictionary size=500,000, maximum edit distance=3, query terms with random edit-distance from 0 to maximum edit distance, verbose=0).

It is 1 million times faster than Norvig's algorithm when ran with the following conditions (dictionary size=29,157, maximum edit distance=3, query terms with fixed edit distance = maximum edit distance, verbose=0).

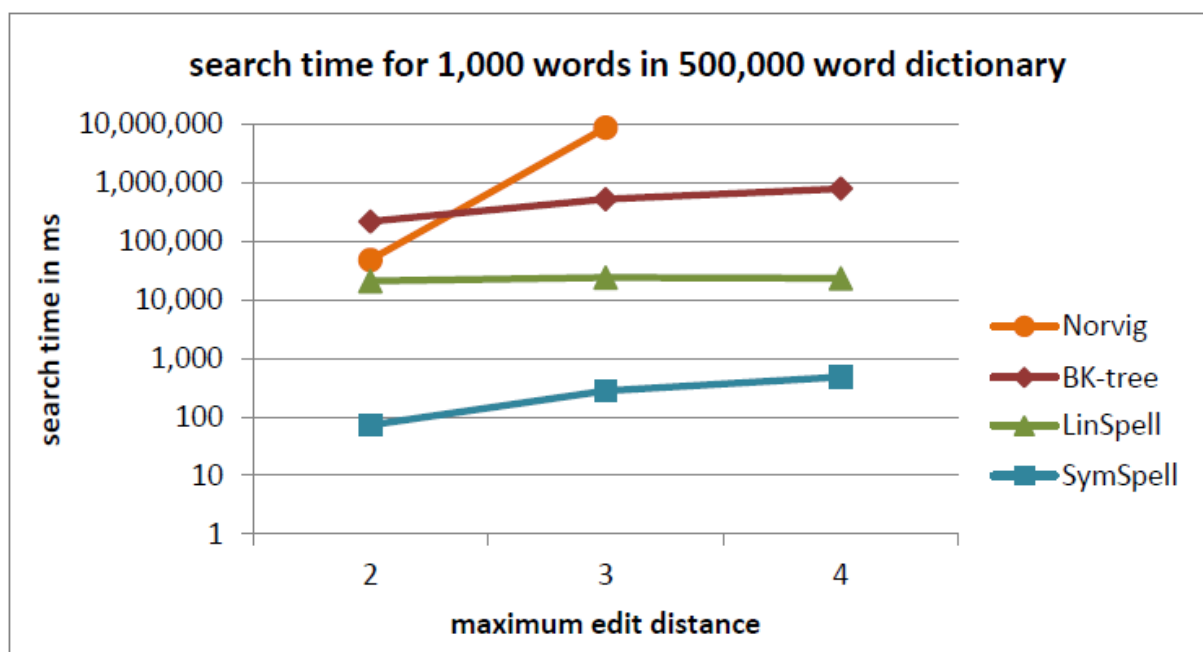


Fig. 20, Comparison of the search time for different spelling checker libraries

To further affirm the testing of the SymSpell library, it is tested on a 2012 Macbook that contains a single core. The result was 0.033 milliseconds/word when edit distance= 2. and 0.180 milliseconds/word when edit distance=3. To aid in increasing the accuracy of rectifying the misspelled words, a customised corpus was created. This corpus solely contains the

unigrams of all the names of the items, be it food, drinks or sides. Since Symspell takes into consideration the frequency of the words when it is doing spelling correction, each word has its own frequency connotated with it. Naturally, words that are in the list of stopwords, will have a higher frequency since there is a higher chance these words will be used. With this customised corpus, words that are unfamiliar to foreigners such as “Char Kway Teow” which is not found in an English dictionary, can be correctly rectified with ease.

4.1.6 Parser for Food Information

The settings in Dialogflow makes it only receptive to JSON formatted object as input for webhook. Thus, the preparation of this formatting will be done at the server (Business Logic Layer) to accommodate to the Dialogflow settings.

<to be added>

4.2 Front-End

As discussed in section 3.1, there are a few publishing platforms to be used as user interface. In this section, we will discuss the implementation details of each platform.

4.2.1 Facebook Messenger

The integration with Facebook Messenger was done with one-click integration feature provided by Dialogflow. There are several features in Facebook Messenger that could be used such as images and cards as response instead of only text response.

<To be added>

4.3 Database

The server database chosen for this project is MySQL. The database stores the information related to the food and beverages such as the price and the names of the items. Data retrieval can be passed easily from the database to the server by using the MySQL connector designed for Python Flask.

5. System Demonstration

5.1 Overview

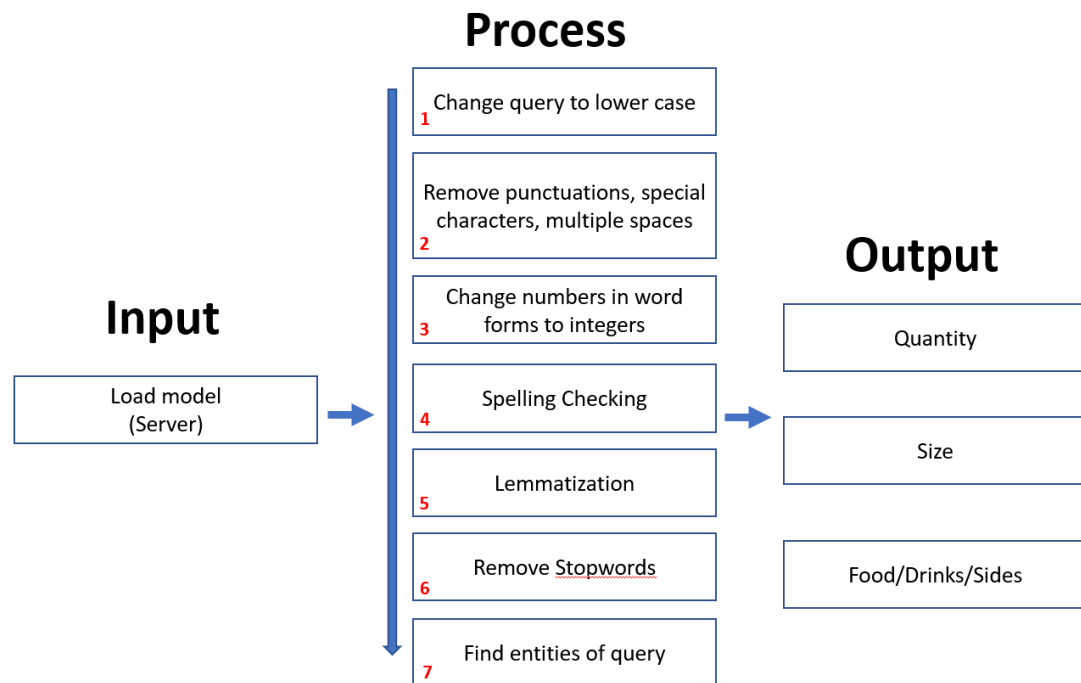


Fig. 21, overview of complex query process

The overview of the system's server is composed of 3 main portions. The first portion being the loading of the model with Spacy API. Next, the data will be processed using the Spacy framework coupled with Regex and SymSpell. The processing follows a sequential notion as seen in Fig. 21. The order of processing starts from 1, which changes the query to lower case, followed by 2, which removes punctuations, special characters and multiple spaces. This processing will continue with 3, 4, 5, 6 and lastly 7. The results of the processing would then be output as a JSON object containing the quantity, size and item's name. This JSON object will then be passed to Dialogflow to be shown into the user interface.

6. Conclusions and Future Work

6.1. Conclusion

<To be added>

6.2. Future Work

<To be added>

References

- [1] D.Marcus, 2016. Messenger Platform gets an update! [online] Facebook. Retrieved from: <https://www.facebook.com/notes/david-marcus/messenger-platform-gets-an-update/10155014173359148/>
- [2] 2016. 80% of businesses want chatbots by 2020. [online] Business Insider. Retrieved from: <http://www.businessinsider.com/80-of-businesses-want-chatbots-by-2020-2016-12>
- [3] Keery. R, “Robot uprising”, e.learning age, p38-39, Nov 2016
- [4] M. Chui, S. Lund and S. Ramaswamy, “What’s now and next in analytics, AI, and automation”, Mckinsey & Company, May 2017
- [5] B. David, “Applying Today’s Automation”, Convenience Store Decisions, Apr 2017, p58-61
- [6] A. Følstad, P. Brandtzaeg, “Chatbots – the new world of HCI”, ACM Interactions, Aug 2017, p38
- [7] A. Følstad, P. Brandtzaeg, “Why people use chatbots”, 4th International Conference on Internet Science, Nov 2017
- [8] B. Jaekel, “Wingstop soars with conversational commerce via chatbot initiative”, Retail Daily, Jun 2016
- [9] M. Hennessy, “Chatbots Are Changing How Customers Order”, QSR Magazine, Nov 2016
- [10] Daniel. B, Adrian. H.M and Florian.M, “Evaluating Natural Language Understanding Services for Conversational Question Answering Systems”, SIGDIAL 2017 conference, Aug 2017

[11] M. Gupta, “A Review of Named Entity Recognition (NER) Using Automatic Summarization of Resumes”, Towards Data Science, Jul 2018

<https://towardsdatascience.com/a-review-of-named-entity-recognition-ner-using-automatic-summarization-of-resumes-5248a75de175>

[12] Robert, “NLTK vs. spaCy: Natural Language Processing in Python”, The Data Incubator, Apr 2016