

National University of Sciences and Technology **(NUST)**

School of Electrical Engineering and Computer Science **(SEECS)**

Department of Computing

Final Year Project

**Speak Up- An Automated Conversational System for English Speaking Proficiency**

**Final Year Project Report by:**

Abdul Basit

Muhammad Abdullah Kazmi

In Partial Fulfillment

Of the Requirements for the degree

Bachelor of Science in Computer Science (BSCS)

School of Electrical Engineering and Computer Science

National University of Sciences and Technology

Islamabad, Pakistan (2020)

**DECLARATION**

We hereby declare that this project report entitled “SPEAK-UP AN AUTOMATED CONVERSATIONAL SYSTEM FOR ENGLISH SPEAKING PROFICIENCY” submitted to “SEECS”, is a record of an original work done by us under the guidance of Supervisor “DR. MUHAMMAD ALI TAHIR” and that no part has been plagiarized without citations. Also, this project work is submitted in the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Sciences.

**TEAM MEMBERS SIGNATURE**

ABDUL BASIT

MUHAMMAD ABDULLAH KAZMI

**ADVISOR SIGNATURE**

DR. MUHAMMAD ALI TAHIR

**DATE:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**PLACE:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**DEDICATION AND ACKNOWLEDGEMENTS**

We would like to dedicate our FYP to our families, who’s tremendous and continuous support towards us has been a source of inspiration for us. We would like to thank everyone who assisted us along this journey of our bachelor’s degree at SEECS, mostly those people who we have the honor of calling our friends.

We would like to thank our advisor, Dr. Muhammad Ali Tahir and co-advisors, Dr. Arsalan Ahmad and Dr. Muhammad Shehzad, who’s constant feedback and help has helped us get this far in the project.

**Table of Contents**

[**ABSTRACT 8**](#_Toc43144396)

[**INTRODUCTION 9**](#_Toc43144397)

[**LITERATURE REVIEW 11**](#_Toc43144398)

[**2.1 PROBLEM BACKGROUND 11**](#_Toc43144399)

[**2.2 PREVIOUS WORK 13**](#_Toc43144400)

[**PROBLEM DEFINITION 14**](#_Toc43144401)

[**METHODOLOGY 15**](#_Toc43144402)

[**4.1 RESEARCH WORK 15**](#_Toc43144403)

[**4.2 Proposed Methodology of Language Model Architecture 15**](#_Toc43144404)

[**4.3 A general approach to human-computer conversation 18**](#_Toc43144405)

[**4.3.1 Our proposal considers two main phases: 19**](#_Toc43144406)

[**4.3.1.1 Knowledge modeling 19**](#_Toc43144407)

[**4.3.1.2 Conversation flow 19**](#_Toc43144408)

[**4.3.2 Formal definitions and foundations 19**](#_Toc43144409)

[**4.3.3 Phase I: Knowledge modeling, extraction and storing 21**](#_Toc43144410)

[**4.3.4 Phase II: Conversation Flow 23**](#_Toc43144411)

[**DETAILED DESIGN AND ARCHITECTURE 26**](#_Toc43144412)

[**5.1 SYSTEM ARCHITECTURE 26**](#_Toc43144413)

[**5.2 ARCHITECTURE DESIGN 26**](#_Toc43144414)

[**5.3 DETAILED SYSTEM DESIGN 27**](#_Toc43144415)

[**5.3.1 User Authentication Module 27**](#_Toc43144416)

[**5.3.2 Query Module 27**](#_Toc43144417)

[**5.3.3 DIALOGFLOW 27**](#_Toc43144418)

[**5.3.4 Natural Language Generation Module 28**](#_Toc43144419)

[**5.3.5 Natural Language Understanding Module 28**](#_Toc43144420)

[**5.3.6 Session Logging 28**](#_Toc43144421)

[**5.3.7 Text to Speech Module 28**](#_Toc43144422)

[**5.4 DETAILED FRONTEND ARCHITECTURE 29**](#_Toc43144423)

[**5.4.1 Smart Mobile Application 29**](#_Toc43144424)

[**IMPLEMENTATION AND TESTING 37**](#_Toc43144425)

[**6.1 Deep Learning based model on Pytorch 37**](#_Toc43144426)

[**6.1.1 Preparations 37**](#_Toc43144427)

[**6.1.2 Load & Preprocess Data 38**](#_Toc43144428)

[**6.1.3 Create formatted data file 38**](#_Toc43144429)

[**6.1.4 Load and trim data 39**](#_Toc43144430)

[**6.1.5 Prepare Data for Models 40**](#_Toc43144431)

[**6.1.6 Defining Models 42**](#_Toc43144432)

[**6.1.7 Encoder 42**](#_Toc43144433)

[**6.1.8 Decoder 43**](#_Toc43144434)

[**6.1.9 Define Training Procedure 45**](#_Toc43144435)

[**6.1.10 Greedy decoding 47**](#_Toc43144436)

[**6.1.11 Run Model 48**](#_Toc43144437)

[**6.2 Rasa Dialogue Generation and Conversation Model 49**](#_Toc43144438)

[**6.2.1 Installing Rasa and its Dependencies 50**](#_Toc43144439)

[**6.2.2 Making Interactive Conversations 53**](#_Toc43144440)

[**6.2.3 Designing the conversational flow 53**](#_Toc43144441)

[**6.3 Dialogue Flow Model deployed using APIs 55**](#_Toc43144442)

[**6.3.1 Intents 55**](#_Toc43144443)

[**6.3.2 Entities 55**](#_Toc43144444)

[**6.3.3 Context 56**](#_Toc43144445)

[**6.3.4 Agents 57**](#_Toc43144446)

[**6.3.5 Small Talk 57**](#_Toc43144447)

[**6.3.6 Training Phrase: 58**](#_Toc43144448)

[**Visualizing Graphs and Attention 60**](#_Toc43144449)

[**Multi-head attention 61**](#_Toc43144450)

[**COMPARISON BASED ON DIFFERENT MODELS RESULT 65**](#_Toc43144451)

[**CONCLUSION AND FUTURE WORK 66**](#_Toc43144452)

[**REFERENCES: 67**](#_Toc43144453)

**Table of Figures**

[**Figure 1: Worldwide data 9**](#_Toc43144460)

[**Figure 2: English speaking in Pakistan 9**](#_Toc43144461)

[**Figure 3: Synchronous Model 16**](#_Toc43144462)

[**Figure 4: Asynchronous Model 17**](#_Toc43144463)

[**Figure 5: General Approach 18**](file:///D:\Downloads\Final%20FYP%20Report___.docx#_Toc43144464)

[**Figure 6: Knowledge Modeling, Extraction and Storing 21**](#_Toc43144465)

[**Figure 7:conversational Flow 24**](#_Toc43144466)

[**Figure 8: Queries and Intents 25**](#_Toc43144467)

[**Figure 9 Speakup home page 29**](#_Toc43144468)

[**Figure 10 Chatbot page 30**](#_Toc43144469)

[**Figure 11 Voicebot page 30**](#_Toc43144470)

[**Figure 12: Google SignUp | SignIN Figure 13: Signing in with Google 31**](#_Toc43144471)

[**Figure 14: Sign In with email 32**](#_Toc43144472)

[**Figure 15: Sign Up with email 33**](#_Toc43144473)

[**Figure 16: Creating Username 34**](#_Toc43144474)

[**Figure 17User Drawer 35**](#_Toc43144475)

[**Figure 18: User Settings 36**](#_Toc43144476)

[**Figure 19: Loading Dataset 38**](#_Toc43144477)

[**Figure 20: Printing Dataset 39**](#_Toc43144478)

[**Figure 21: Printing dataset entries 40**](#_Toc43144479)

[**Figure 22: Embeddings from dataset 41**](#_Toc43144480)

[**Figure 23: RNNS 42**](#_Toc43144481)

[**Figure 24: Encoder 43**](#_Toc43144482)

[**Figure 25: Decoder 44**](#_Toc43144483)

[**Figure 26: Attention Layer 45**](#_Toc43144484)

[**Figure 27: Clipping the model 46**](#_Toc43144485)

[**Figure 28: HYPER PARAMETERS 48**](#_Toc43144486)

[**Figure 29:PYTORCH MODEL RESULTS 49**](#_Toc43144487)

[**Figure 30: RASA Model 50**](#_Toc43144488)

[**Figure 31: Making Project 51**](#_Toc43144489)

[**Figure 32: Setting Intents 51**](#_Toc43144490)

[**Figure 33: Training 52**](#_Toc43144491)

[**Figure 34: Training Prameters 52**](#_Toc43144492)

[**Figure 35: Training Epochs 52**](#_Toc43144493)

[**Figure 36: Designing Conversation Flows 53**](#_Toc43144494)

[**Figure 37: Setting stories 54**](#_Toc43144495)

[**Figure 38: RASA Results 54**](#_Toc43144496)

[**Figure 39: Rasa Results 55**](#_Toc43144497)

[**Figure 40: DialogFlow Model 56**](#_Toc43144498)

[**Figure 41: Setting COntexts 57**](#_Toc43144499)

[**Figure 42: Small Talk 58**](#_Toc43144500)

[**Figure 43: Training Phrases 58**](#_Toc43144501)

[**Figure 44: Smart Talk Training 59**](#_Toc43144502)

[**Figure 45ENd User API 59**](#_Toc43144503)

[**Figure 46: DialogFlow Results 60**](#_Toc43144504)

[**Figure 47:Head and Layers 61**](#_Toc43144505)

[**Figure 48Graph :What is your name? 62**](#_Toc43144506)

[**Figure 49Graph :How are you? 63**](#_Toc43144507)

[**Figure 50Graph: Who are you? 64**](#_Toc43144508)

[**Figure 51: Comparing Results 65**](file:///D:\Downloads\Final%20FYP%20Report___.docx#_Toc43144509)

# ABSTRACT

Language Learning is the task of developing the ability to communicate in a language. Children who are weak in their spoken have difficulty in communicating with everyone, as they have difficulty in making sentences out of words. These children have a lot of random thought which they can describe in their native language but not in English. Speak Up has developed a language model using Natural Language Processing (NLP) and Deep Learning, and the aim of this project would be to help the children in learning better language sentence generation. Our project takes as input a sentence by our user and it will return an answer based on that sentence to conversate with the user. It will help the user to gain confidence and improve their spoken as well as understanding level of English. The application also provides our user to chat with our chatbot so that they can improve their sentence structure. The vocabulary we use consists of the most frequently used, day-to-day words, so that the child learns to communicate. Moreover, a child can conversate with our chatbot and see his improvement over the period. We tried multiple neural network models but integrated the one which was best suited in achieving our goal. Our models are based on Long Short-Term Memory (LSTM) architectures, which have much success in text-based domain.

# INTRODUCTION

Speak-Up is an intelligent conversational chatbot. Its main purpose is to improve the proficiency of English language for people who need it. It will help them in their daily as well as professional life. Mainly, Speak-Up will focus on the English difficulties faced by our students.

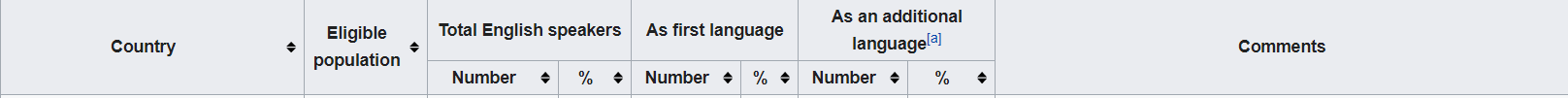
According to Wikipedia's 2009 report, English is Pakistan's official language and 49 per cent of Pakistani citizens can speak English.

Figure 1: Worldwide data



Figure 2: English speaking in Pakistan

According to the 2014 Dawn News article, English teaching is class-based because the education system in the country is designed to provide citizens of different economic groups with different levels of English teaching [1].

Many who can afford to go to better schools have access to better leaning content and well-trained teachers while the country's low status loses this opportunity to learn English in line with society's presumed expectations.

According to the British Council 's report "English in Pakistan Today: Class, Workspace and Shift in Language Use," the presence of pseudo-English-Medium Schools in Pakistan claiming to have English as their contact and interaction medium but the entire interaction between students and teachers was in Urdu [1].

It is said that English is confined to the 40 min class session rather than being taught as a language.

There is strong inequality that urban areas have better teaching and learning facilities than rural ones. The level of English proficiency among teachers in Pakistan is very low since 90 per cent of teachers in Punjab were unable to teach different subjects in English medium.

According to Punjab Director Richard Weyers of the British Council, only 3 percent of school and college students have access to private schools that impart proper English language while 97 percent of public school students have no access to English due to poor facilities.

The aim of our project is to help these growing number of individuals to better communicate in the official language of Pakistan, so that, there is no discrimination between the rich and the poor on the basis of language in our country.

Speak-Up solely focuses on the problems of the students. Students who are weak in speaking English can use our application to improve their spoken English by engaging themselves with the application.

The main users of our application will be children who have difficulty in speaking a language in day-to-day scenarios. Scope of this product is at national level at this time, as it will help the students of Pakistan in shaping their future better. Speak-Up will help the students in their academic as well as professional life, they will learn the importance of English language worldwide. Our main objective is to take this application to every underprivileged school of Pakistan and describe the importance of English as a whole [1].

Our aim down the line is to integrate our application with schools, so that our project can become a part of classroom and thus allow children to learn effectively in class.

In this section, we briefly discussed the reasons of why we are targeting this specific domain. We summarized the current issue and suggested our solution.

# LITERATURE REVIEW

## 2.1 PROBLEM BACKGROUND

Great Britain had ruled the Sub-Continent. Because of British law English became the official language of this region. After independence English maintained its status as an official language in Pakistan. We are a multilingual society. English and Urdu are our social, educational, and institutional languages [3].

These studies have also concluded that learning is influenced by several factors. Learning is a process which lasts for life. We may split this into two groups,

a) Natural Learning

b) (institutional) systematic learning

In Pakistan learning English is seen as a obstacle for both students and teachers alike. In English subject the failure ratio is very high and deceptive compared to other subjects. The rate of dropout is very high, till secondary stage. Pakistan is a destitute region.

Pakistan is a poor country. The bulk of Pakistan’s population lives in rural areas. Social ties and the element of schooling play a significant role in learning English [3].

English is ranked among the most influential and dominant languages in the world. Without their learning, no country in the world can imagine competing with the pace of growth, as it is important in the fields of industry, trade, commerce, science, communication, and technology and especially in education.

Narendra Rathod narrates in his paper, Social Factors in Second Language Learning, at an international conference on Global English on 5 November 2012, that the connection between social status and achievement exists in the L2. Most studies show that children from lesser socioeconomic groups are less competitive in L2 compared with children from higher groups [4].

English learning is only possible in a suitable and helpful environment. There is no factor that helps the learners in rural areas. To some extent all the factors are against learning. Because of the rural background, English learning process by the students is greatly affected.

English is known as the officer and the upper class. English enables modern books on engineering, medicine, agriculture, zoology, and literature to be read out. English helps to understand the cutting-edge technology. The English language constitutes the way forward. Now the world is turning into a global village and the language of foreign communications is English [5].

Teachers should be held highly esteemed as they enable our kids to do something in life. If you have good teachers, it can compensate for all other deficiencies. Specially the English teachers face a challenge as most children in our schools are first-generation learners. Language is extraneous to them. And English language teachers must develop the motivation between them to learn the language [5].

We are, unfortunately, a country where we could not agree on a national language strategy. Children taught in their mother tongue initially become better English language learners later. But we face multiple challenges in the absence of a language policy here [6].

English language teachers often take up the profession because they want to make a difference, see the world, get long summer vacations, get a thrill from the impact of being a favorite teacher. Some also take up teaching because it happens to be the tradition of their family or they may like wages and benefits [6].

## 2.2 PREVIOUS WORK

Augmented and Alternative Communications (AAC) is a term used to help people to understand and/or produce the language they speak and/or write. This is achieved either by supplementing their expression or by removing their voice or their writing entirely.

Certain devices for literate users are the latest such as the French Alphabet Board (FAB). For those who are skilled but who cannot easily communicate by speech, the most expensive of these devices is the light writer SL40. The computer is compact, lightweight, and easy to hold, and the dual screen enables the user to type the message while the user sits naturally for quick dialog. It also contains a text-to-speaking function when you talk to others in the room [4].

All these devices are restrictive and have their negative points that make it possible for the domain to improve seriously. Their main problems are in one or more categories:

● These are expensive products

● Many of these are not for adolescents, but for people who have developed their communication skills later in life.

With this in mind, and since we have to targeted children because this is their age for rapid improvement chances, we want to develop a system to help children communicate in everyday situations effectively while not at the same time being confined to the current vocabulary. They can explore any new word and effectively communicate, learn how to structure sentences in real life and ultimately reach the point where they can make these words themselves and are not device dependent [7].

# PROBLEM DEFINITION

With a 2015 report of low overall proficiency level, Pakistan ranks 48 from 72 countries on the list. The reasons are the level of spending on education. Many students are deprived of quality education due to not having enough schools in our rural areas. Schools that are created in our rural areas have no standard of education and there is no check and balance among them.

So, to combat this, Speak-Up aims to provide an English learning application, which will be targeted towards children in Pakistan who do not have proper English learning and it hinders their communication and speech.

The application will allow the students to express his/her random thoughts on the application by speaking a sentence and our application will analyze it and produce a response which will help the students to conversate with the application and improve their spoken as well as understanding level of English. This will be done through our in-house build AI Language Models. These language models are trained such that any sentence when given to this model, these Probabilistic Models will output the response from those sentences [7].

The application will further output the sentence not only in a text format, but also converts the text to speech using our endogenous text-to-speech module. This is immensely necessary, as research has shown that we tend to remember things when they are associated with a sound [7].

# METHODOLOGY

## 4.1 RESEARCH WORK

As mentioned in the Literature Review the previous worked are done on mostly closed ended question answering models and we tried to implement different models and see their results for our research purpose. We will be implementing firstly model from scratch and then try to improve it by tuning hyperparameters. Then we will define and train some other state of the art models like rasa, then we will check some other dialogue-based models.

**Deep Learning based model on Pytorch**

**Rasa Dialogue Generation and Conversation Model**

**Dialogue Flow Model deployed using APIs**

## 4.2 Proposed Methodology of Language Model Architecture

After preprocessing our dataset in this novel way, our next obstacle was how to develop a Neural Network architecture to suit our needs. After lots of experimentation, we ended up going with an **LSTM-Based Encoder-Decoder Architecture.** This architecture is the best so far when it comes to language models. It basically takes the input and converts it into a vector. This vector is then passed on to the hidden layers in the LSTM network. These hidden layers do the job of “remembering” the data as context data. At each time step, along with the input, this context vector is also passed along, so that when generating the output, all the previous context vectors are taken into deep consideration.

Our first model had a **Synchronous Architecture,** which means that both our backward and forward sequences were both being trained at the same time, “synchronously”. Both these datasets were trained on one single language model.

This allowed for sharing of parameters and so allowed both these models to be context-aware of each other.

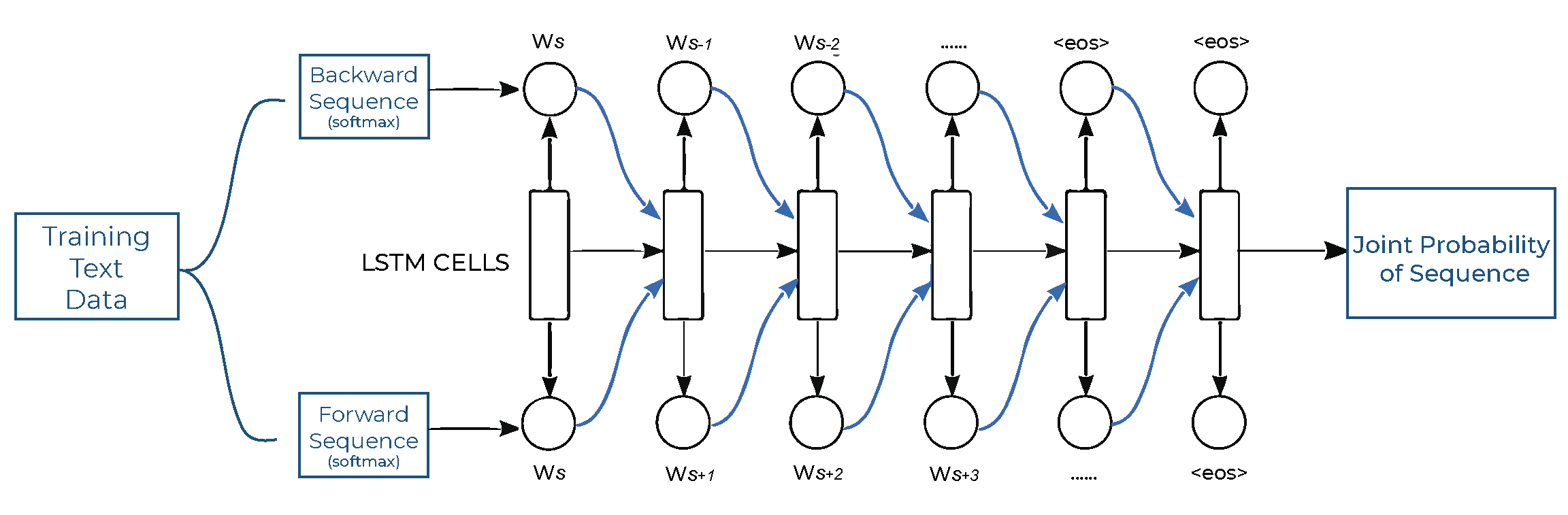


Figure 3: Synchronous Model

When both the datasets were trained simultaneously on this model, we then received a set of joint probabilities of words. We received probabilities because we were using *softmax* as the last layer of our model, and softmax basically assigns each word a probability after training. So, we then used this joint probability to generate output from our model. This proved to be a good method, but because of this joint training of two separate and distinct datasets, we hypothesized that we might get better results if we could somehow train these datasets separately. So, in order to test our hypothesis out, we then moved on towards developing another architecture to cater to our problem [12].

Our next model had an **Asynchronous architecture.** This essentially meant that now, we had two separate models for both of our datasets. The first model was to be trained only on backward sequences, while the second model would be trained on whole sequences, which would act as a final generator.

Our first model in this architecture was only trained on backward sequences, and this allowed the model to actually learn the internal structures of the sequences, and this allowed better optimization and learning of the model, as it was now being trained on sequences which were all similar, unlike in the first architecture. The probabilities we got from this model were far better.

Our next model in this architecture was trained on whole sequences, and not only on the second half of the sequence. What this allowed us to do was that whatever output we now got from our first model; this output would act as a *context* for this model. So, the whole sequence generated from the backward language model would be given as input (context) to our forward language model, and then on the basic of this input, it would generate far better sequences, as compared to synchronous mode [12].

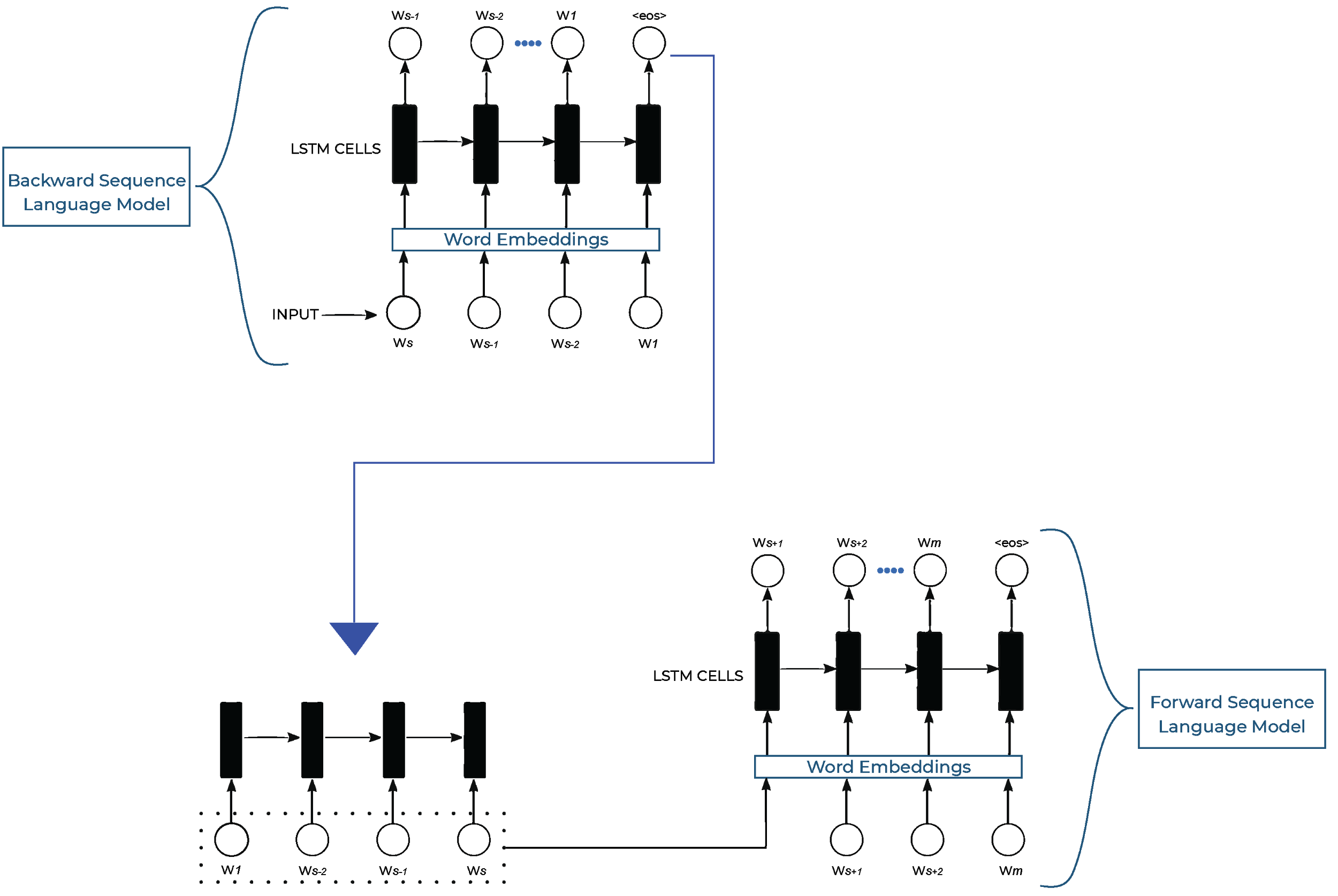


Figure 4: Asynchronous Model

## 4.3 A general approach to human-computer conversation

The very first developments like ELIZA (1966) and ALICE (1995) come to mind while learning about human-computer interaction, because of their significance. ELIZA was designed to prove that natural communication with a machine was feasible but did not pass the Turing test because its implementation was based on string matching and no meaning was provided for the written answers. ALICE launched the famous AIML (Artificial Intelligence Markup Language), which uses pattern matching rules to give the actual words 'meaning': topic, themes and categories have now been considered [10].

Chatbots have come a long way since. iOS apps also have basic chatbots that manage common questions, such as making phone calls and setting alarms (or maybe we can say who). In addition, automation tools were developed to create more complex chatbots, and are now available for commercial use. The ever-increasing presence of machine learning in everyday life has made the whole process of 'thinking' a little more powerful than simply matching patterns, and perhaps the most common examples are those available directly from tech giants like Google, Facebook and Amazon. While each structure having specifics and drawbacks in its own execution, most methods stem from a general concept —receive raw data, give it meaning, and then act appropriately according to a knowledge base. Fig. 1 illustrates this process for a text-based chatbot.

Figure 5: General Approach

To process natural language, chatbots rely on pattern recognition and bigram identification [14], a procedure which is usually handled by conversation frameworks. However, it is up to the chatbot designers to generate the knowledge base and provide the learning engine with appropriate examples.”

### 4.3.1 Our proposal considers two main phases:

### 4.3.1.1 Knowledge modeling

This process defines how information is interpreted in the knowledge base and processed therein.

### 4.3.1.2 Conversation flow

This process will describe both the lexicon used by the tutor, and the order in which ideas are introduced.

This section first presents the formal definitions and fundamentals of the methodology proposed. Each phase is later described and contrasted with actual-life queries. Ultimately, as specifics of execution differ across various discussion contexts, they are not addressed in the technique per se. Nonetheless, dispute management and good methods in execution are widely discussed with a case study.

### 4.3.2 Formal definitions and foundations

A chatbot can be described as a conversational agent which gives an appropriate response when prompted with a known query. Formally speaking, a chatbot is a function *f* of the form *f*:*Q* → *R*, which maps queries *q* ∈ *Q* to responses *r* ∈ *R*.

The question has to be translated from natural language to a given entry in the knowledge base to give the correct answer. This method includes breaking down the user feedback, i.e. a paragraph, to define core conversational concepts. Name and purpose are the two prominent principles at play. An entity is an abstract object which holds relevance to the user. It can be thought of both as a subject or an object in a conventional sentence: *The quick brown fox jumps over the lazy dog* refers to a *quick brown fox* as a subject, and to the *lazy dog* as the object. Both the subject and the object are entities in this sense and can be grouped together into classes.

In the other hand, intents are abstract representations of consumer wishes. If the user asks for a question, they may have something they want to do or know. However, the purpose is not always present in traditional questions.

When a person asks, ‘*Can you tell me what time it is?*’ their intention is to *find out the time*. The imperative sentence ‘*Show me my agenda*’, for example, can be rewritten as ‘*Would you show me my agenda?*’ which is a question in which the user wants *their agenda* (the entity) to be *shown* (the intent). Nevertheless, there are some other queries where the intention cannot be extracted from a rearrangement of their words. A user asking, ‘*Why do we snore?*’ wants to know the *reason* of why we *snore* when we sleep. The intention of finding out the *reason* is not in the input text.

**Queries as Functions of First-order Logic** First-order logic is a branch of the study of reasoning, dealing with inference and ‘belief’ management using formulas in the form of predicates. It uses truth-functional connectives like ¬ (not), ∧ (and), ∨ (or) and many others; along the use of functions to describe the state of a variable which is either true or false [8]. In the context of chatbots, this truth value be the presence or the absence of a condition to trigger a response if it exists in the knowledge base. At first glance, intents may resemble verbs in a common sentence, but in fact they are relations of variables to a truth state: first-order logic functions. A query, then, has the form:

*gn*(*t*1*,t*2*,..., tn*) (1)

where *gn* ∈ *G* is an *n*-ary function symbol in the set of functions *G*, and *ti* is a term in the set of terms *T*. The set of terms *T* is the set of known entities and *G* is the set of known intents in the knowledge base.

For example, show () is a function in the statement show(agenda) that receives a single parameter to generate an answer. Afterwards the agency goal is revealed to the consumer. Under this assumption, when the user wants to know the status of a recently booked flight, show(flight) could also be a possible query. If the current user wanted to know the time, the function would look similar: tell time(now). Another example in case the bot is designed to manage specific time zones or countries may be telling time(here). The Problem 'What's New Zealand Time? 'Can be translated as meaning time(nz), or anything in those lines. Another example in case the bot is designed to manage specific time zones or countries may be telling time(here). The Problem 'What's New Zealand Time? 'Can be translated as meaning time(nz), or anything in those lines. Following the function notation used throughout this work, it can be written as difference (x86, x64). The difference function is binary. However, *n*-ary functions may be defined, as stated in Eq. 1.

It's important to note that the causes of questions are not stored as text in the knowledge base. Alternatively, a predicate of the first order, with the form specified in Eq. For every question 1 is built. First-order logic has limited expressiveness [13], as there are some principles that cannot be articulated using this formal framework. Nonetheless, the information modeling process can be driven by sufficient delimitation of the scope of the agent, speeding up the implementation step and reducing the training complexity.

### 4.3.3 Phase I: Knowledge modeling, extraction and storing

As described above, the sets of queries and answers must first be defined, and there are several ways to perform that task. For example, Huang et al.[6] use tuples of the form <input, response> to deal with information extraction and representation, which are constructed by rating the responses of a web forum thread as either 'fascinating,' 'appropriate' or 'unfit.' Ales et al. [1], on the other hand, focus on automated emotion detection of news headlines by means of self-organizing maps to give meaning to data. Both ideas revolve around a medium-sized base of knowledge, with a few concepts and queries.

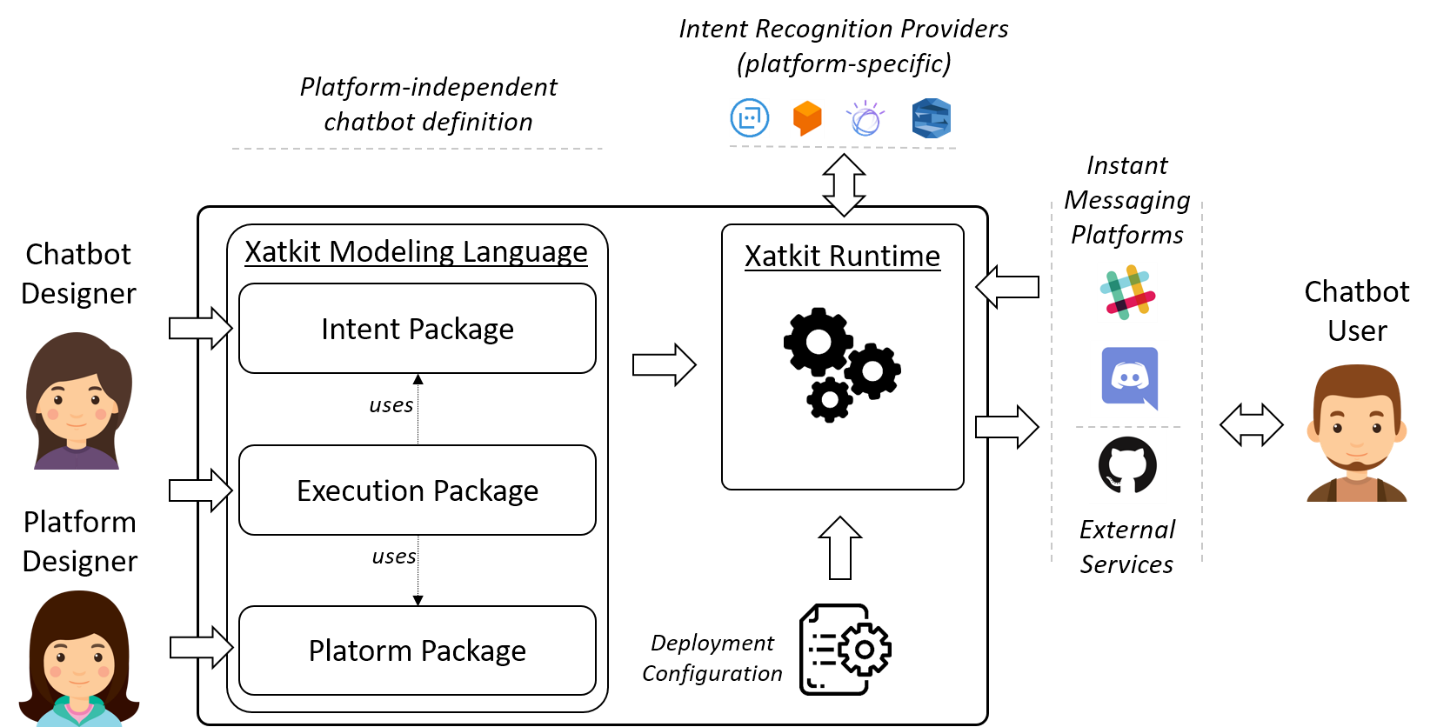


Figure 6: Knowledge Modeling, Extraction and Storing

A tutor designed for a college course, though, can address hundreds of different concepts and definitions. To this end, an expert 's input is recommended, especially if an adequate lexicon is used to aim for accurate pedagogical explications. Depending on the context, the tutor 's language can be either informal and relaxed, or a little more formal and detailed. For example, the input from an expert is advised in the field of mathematics — using abstract constructs and precise notes — because technology alone does not ensure that students learn mathematics better than using a regular textbook [12]. In addition, problem situations can be represented in a few ways, even in the natural language. It is important that students use an appropriate lexicon to incite them to translate everyday situations into mathematical models [11].

Information should be divided into 'units' to comply with the trigger-response strategy, extracted from the expert and then laid down for each unit of information on a knowledge base with other questions in mind. Each of these units represents a single query, a specific combination of functions and parameters giving some answer.

Although there are many data structures to store the knowledge base, most chatbot conversation structures are trees-based [15]. Each node in the tree represents a unique response, ranging from a simple greeting to detailed previous queries. It is also important to note that the similarity between user input and all known queries must be calculated for the conversation service to determine which answer the user is looking for. This process is usually done through machine learning algorithms using similarity measures between sentences in which each word or character can represent a single dimension, and its accuracy is refined by providing thousands of correctly labeled examples of user inputs. It is therefore advisable to group units of knowledge by similarity of the user input that will trigger them, rather than clustering them by subject.

For instance, grouping the examples provided above, one can see that all three queries using the function tell time have a similar input:

*What is the time now?*

*What time is it here?*

*What is the time in New Zealand?*

The input of the tell time function differs slightly from that of the show function, in which the phrase Show me predominates. Therefore, the creation of branches according to attempts reduces the search complexity.

### 4.3.4 Phase II: Conversation Flow

When information is divided into small atomic units the next step is to design how to present it. An easy way to do this is to create a glossary and a naming convention to keep track of the queries available and to control their trigger order. For e.g., each unit of knowledge in the tree was given a unique ID for the development of the intelligent tutor for the introductory mathematics course in our institution. The ID was automatically generated from abbreviations of the intents and entities names, with a hyphen separating the intent from the entities, and entities separated by a plus sign: Definition of the natural numbers was used by def-N, corresponding to the question '*What is a natural number?* ' or ' *What is natural numbers definition?’*

Some communication structures allow to group individuals into groups, as is the case with IBM Watson. Then, the glossary may contain entities clustered as follows:

**Numbers.** Named numbers sets, such as natural numbers, integrals, rationals, reals, etc.

**Terms.** Mathematical terms related to the material of the course, e.g. infinite decimal expansion and number representation.

**Equations.** Terms and notation specific to equations: system of linear equations and solution of a linear equation.

**Proofs.** Mathematical evidence requiring a rigorous and logical explanation, e.g. proof that 2 is irrational or proof that the cardinalities of the natural and the integral are the same.

**Modifiers.** Terms which alter the bot 's behavior. Typically used in combination with already specified entities, including no [other] solution and positive [something] or negative [something] solution.

**Algebraic Components.** Words specifically relevant to the algebraic operations,

For example, replacement or x of t, referring to the x(t) notation.

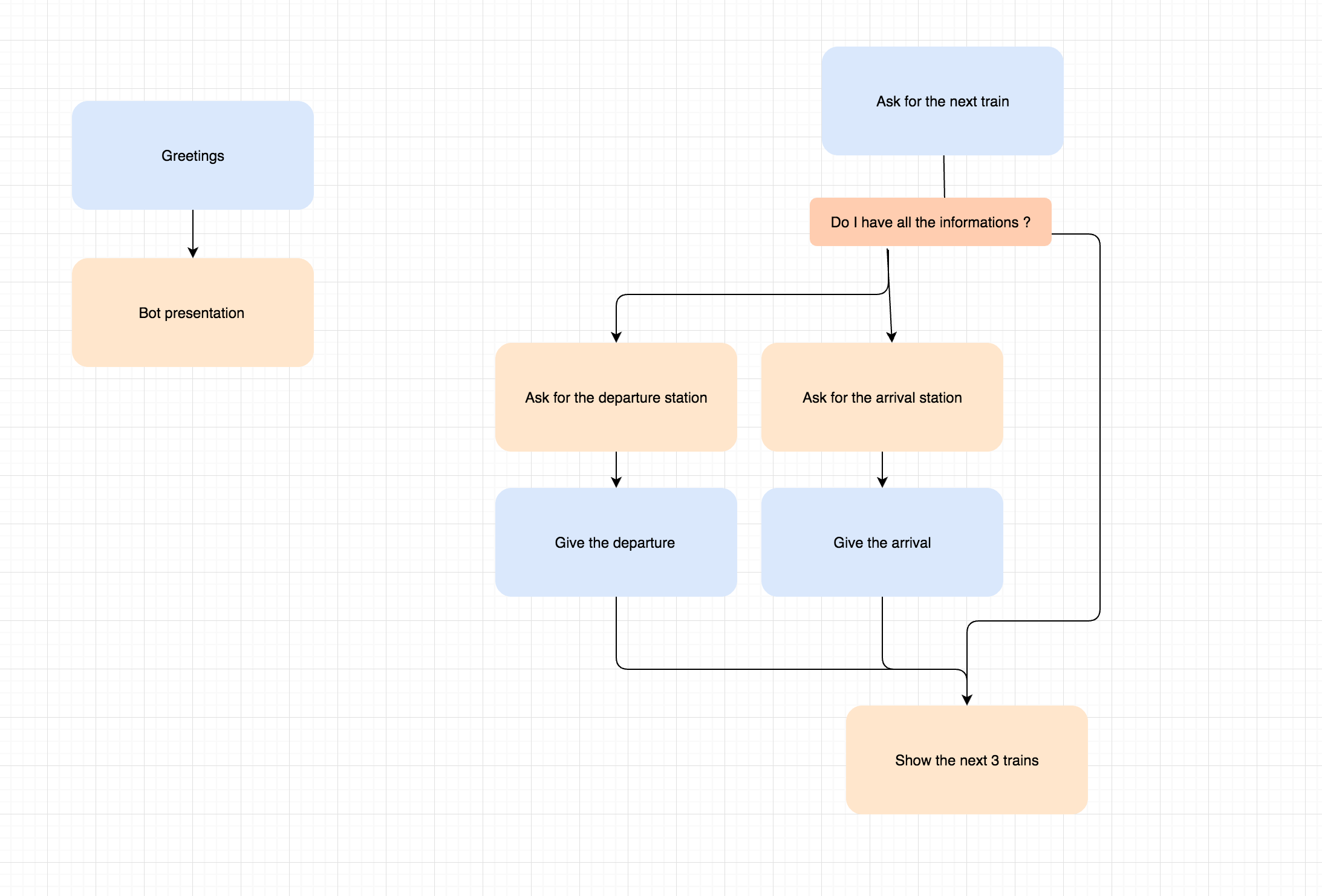
**Wrong terms.** This category may be used to encapsulate words sometimes asked that are either incorrect or non-existent. The term unreal numbers is a fine example: since the set of rational numbers and irrational numbers complement one another, unreal numbers should be a complement to the reals. It is simply a false belief.

Figure 7:conversational Flow

Entities are then coupled with attempts to formulate a particular set of conditions required to activate their response, written by the expert taking into account the pedagogical nature of the language used in the response. Through this way, teachers can work easily alongside software engineers to model the questions efficiently and generate the knowledge base.

An example is given of an abstract representation of the base of knowledge, where queries are grouped according to their meaning. The conversation structure will decide which branch contains the desired question from the top of the tree (usually a greeting or welcome message) and start iterating through the contents until it finds it or reaches the end. Chatbot programmers are advised to provide a note of error at the end of a section, should the question not be identified.

This method works if queries are provided in any order by the teacher. However, it may be possible to model more complicated conversations by splitting each of the various intents into smaller

pieces of conversation as needed.

Start

a

b

.

.

n

intent A

Entities

Query

Query

Query

Collapsed

branch

Figure 8: Queries and Intents

# DETAILED DESIGN AND ARCHITECTURE

## 5.1 SYSTEM ARCHITECTURE

The major functionalities and responsibilities of our system are as follows:

F-1: User must be able to sign up

F-2: User must be able to login

F-3: Application will display a certain number of words or images

(representing emotions) for the user to select

F-4: User will also have the option of typing his own word

F-5: The application will generate and display sentences

F-6: The sentences will also be converted to speech

F-7: This data will be logged by our application

The system was broken down into separate components based on their functionality. Se we basically decided to divide the system into multiple little subsystems, with each subsystem representing a certain core function of our application.

We found that this seemed to be the best approach going forward, as it allowed us to focus solely on every component functionality and invest time to produce quality output from the function.

It also made it easy to divide the responsibilities this way and made writing test cases faster and saved us time and headache afterwards [16].

## 5.2 ARCHITECTURE DESIGN

The major components of the system architecture include:

* System Frontend
* Natural Language Generation Module
* Natural Language Understanding Module
* Mobile Application Backend
* Text-to-Speech Component
* DIALOGFLOW Connectivity

**User Session Loggin****g System Flow Explained:**

* User Signs up or Logs in using the front end.
* Logging in also triggers the backend to start logging the session.
* Upon successful login, the user will be having a frontend where he will either have the option to chat or speak with our system.
* Upon speaking a sentence, the DIALOGFLOW will be triggered.
* Then these sentences will be returned to user via the DIALOGFLOW.
* Upon reaching the client, the words and the respective sentences generated will immediately be logged.
* These sentences will be displayed to the user.
* The user will also see a speech button on the front panel, pressing which will activate our Text-to-Speech module and convert the text to speech.

## 5.3 DETAILED SYSTEM DESIGN

### 5.3.1 User Authentication Module

User authentication module has the responsibility of signing up each individual user and maintaining his/her session logs. The sign-up process will be done by the parent, wherein he or she will provide credentials, which will be hashed and saved in our backend database.

During the Sign in process, the entered credentials would be checked through our database to make sure that the user is who he/she claims they are. Once confirmed, the user session would be displayed to the user [16].

### 5.3.2 Query Module

Once logged in, the user will then face the home page. On the drawer, there will the details of the user's profile. On the home page, there will be the option of either chatting or speaking to chatbot. Upon speaking a sentence, the DIALOGFLOW will be triggered [17].

### 5.3.3 DIALOGFLOW

Once the query has been entered, the DIALOGFLOW is triggered. This API is deployed on API Gateway, on Amazon Web Services. This is because our Machine Learning models are deployed on Amazon Sagemaker, so we needed a method to provoke those models. Furthermore, we created a microservice application using AWS Lambda function. This handles all the pre and post processing of text that was needed.

Once triggered, the API sends the request to the function, which after doing all the computations, returns the result to the client via the DIALOGFLOW [17].

### 5.3.4 Natural Language Generation Module

Our NLG module consists of two Deep Learning based Neural Probabilistic Language Models, who have learned the structure of English sentences, and are able to work in a joint effort to produce a semantically and syntactically correct sentence. The first half of the sequence is generated from the backward module, which is then sent to the next model as context, which completes the generation of the sentence. The successfully created sentences are returned to Lambda [17].

### 5.3.5 Natural Language Understanding Module

After the generation of sentences, the Lambda function sends the sentences to the NLU module. It is over here that different Parts of Speech tags, like nouns, verbs, adverbs, etc. are identified in the sentence, with the use of NLTK library. These are then joined with the sentence, and then the resultant is sent back [16].

### 5.3.6 Session Logging

As soon as the sentences are returned to the client, the user session immediately records what word was used to generate what sentences [16].

### 5.3.7 Text to Speech Module

Once the sentences are returned to the client, they are displayed to the user who queried for them, along with the sentence, there will be a ‘play’ button at the end of each sentence. Upon clicking it, Google’s Text-To-Speech module will be called upon using the google api query we have implemented [17].

## 5.4 DETAILED FRONTEND ARCHITECTURE

### 5.4.1 Smart Mobile Application

Smart mobile application is developed in flutter, whose main advantage is that it is CROSS-PLATFORM. It means you write the code once and then you can build both apps for Android OS i.e. APK and for iOS i.e. IPA.

Smart mobile App contains 9 major screens through which user can interact with the system.

**Home Screen:**

Home page of SMART Mobile App contains two buttons and one drawer on the Appbar, one will redirect the user to the chat screen and other one will redirect the user to conversating chatbot screen.



Figure 9 Speakup home page

**Chatbot Screen:**

Chatbot screen of SMART mobile App contains an input widget and a message option. User can type in a sentence and send it using the send button.



Figure 10 Chatbot page

**Voicebot Screen:**

Chatbot screen will contain three buttons, one for speaking, other for stopping and third one is for cancelling. This screen also contains two fields, in the first one our spoken sentence will appear and the response of the chatbot will appear in the second field.

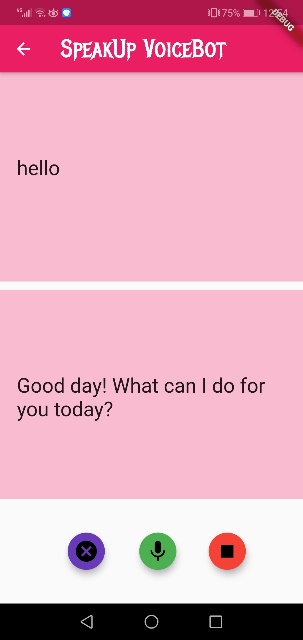


Figure 11 Voicebot page

**Google Sign in:**

Google sign in button will appear. User can click on that button and he will be asked to login by his current google account. This will automatically validate the user to his home screen if his google account is verified. This step is implemented by using Google Firestore SHA-1 and SHA-256 fingerprints. “Sign up using email” button will direct the user to the sign-up screen and “Already have an account? Log in.” will direct the user to the login screen.

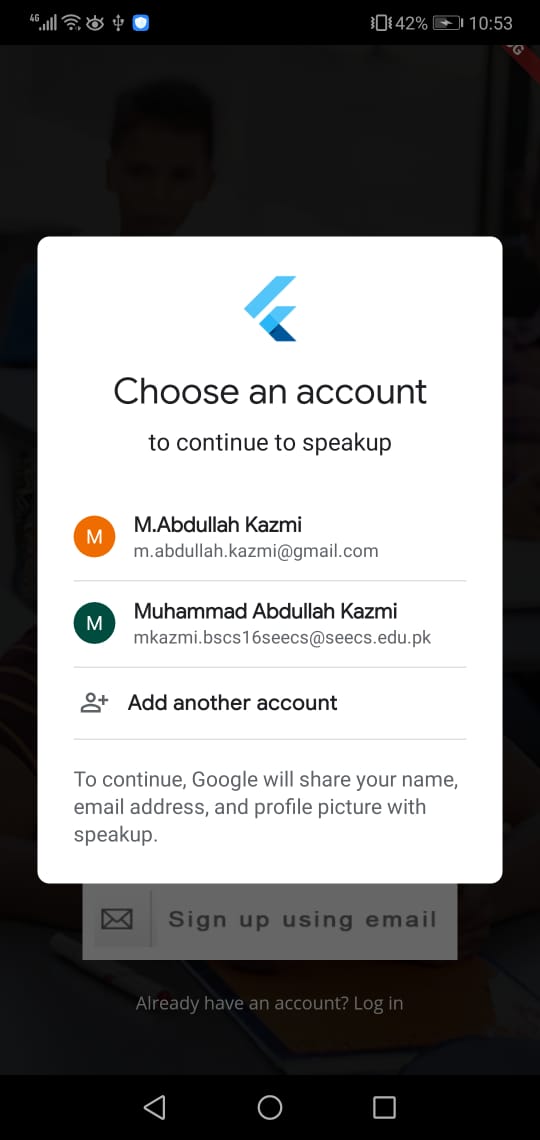
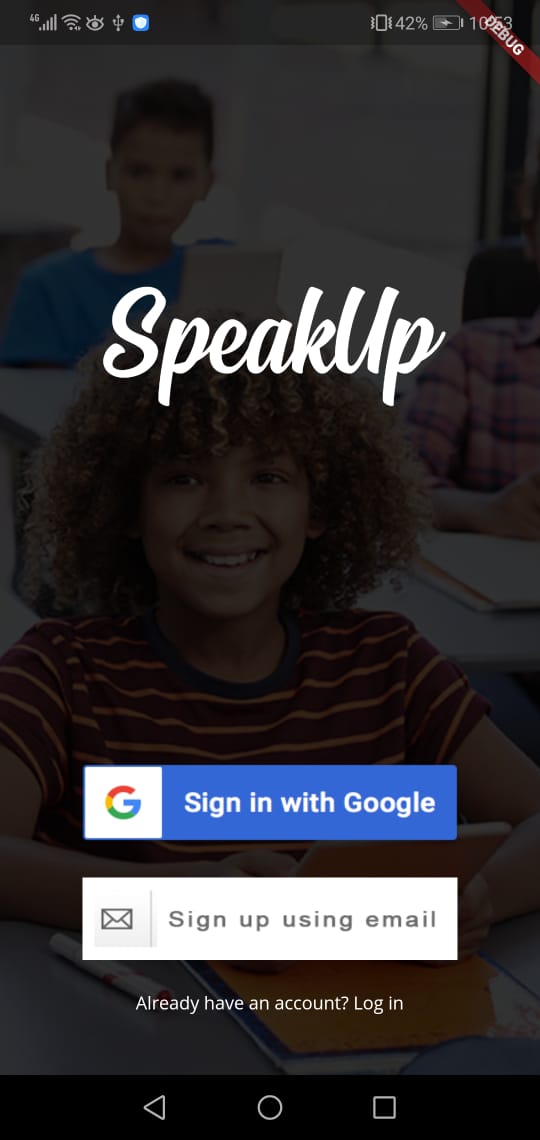


Figure 12: Google SignUp | SignIN Figure 13: Signing in with Google

**Login Screen:**

Login screen contains two input bars and one login button. Two input bars are for user to enter his email and password. After entering the credentials user will hit the sign in button. On hitting sign in button, app will send the POST request to backend firestore to validate the user and start the session. If user validates successfully, app will redirect the user to Profile Screen or else he will be told to reset the credentials in case of wrong credentials.

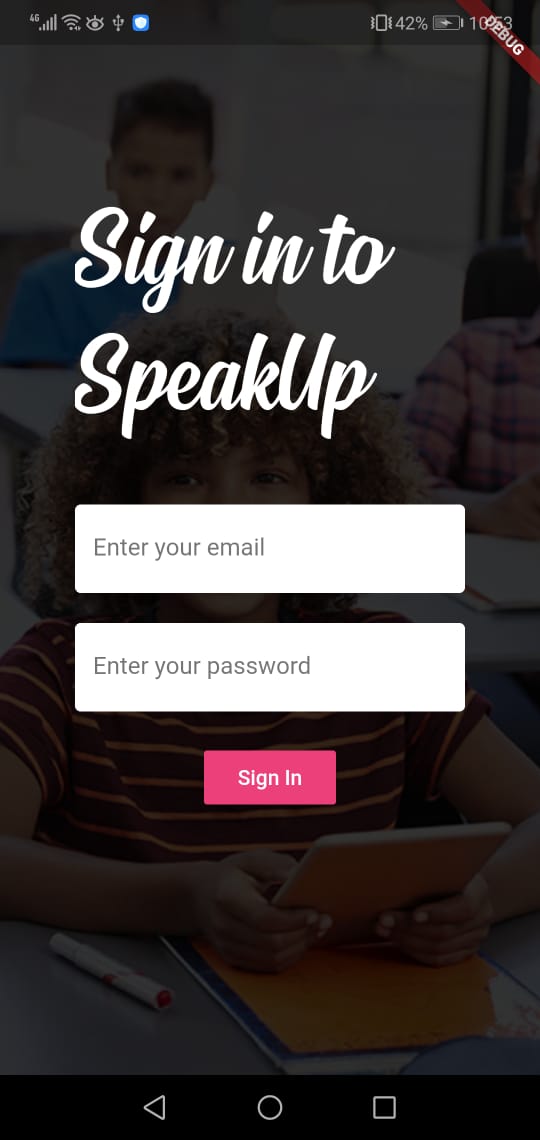


Figure 14: Sign In with email

**Sign up page:**

Sign up page contains three input bars and one sign up button. Three input bars are required to fill up for signing up to use our system. In these three input bars, user will provide his email, new password and confirm new password. On hitting sign up, new user will be created against unique userID and his information will be stored. If user enters unverified email, his account will not be created.

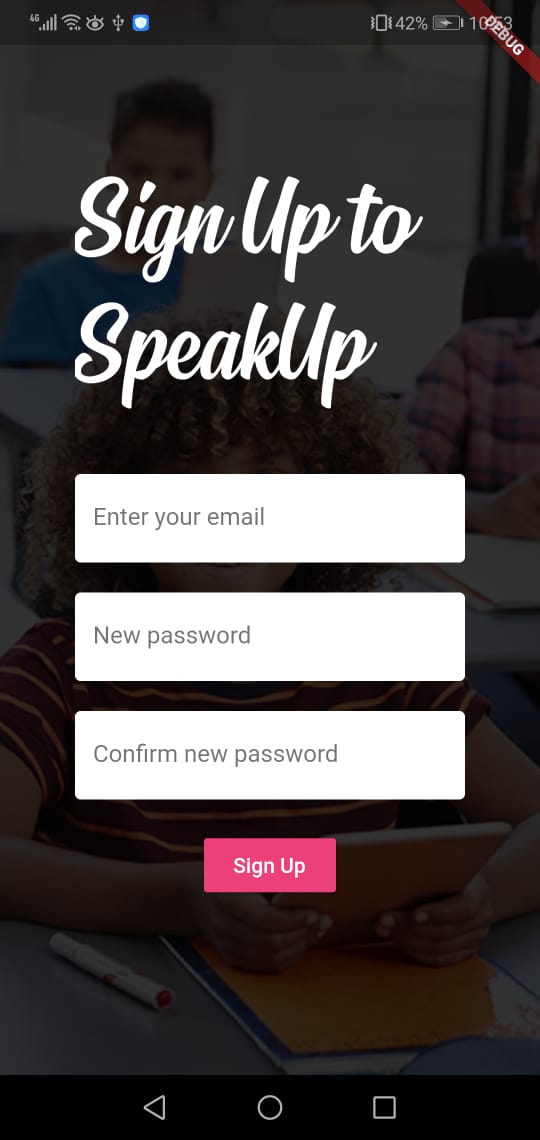


Figure 15: Sign Up with email

**Username page:**

On signing up successfully, user will be directed to create username page, where he will be asked to enter his username for Speak-Up. Username page will have one input field and one button. Input field will take his username and upon pressing the button that username will be created. Entered username will be stored in firestore against unique userID.

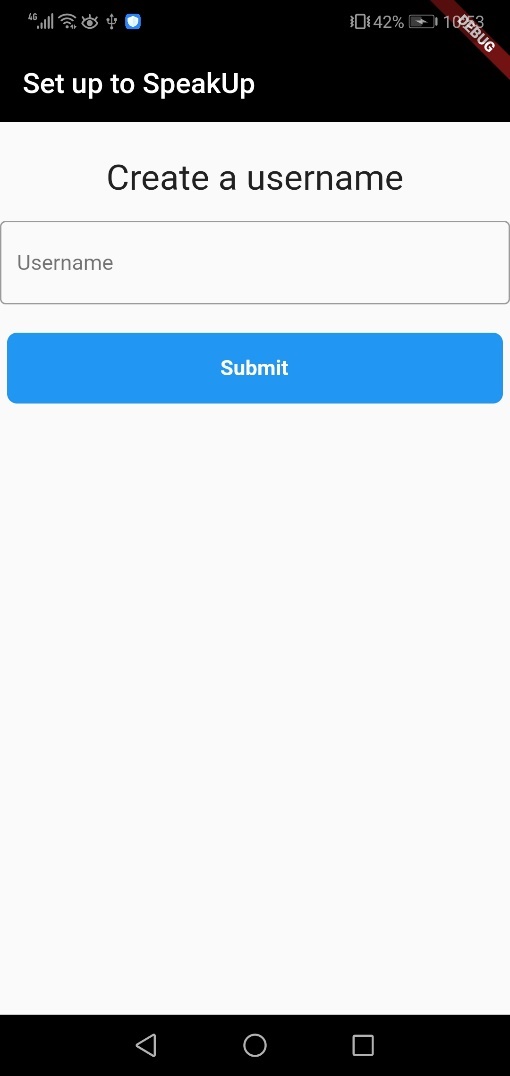
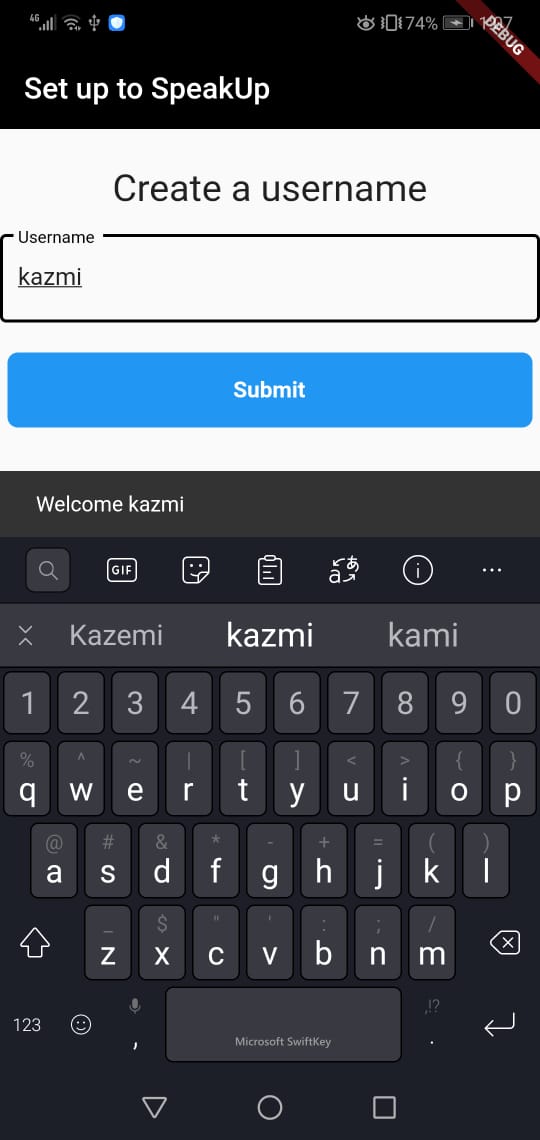
 

Figure 16: Creating Username

**Drawer:**

Drawer will display user’s image, his username and display name with two buttons. Setting button will direct the user to the setting page where he can edit his username and display name. Logout button will end the session for the user and redirect him to the login page.

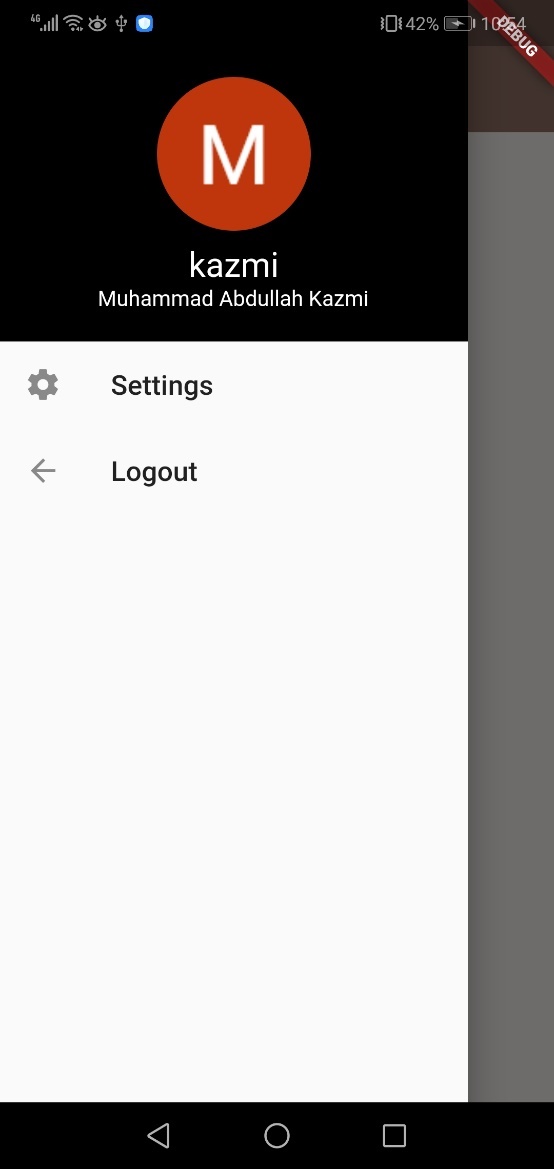


Figure 17User Drawer

**Settings page:**

Settings page will contain two input fields and one button. One input field will be for changing display name and the other one will be for changing username. Upon pressing edit profile button, snackbar will appear on the bottom of the screen notifying the user that changes have been made.

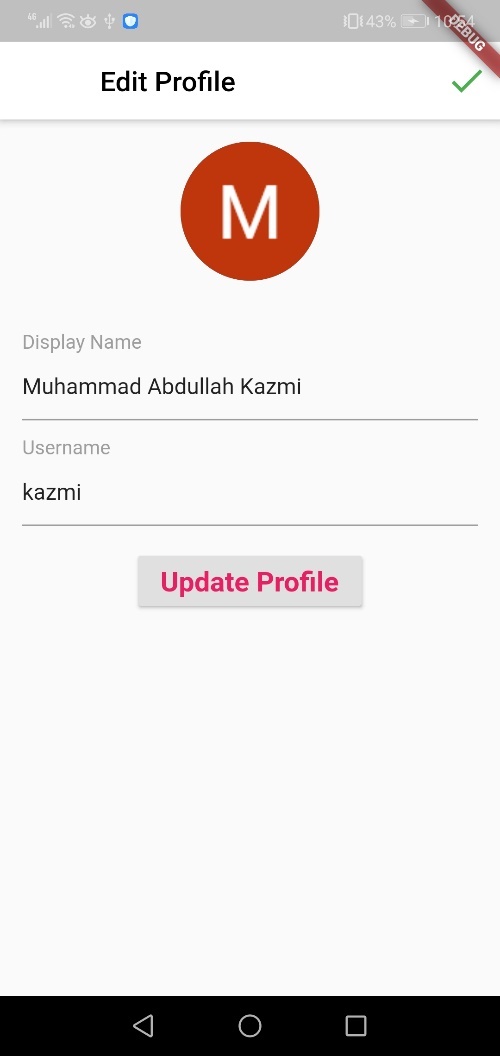


Figure 18: User Settings

# IMPLEMENTATION AND TESTING

## 6.1 Deep Learning based model on Pytorch

### 6.1.1 Preparations

To start, download the data ZIP file here and place it under the current directory in a data / directory.

Following this let us import some necessities.

**from** \_\_future\_\_ **import** absolute\_import

**from** \_\_future\_\_ **import** division

**from** \_\_future\_\_ **import** print\_function

**from** \_\_future\_\_ **import** unicode\_literals

**import** torch

**from** torch.jit **import** script, trace

**import** torch.nn **as** nn

**from** torch **import** optim

**import** torch.nn.functional **as** F

**import** csv

**import** random

**import** re

**import** os

**import** unicodedata

**import** codecs

**from** io **import** open

**import** itertools

**import** math

USE\_CUDA **=** torch**.**cuda**.**is\_available()

device **=** torch**.**device("cuda" **if** USE\_CUDA **else** "cpu")

### 6.1.2 Load & Preprocess Data

The next move is to reformat our data file and then load the data into systems in which we can function.

Cornell Movie-Dialogs Corpus is a rich feature film character dialog dataset:

220,579 Speeches between 10,292 pairs of film characters

9,035 of 617 characters

Total statements 304,713

This dataset is broad and complex, and the formality of language, time intervals, emotion, etc. varies greatly. Our hope is that this diversity will make our model robust to many forms of queries and inputs.

Next, to display the original version, we'll look at some lines of our datafile.

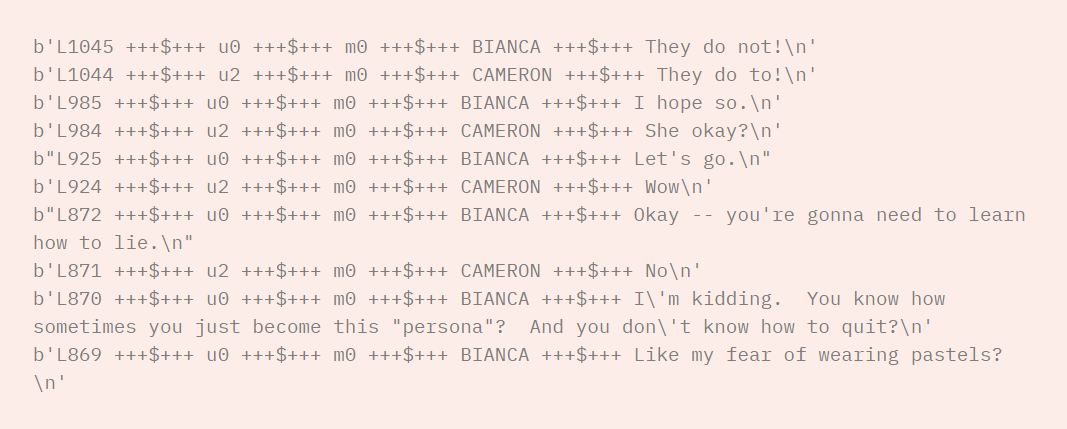


Figure 19: Loading Dataset

### 6.1.3 Create formatted data file

For convenience, we will create a beautifully formatted data file which contains a tab-separated query sentence and a response sentence pair in each line.

The following functions allow the extraction of the data file from raw movie lines.txt.

LoadLines divides each file line into a field dictionary (lineID, characterID, movieID, character, text).

LoadConversations group fields of loadLines lines into movie conversations.txt based conversations.

ExtractSentencePairs removes pairs of conversational sentences.

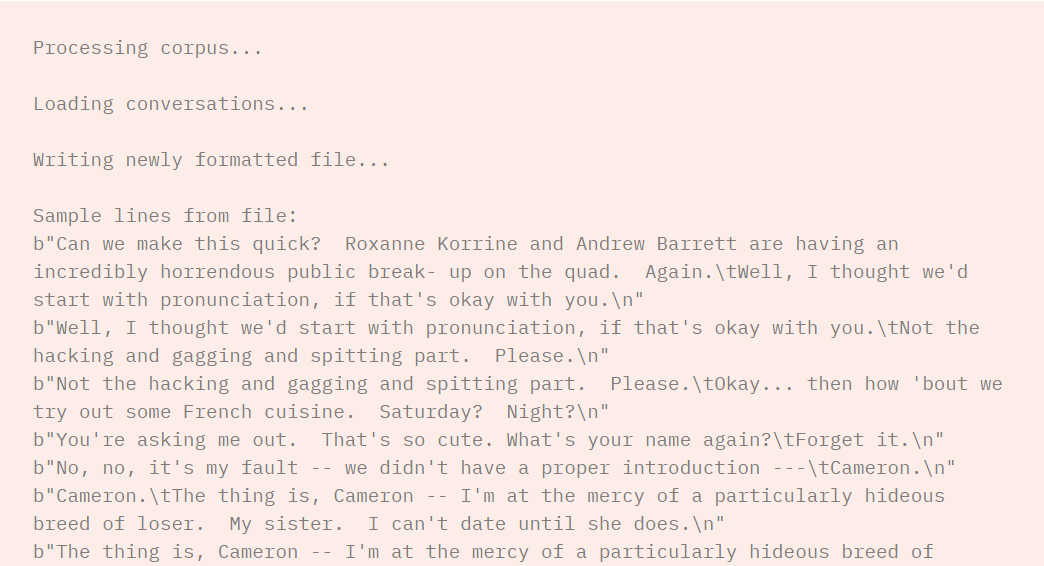


Figure 20: Printing Dataset

### 6.1.4 Load and trim data

Our next business order is to build a vocabulary and load the pairs of query / response sentences into memory.

Note that we are dealing with sequences of words that don't implicitly map to a discrete numerical space. So, we must create one by mapping every single word that we find in our dataset to an index value.

We define a Voc class for this, which maintains a mapping from words to indexes, a reverse mapping of indexes to words, a count of each word and a total count of words. The class includes methods for adding a word to the vocabulary (addWord), adding all terms to a sentence (addSentence), and trimming terms (trim) that are rarely used. More trimming will be discussed later.

Now we can assemble our pairs of sentences for the vocabulary and query / response. We have to perform some pre-processing before we're ready to use this data.

Second, we will use unicodeToAscii to convert the Unicode strings to ASCII. Next, all letters should be converted to lowercase and all non-letter characters trimmed except for simple punctuation (normalizeString). Eventually, we must filter out sentences longer than the MAX LENGTH threshold (filterPairs) to help in training convergence.

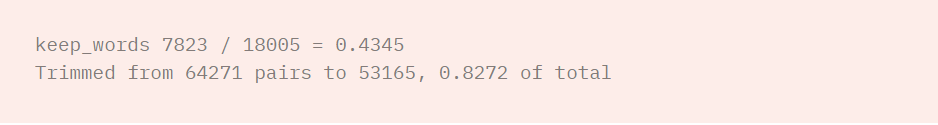


Figure 21: Printing dataset entries

### 6.1.5 Prepare Data for Models

Though we've put a lot of effort into preparing and massaging our data into a nice vocabulary object and a list of sentence pairs, our models will eventually expect numerical torch tensors as inputs. One way to prepare the data processed for the models can be found in the translation tutorial seq2seq. In this tutorial, we use a batch size of 1 , which means that all we have to do is convert the words in our sentence pairs from the vocabulary to their corresponding indexes and feed this into the models [13].

However, you will need to train with mini batches if you are interested in speeding up training and/or would like to leverage GPU parallelization capabilities.

Using mini-batches also means we have to be mindful of the difference in the duration of the sentence in our lots. In order to accommodate sentences of different sizes in the same batch, we will make our batched input shape tensor (max length, batch size) where sentences are zero padded after an EOS\_token. [13].

If we simply convert our English sentences to tensors by converting words to their indexes (indexesFromSentence) and zero-pad, our tensor will form (batch size, max length), and indexing the first dimension will return a complete sequence over all time measures. We need, however, to be able to index our batch over time and across all batch sequences. So we transpose our input batch shape to (max length, batch size), so that indexing across the first dimension returns a time step across all the batch sentences. We address this implicitly transpose in the feature zeroPadding.

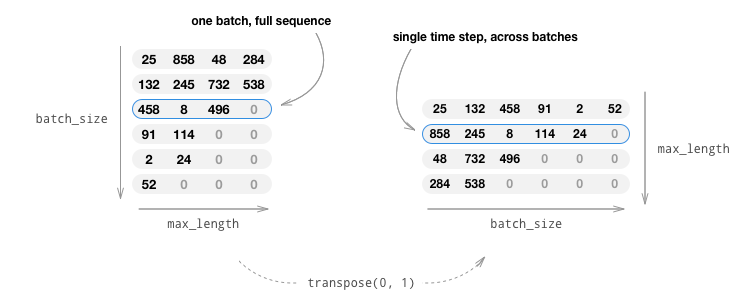


Figure 22: Embeddings from dataset

The function inputVar handles the process of converting sentences to tensor and eventually produces a properly formed zero-padded tensor. It also returns a tensor of lengths for each of the batch sequences that will be passed later to our decoder.

The function outputVar performs a similar function to inputVar, but instead of returning a tensor for lengths, it returns a binary mask tensor and a maximum length for the target sentence. The binary mask tensor has the same form as the target tensor for the output but each element that is a PAD token is 0 and all other elements are 1.

Batch2TrainData simply takes a bunch of pairs and uses the functions to return the input and target tensors.

### 6.1.6 Defining Models

#### **6.1.6.1 Seq2Seq Model**

Our Chatbot 's brains are a sequence-to - sequence model (seq2seq). The goal of a seq2seq model is to take a sequence of variable lengths as an input and to return a sequence of variable lengths as an output using a fixed model.

Sutskever et al . discovered that we can accomplish this function by using two separate recurrent neural nets together. One RNN acts as an encoder that encodes an input sequence of variable lengths to a context vector of a fixed length. This context vector (the RNN 's final secret layer) will, in principle, contain semantic knowledge about the query sentence that is input into the bot. The second RNN is a decoder that takes an input word and context vector and returns a guess for the next word in the sequence and a hidden state to be used in the next iteration [12].

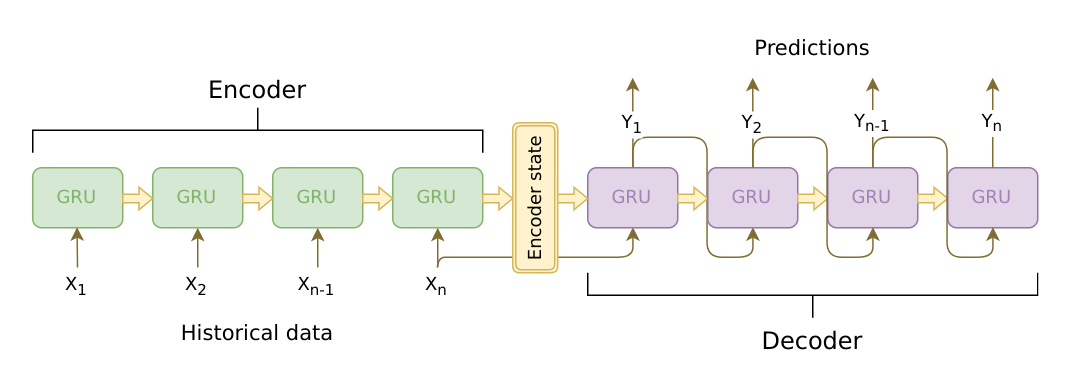


Figure 23: RNNS

### 6.1.7 Encoder

The RNN encoder iterates one token (e.g. word) at a time through the input sentence, outputting a "output" vector and a "hidden state" vector at each time step. The secret state vector is then transferred to the next phase of the process, thus recording the output vector. The encoder converts the context it saw at each point of the sequence into a set of points in a high-dimensional space that will be used by the decoder to produce meaningful output for the given task [20].

A multi-layered Gated Recurrent Module is at the core of our encoder, developed in 2014 by Cho et al .. We 're going to use a GRU bidirectional version, meaning that there are actually two separate RNNs: one that feeds the input sequence in regular sequential order, and one that feeds the input sequence backwards. Every time step the outputs of each network are summed up. Using a bidirectional GRU will give us the advantage of encoding context past as well as future.

Bidirectional RNN:

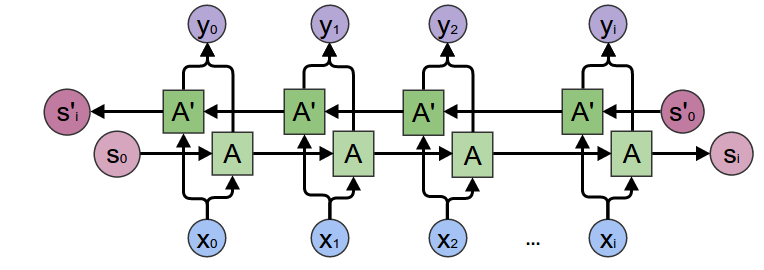


Figure 24: Encoder

Note that an embedding layer is used to encode our word indices in a function space that is arbitrarily sized. For our models, this layer maps every word to a hidden size feature space. These values should code semantic similarity between similar meanings when trained [20].

Finally, if we move a padded sequence batch to an RNN node, we have to pack and unpack the padding around the RNN package using nn.utils.rnn.pack padded sequence and nn.utils.rnn.pad packed sequence, respectively.

**Computation Graph:**

1. Convert word indexes to embeddings.
2. Pack padded batch of sequences for RNN module.
3. Forward pass through GRU.
4. Unpack padding.
5. Sum bidirectional GRU outputs.
6. Return output and final hidden state.

### 6.1.8 Decoder

The decoder RNN generates the response sentence in a token-by-token fashion. It uses the encoder’s context vectors, and internal hidden states to generate the next word in the sequence. It continues generating words until it outputs an EOS\_token, representing the end of the sentence. A common problem with a vanilla seq2seq decoder is that if we rely solely on the context vector to encode the entire input sequence’s meaning, it is likely that we will have information loss. This is especially the case when dealing with long input sequences, greatly limiting the capability of our decoder [21].

To combat this, [Bahdanau et al.](https://arxiv.org/abs/1409.0473) created an “attention mechanism” that allows the decoder to pay attention to certain parts of the input sequence, rather than using the entire fixed context at every step.

At a high level, attention is calculated using the decoder’s current hidden state and the encoder’s outputs. The output attention weights have the same shape as the input sequence, allowing us to multiply them by the encoder outputs, giving us a weighted sum, which indicates the parts of encoder output to pay attention to. [Sean RoModelson’s](https://github.com/spro) figure describes this very well [22]:

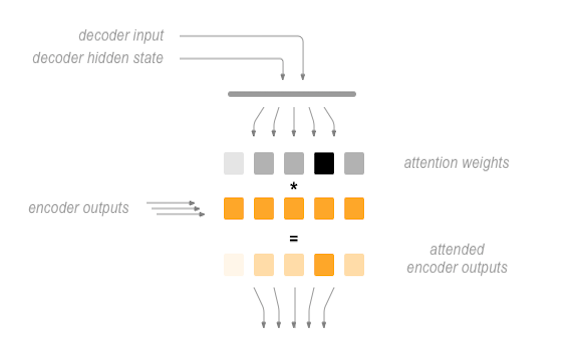


Figure 25: Decoder

[Luong et al.](https://arxiv.org/abs/1508.04025) improved upon Bahdanau et al.’s groundwork by creating “Global attention”. The key difference is that with “Global attention”, we consider all of the encoder’s hidden states, as opposed to Bahdanau et al.’s “Local attention”, which only considers the encoder’s hidden state from the current time step. Another difference is that with “Global attention”, we calculate attention weights, or energies, using the hidden state of the decoder from the current time step only. Bahdanau et al.’s attention calculation requires knowledge of the decoder’s state from the previous time step. Also, Luong et al. provides various methods to calculate the attention energies between the encoder output and decoder output which are called “score functions”:[23]

where htht = current target decoder state and h¯sh¯s = all encoder states.

Overall, the Global attention mechanism can be summarized by the following figure. Note that we will implement the “Attention Layer” as a separate nn.Module called Attn. The output of this module is a softmax normalized weights tensor of shape *(batch\_size, 1, max\_length) [24]*.

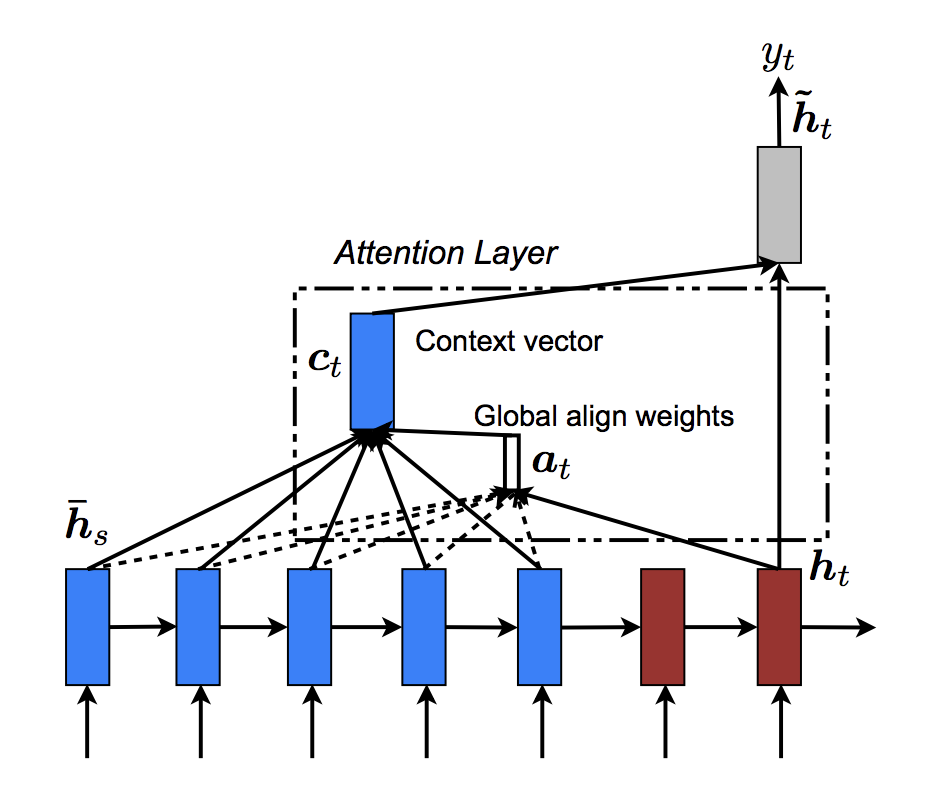


Figure 26: Attention Layer

### 6.1.9 Define Training Procedure

#### **6.1.9.1 Masked loss**

Since we are dealing with batches of padded sequences, we cannot simply consider all elements of the tensor when calculating loss. We define maskNLLLoss to calculate our loss based on our decoder’s output tensor, the target tensor, and a binary mask tensor describing the padding of the target tensor. This loss function calculates the average negative log likelihood of the elements that correspond to a 1 in the mask tensor [25].

#### **6.1.9.2 Single training iteration**

The train function contains the algorithm for a single training iteration (a single batch of inputs).

We will use a couple of clever tricks to aid in convergence:

The first trick is using **teacher forcing**. This means that at some probability, set by teacher\_forcing\_ratio, we use the current target word as the decoder’s next input rather than using the decoder’s current guess. This technique acts as training wheels for the decoder, aiding in more efficient training. However, teacher forcing can lead to model instability during inference, as the decoder may not have a sufficient chance to truly craft its own output sequences during training. Thus, we must be mindful of how we are setting the teacher\_forcing\_ratio, and not be fooled by fast convergence [25].

The second trick that we implement is **gradient clipping**. This is a commonly used technique for countering the “exploding gradient” problem. By clipping or thresholding gradients to a maximum value, we prevent the gradients from growing exponentially and either overflow (NaN) or overshoot steep cliffs in the cost function [25].

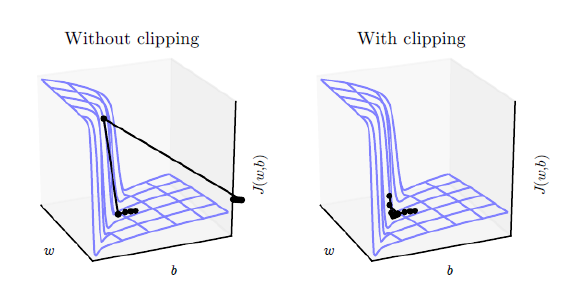
[](https://pytorch.org/tutorials/_images/grad_clip.png)

Figure 27: Clipping the model

**Sequence of Operations:**

1. Forward pass entire input batch through encoder.
2. Initialize decoder inputs as SOS\_token, and hidden state as the encoder’s final hidden state.
3. Forward input batch sequence through decoder one-time step at a time.
4. If teacher forcing set next decoder input as the current target; else: set next decoder input as current decoder output.
5. Calculate and accumulate loss.
6. Perform backpropagation.
7. Clip gradients.
8. Update encoder and decoder model parameters.

After training a model, we want to be able to talk to the bot ourselves. First, we must define how we want the model to decode the encoded input.

### 6.1.10 Greedy decoding

Greedy decoding is the decoding method that we use during training when we are **NOT** using teacher forcing. In other words, for each time step, we simply choose the word from decoder\_output with the highest softmax value. This decoding method is optimal on a single time-step level.

To facilize the greedy decoding operation, we define a GreedySearchDecoder class. When run, an object of this class takes an input sequence (input\_seq) of shape (input\_seq length, 1), a scalar input length (input\_length) tensor, and a max\_length to bound the response sentence length. The input sentence is evaluated using the following computational graph:

**Computation Graph:**

1. Forward input through encoder model.
2. Prepare encoder’s final hidden layer to be first hidden input to the decoder.
3. Initialize decoder is first input as SOS\_token.
4. Initialize tensors to append decoded words to.
5. Iteratively decode one-word token at a time:
6. Forward pass through decoder.
7. Obtain most likely word token and its softmax score.
8. Record token and score.
9. Prepare current token to be next decoder input.
10. Return collections of word tokens and scores.

### 6.1.11 Run Model

Finally, it is time to run our model!

Regardless of whether we want to train or test the chatbot model, we must initialize the individual encoder and decoder models. In the following block, we set our desired configurations, choose to start from scratch or set a checkpoint to load from, and build and initialize the models. Feel free to play with different model configurations to optimize performance.

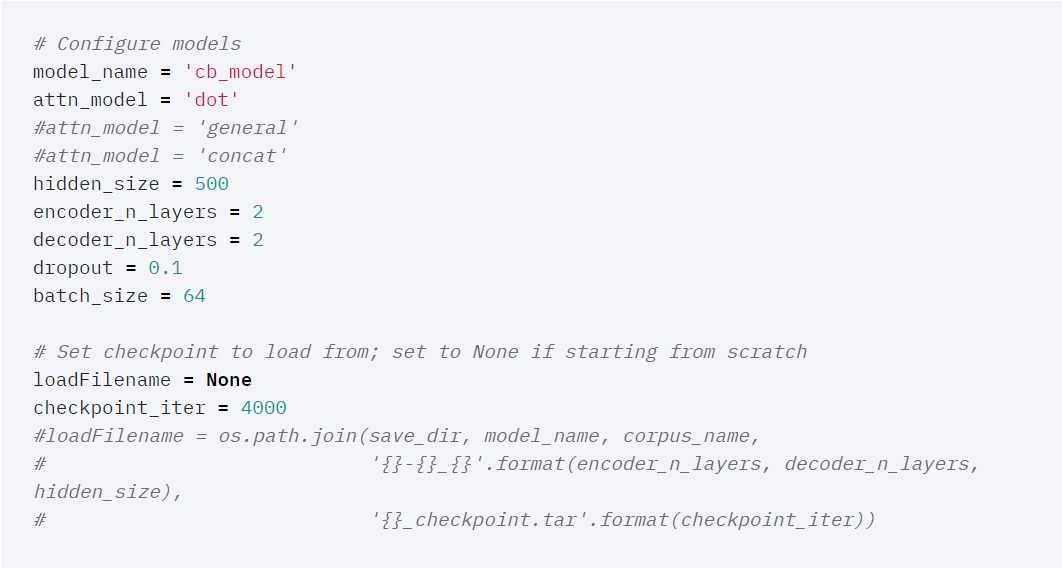


Figure 28: HYPER PARAMETERS

Results:

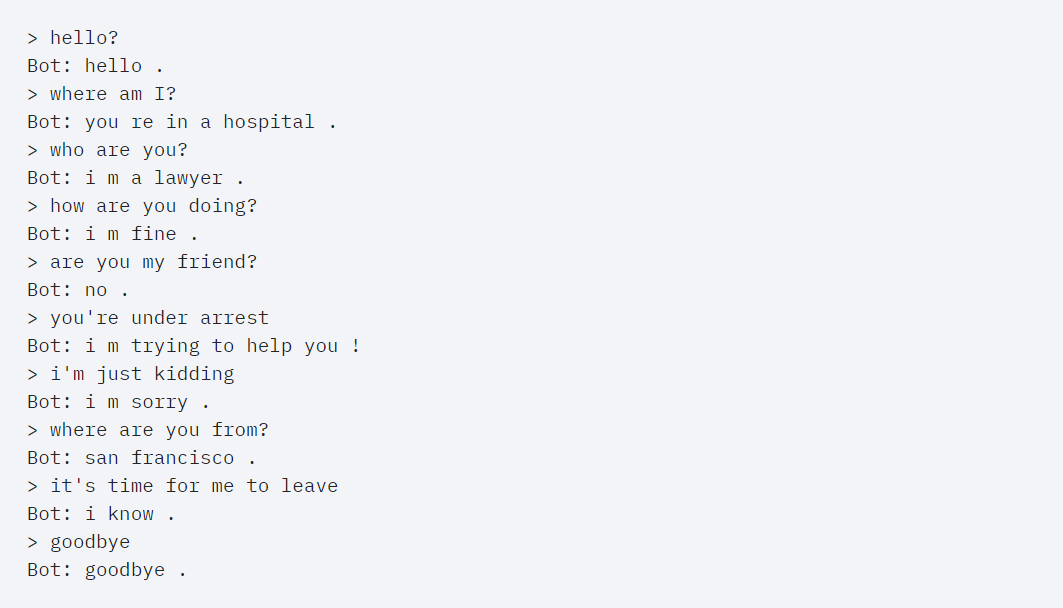


Figure 29:PYTORCH MODEL RESULTS

## 6.2 Rasa Dialogue Generation and Conversation Model

Why we use the Rasa Stack for Building Chatbots

The [Rasa Stack](https://rasa.com/products/rasa-stack/) is a set of open-source NLP tools focused primarily on chatbots. In fact, it is one of the most effective and time efficient tools to build complex chatbots in minutes. Below are three reasons why I love using the Rasa Stack:

It lets you**focus on improving the “Chatbot” part** of your project by providing readymade code for other background tasks like deploying, creating servers, etc.

The default set up of Rasa**works well right out of the box** for intent extraction and dialogue management, even with lesser data

Rasa stack is**open source**, which means we know exactly what is happening under the hood and can **customize** things as much as we want [16],

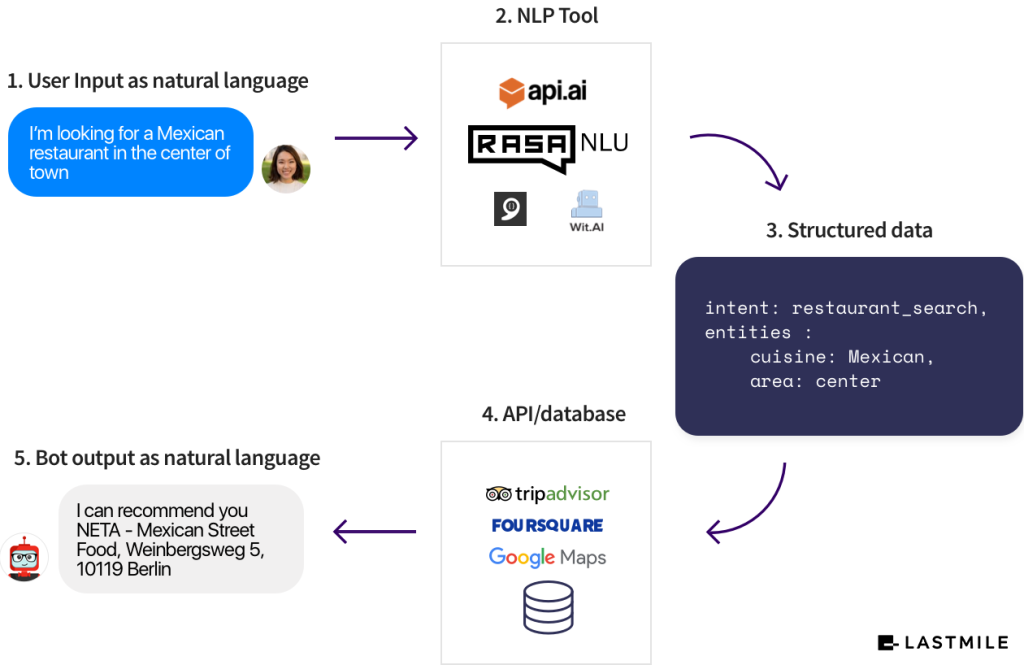


Figure 30: RASA Model

### ****6.2.1 Installing Rasa and its Dependencies****

You can use the code below to install all the dependencies of the Rasa Stack:

pip install -r requirements.txt

This step might take a few minutes because there are quite a few files to install. You will also need to install a spaCy English language model:

python -m spacy download en

Extracting User Intent from a Message

The first thing we want to do is figure out the **intent**of the user. What does he or she want to accomplish? Let us utilize Rasa and build an NLU model to identify user intent and its related entities [16].

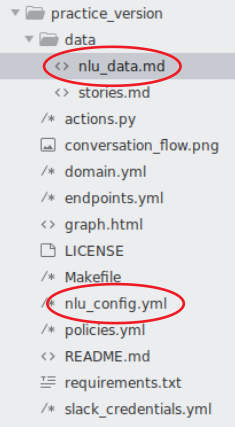


Figure 31: Making Project

The two files we will be using are highlighted above.

**data/nlu\_data.md** – This is the file where you will save your training data for extracting the user intent. There is some data already present in the file:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/04/nlu_data.png)

Figure 32: Setting Intents

As you can see, the format of training data for ‘intent’ is quite simple in Rasa. You just must:

Start the line with “## intent:intent\_name”

Supply all the examples in the following lines

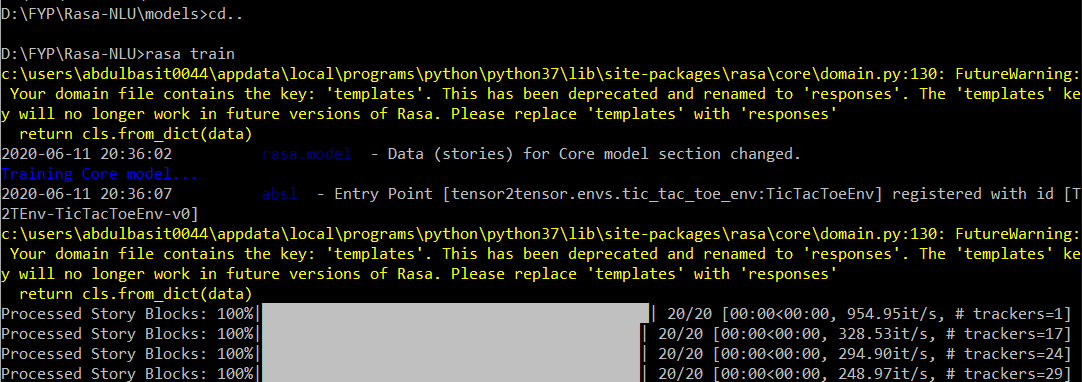


Figure 33: Training

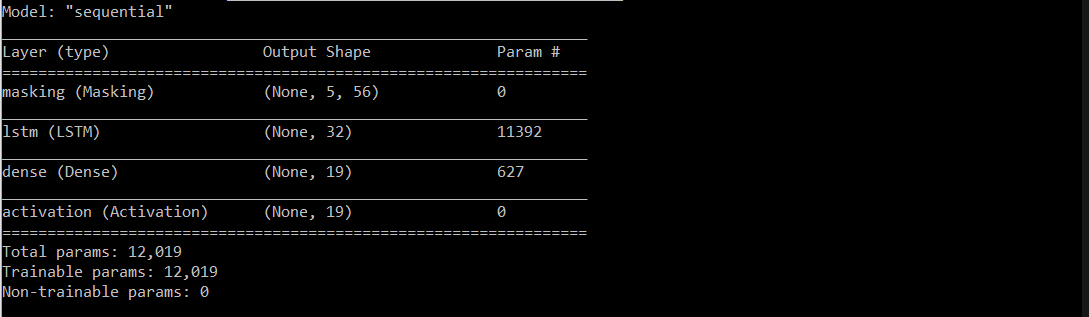


Figure 34: Training Prameters

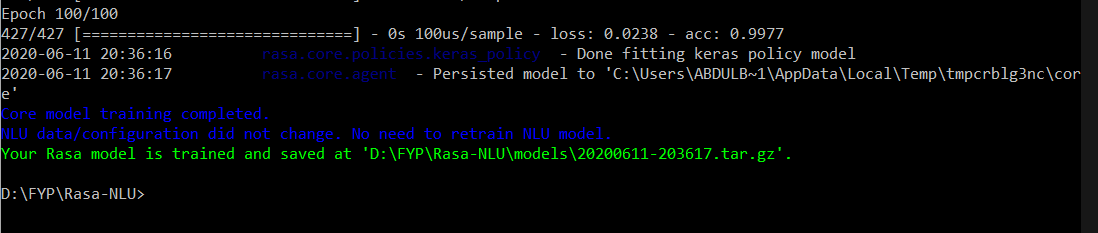


Figure 35: Training Epochs

### 6.2.2 Making Interactive Conversations

One of the most important aspects of a chatbot application is its ability to be **interactive**. Think back to a chatbot you have used before. Our interest naturally piques if the chatbot can hold a conversation, right?

The chatbot is expected to extract all the necessary information needed to perform a task using the back and forth conversation it has with the end user [17].

### 6.2.3 Designing the conversational flow

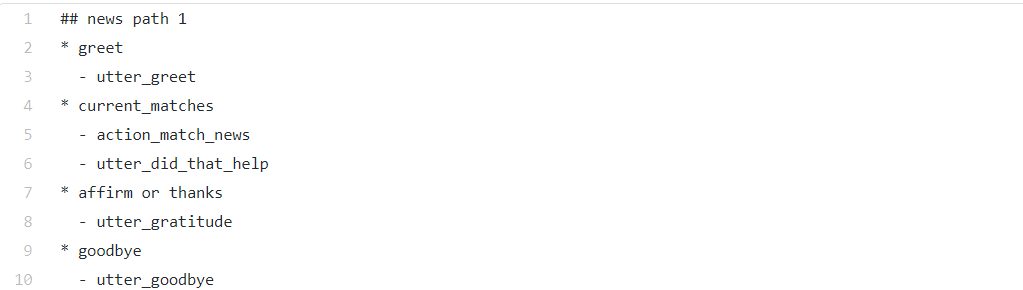


Figure 36: Designing Conversation Flows

In addition to classified intents, extracted entities and previous dialogue states, the predictions of the dialogue management model can be influenced by slots. [Slots](https://rasa.com/docs/core/slots/) are key-value stores designed to keep the context of the conversation by storing important pieces of information throughout the conversation or until reset. Using slots, you can model when the assistant should ask for the necessary details and when to skip these questions if the details were already provided.

In some situations, details returned by custom actions can also influence the dialogue. Slots returned by custom actions provide information about the outside world and drive the conversation to a specific direction. For example, in the newsletter subscription example used above, the behavior of the assistant differs depending on whether the user is already a subscriber of the newsletter or not. In this case, a custom action can check if the user has already subscribed to the newsletter and set a Boolean slot to True or False using a SlotSet () method [28]:

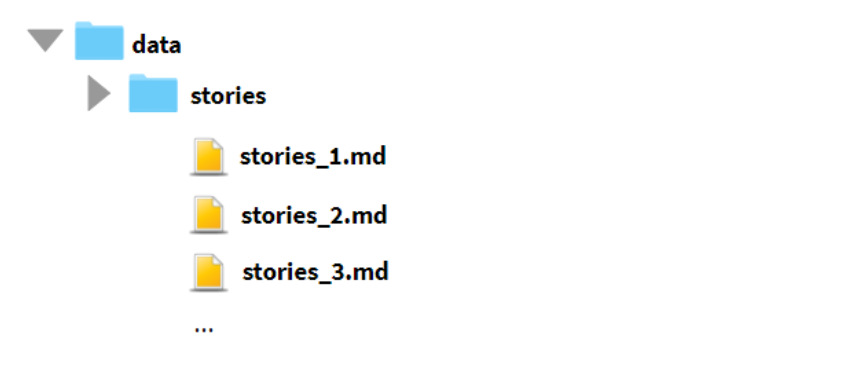


Figure 37: Setting stories

Training data plays a crucial part in building a successful AI assistant. How much and what kind of stories you need always depends on your use case, so it is important to keep in mind what users you are developing your assistant for. This is the conversation snippet from rasa model.”

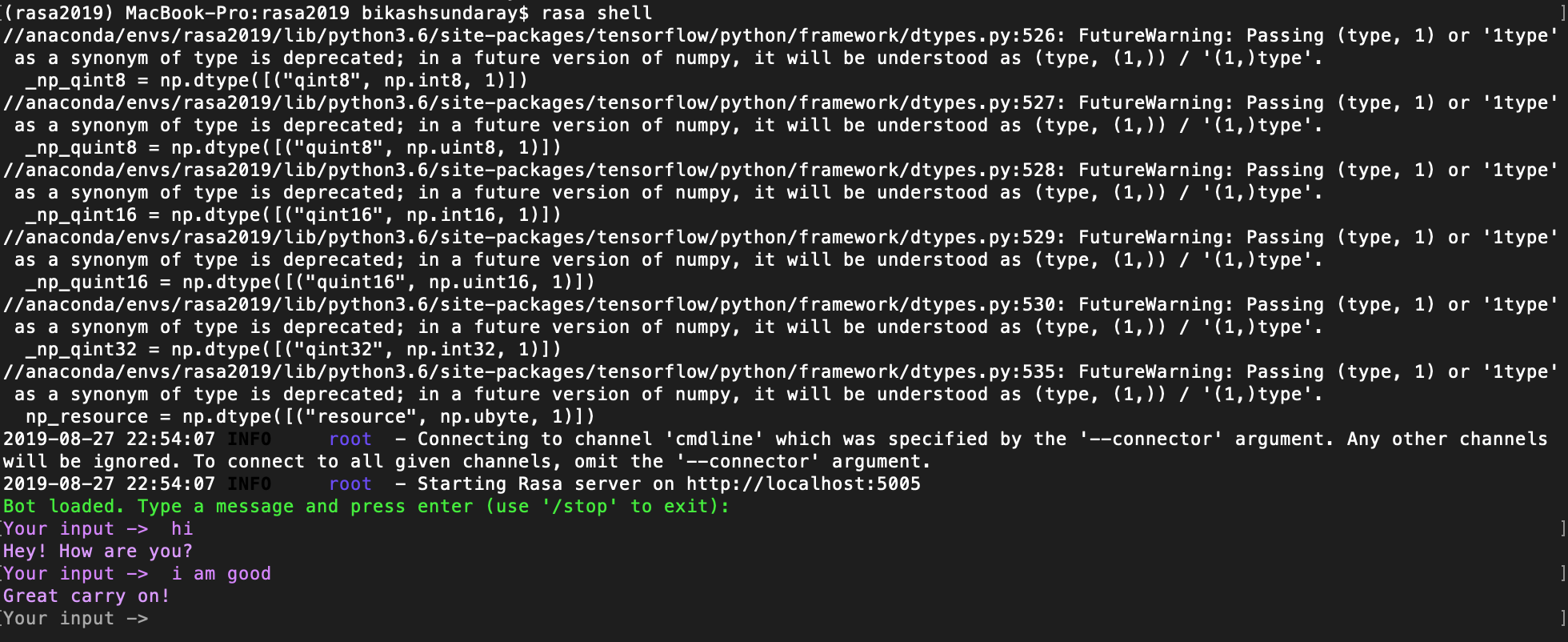


Figure 38: RASA Results

Rasa Bot Results:

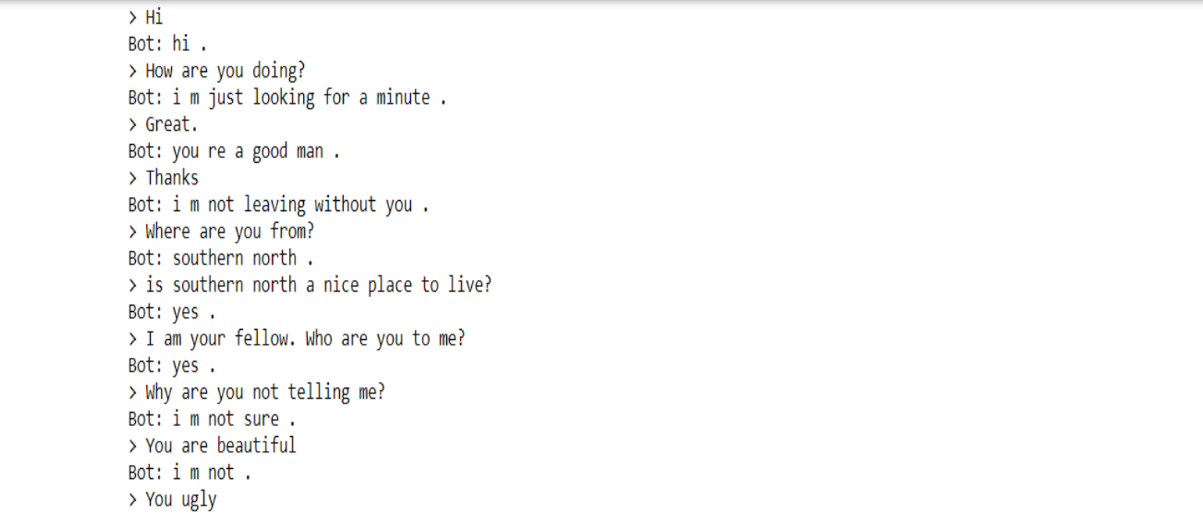


Figure 39: Rasa Results

## 6.3 Dialogue Flow Model deployed using APIs

### 6.3.1 Intents

Intents are the front line of any chatbot, they be the verb. Also, the working word or phrase, the intention of the user conversing with your chatbot.

Hence the first action you will perform is to detect what the intention of the user is. Creating intents in DIALOGFLOW is top of the list, for good reason. First you can name your intent and add training phrases. The training phrases will be used by DIALOGFLOW to train its cognitive model [29].

### 6.3.2 Entities

If intents are the verbs, entities are the nouns. These are the key elements you want to capture form user input.

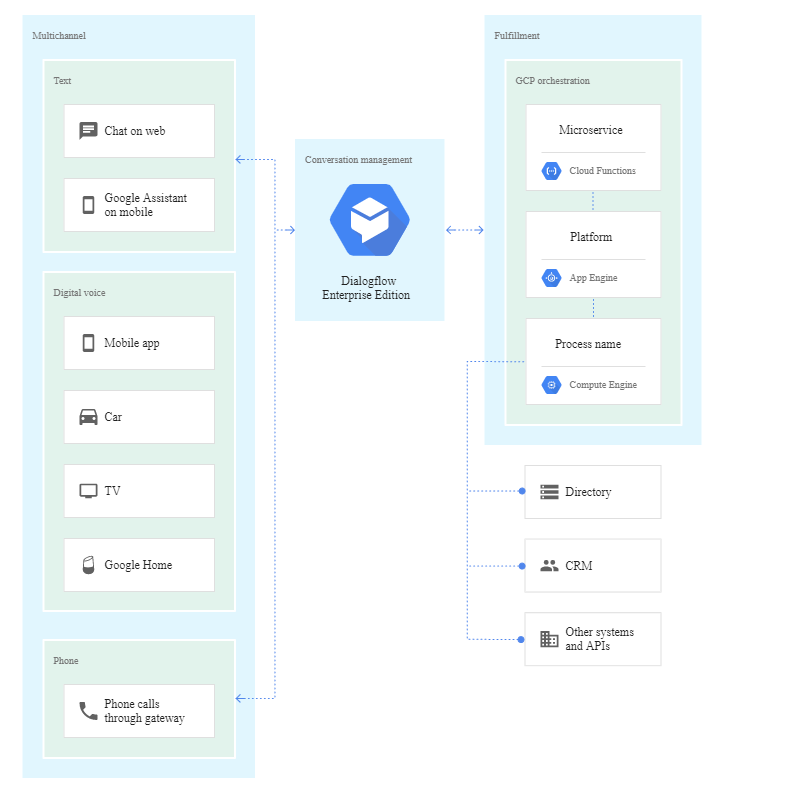


Figure 40: DialogFlow Model

### 6.3.3 Context

Context can be managed by making use of DIALOGFLOW as merely an API and performing this within your own framework. But also, you can set the context within the intents.

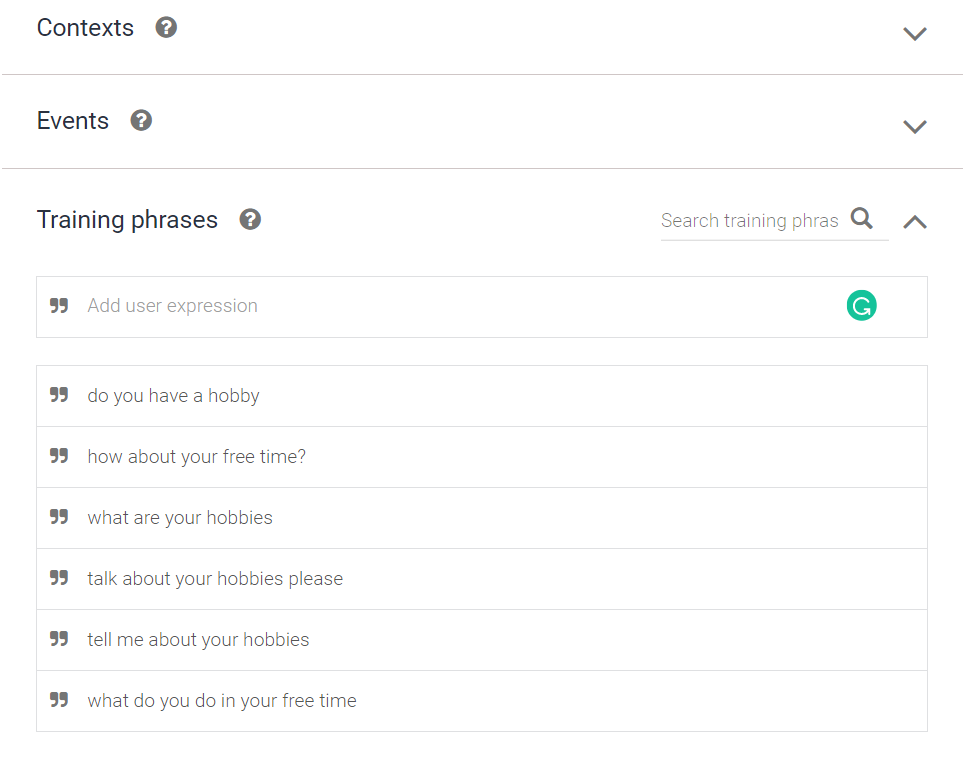


Figure 41: Setting COntexts

### ****6.3.4 Agents****

Agents translate user requests to actionable data i.e. intents. It is essentially a module within dialog flow which incorporates Natural Language Processing to understand what the user meant and to figure out what “action” must be carried out.

Agents manage conversations with the user through intent, entities, contexts and other building blocks [27].

### 6.3.5 Small Talk

Your agent can learn how to support small talk without any extra development. By default, it will respond with predefined phrases. Use the form below to customize responses to the most popular requests [18].

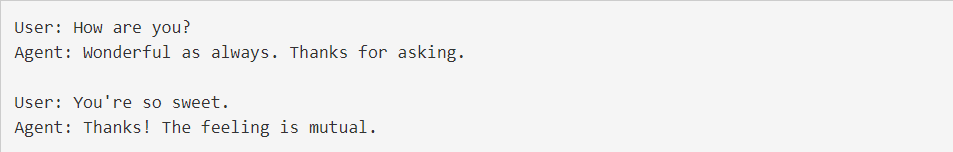


Figure 42: Small Talk

### 6.3.6 Training Phrase:

Phrases you can expect from the user that will trigger the intent.

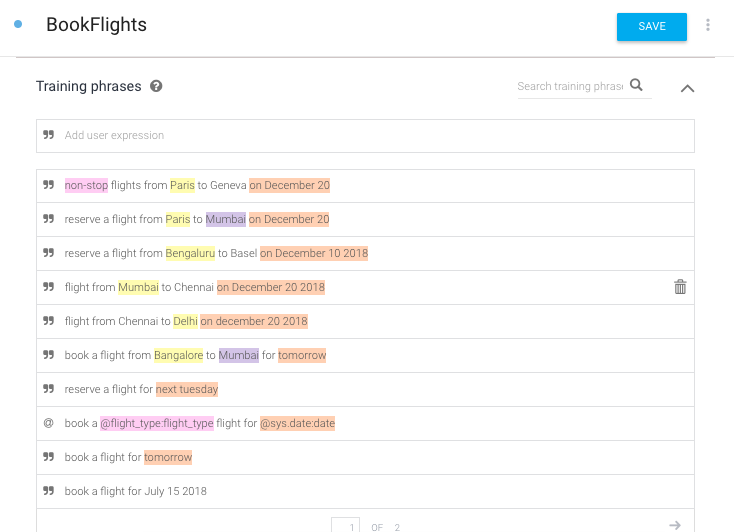


Figure 43: Training Phrases

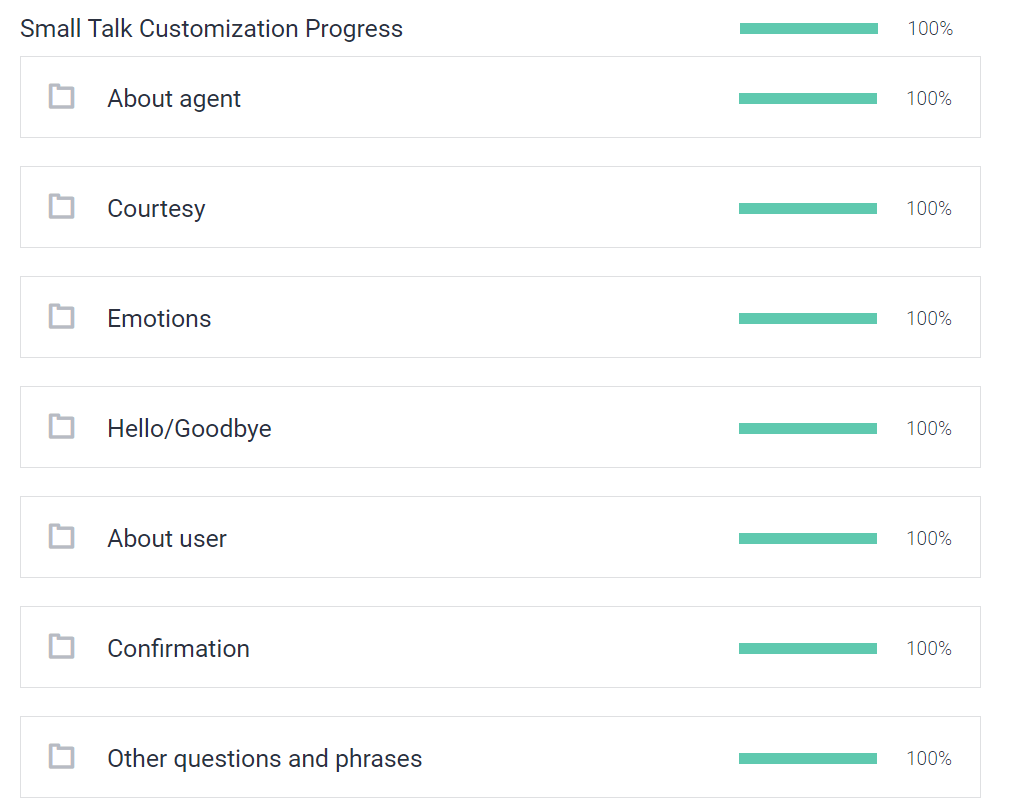


Figure 44: Smart Talk Training

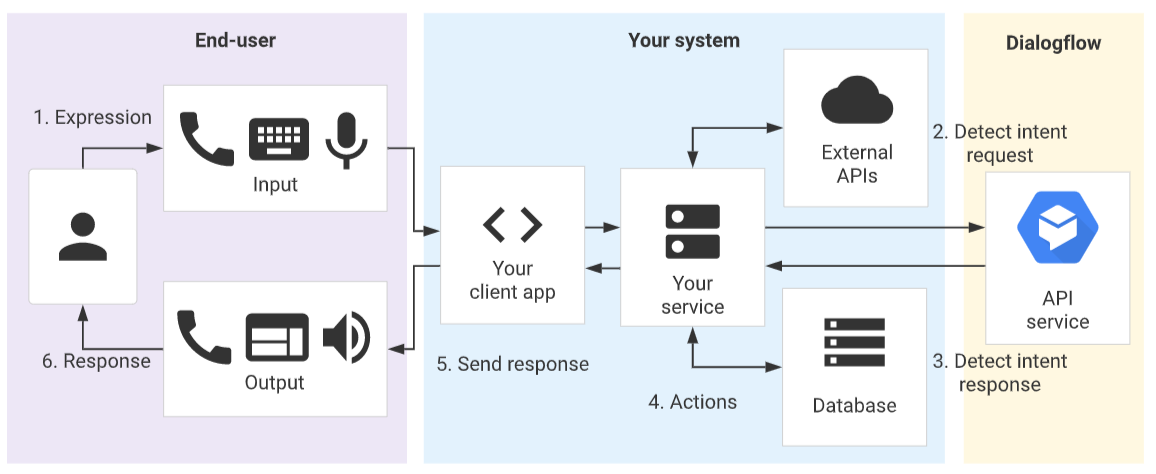


Figure 45ENd User API

For using DIALOGFLOW as an API, the sequence of events be follows:

The end-user types or speaks an expression.

Your service sends this end-user expression to DIALOGFLOW in a detect intent request message.

DIALOGFLOW sends a detect intent response message to your service. This message contains information about the matched intent, the action, the parameters, and the response defined for the intent.

Your service performs actions as needed, like database queries or external API calls.

Your service sends a response to the end-user.

The end-user sees or hears the response.

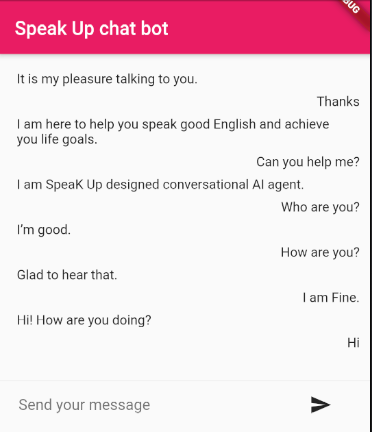


Figure 46: DialogFlow Results

## Visualizing Graphs and Attention

Be careful, we have a lens through which we can see how MODEL is a mixture of language representations.

The following visualization shows the focus of a sample input text. This view visualizes attention as lines which connect the word updated (left) to the word (right), as mentioned above. The strength of color represents the weight of attention; weight above one reveals rather dark lines, while weights similar to zero look like light lines, or are simply not apparent. The consumer may only stress a phrase from this term. This visualization is called the attention-head view for reasons discussed later. It is based on the excellent [Tensor2Tensor visualization tool](https://github.com/tensorflow/tensor2tensor/tree/master/tensor2tensor/visualization) from [Llion Jones](https://medium.com/@llionj).

### Multi-head attention

The visualization above shows one mechanism of attention in the model. In fact, MODEL has learned multiple mechanisms of attention, named heads, which work side by side. As we shall soon see, multi-head attention enables the model to identify a wider range of connections between the words than a single attention mechanism might be possible.

MODEL stacks have several attention layers, each of which operates on the layer performance that had previously been created. Thanks to this repeated form of embedded word, MODEL can form rich depictions as it reaches the depths of the model.

Since the heads of attention do not share parameters, each head has a unique pattern of attention. The MODEL version we are considering — MODEL Base — consists of 12 layers and 12 heads, resulting in 12 x 12 = 144 different mechanisms of attention. The model view (interactive form available here) lets us visualize attention in all heads at once:

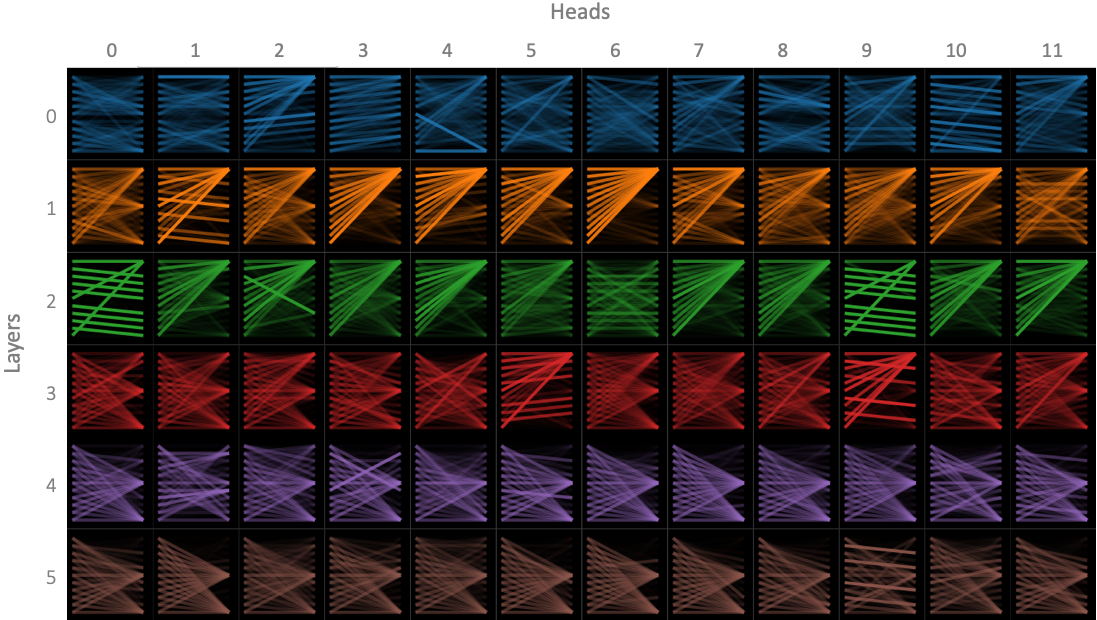


Figure 47:Head and Layers

For a specific head (indexed by column), in a certain layer (indicated by row), each cell in the model view shows a thumbnail form of a focused view from an earlier point of view. The patterns of focus are unique to the input text (the same in this case as the feedback to the above view). We can see from the model view that MODEL produces a range of patterns of attention. We should analyze how MODEL can generate these varied trends in this second part. In this post.

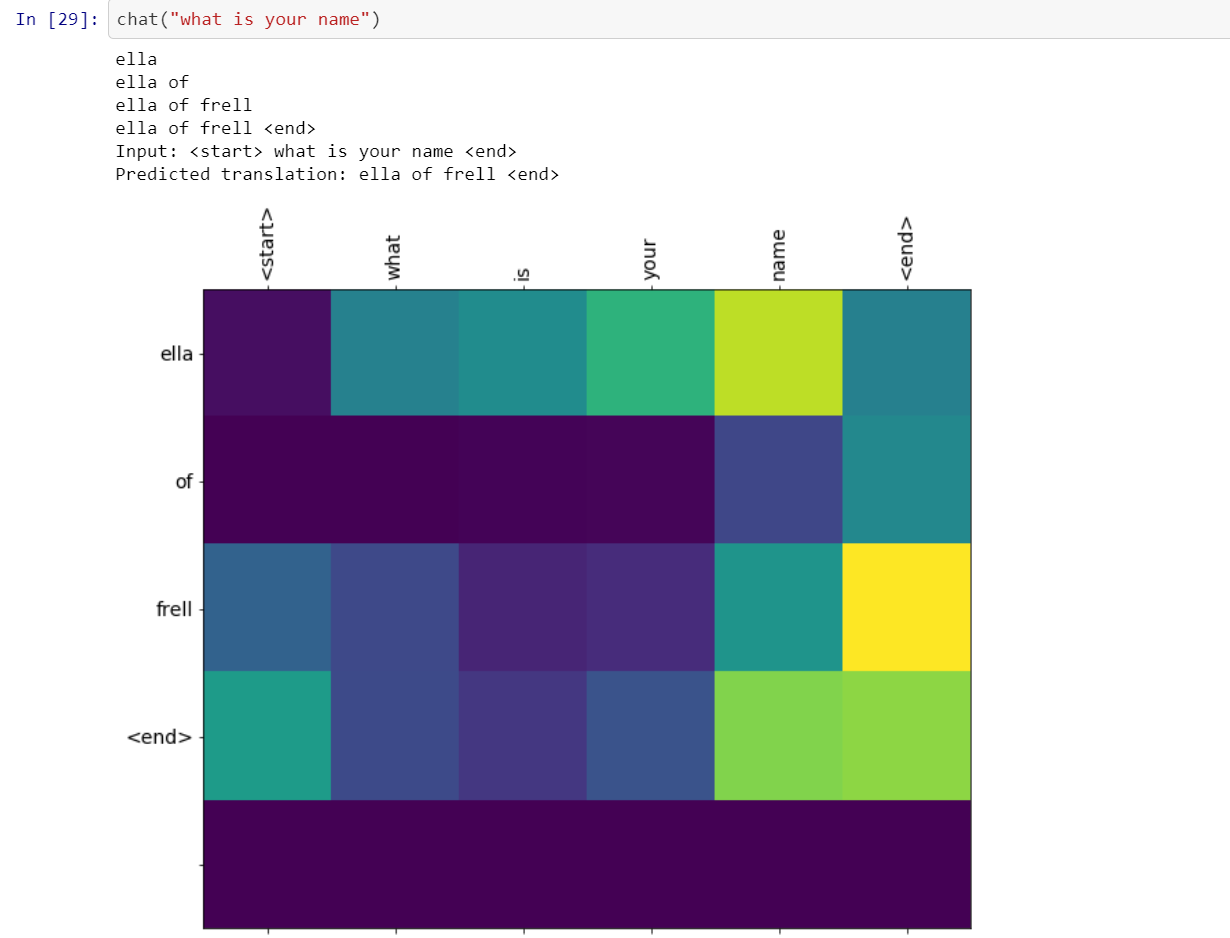


Figure 48Graph :What is your name?

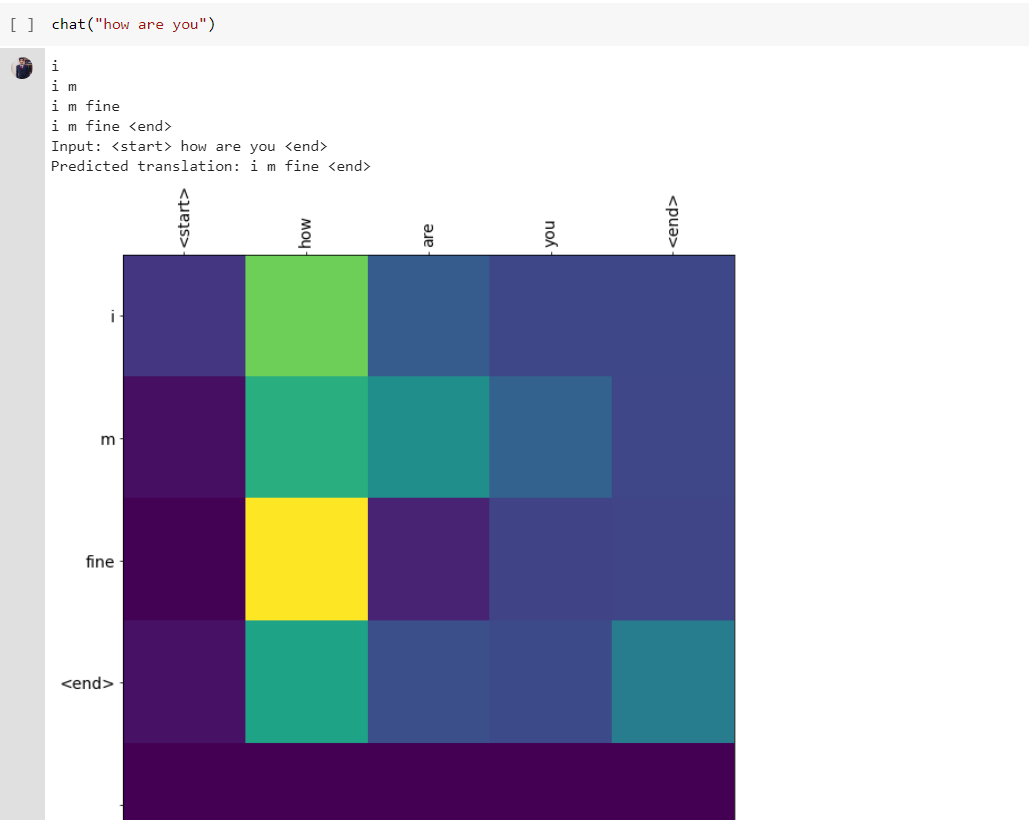


Figure 49Graph :How are you?

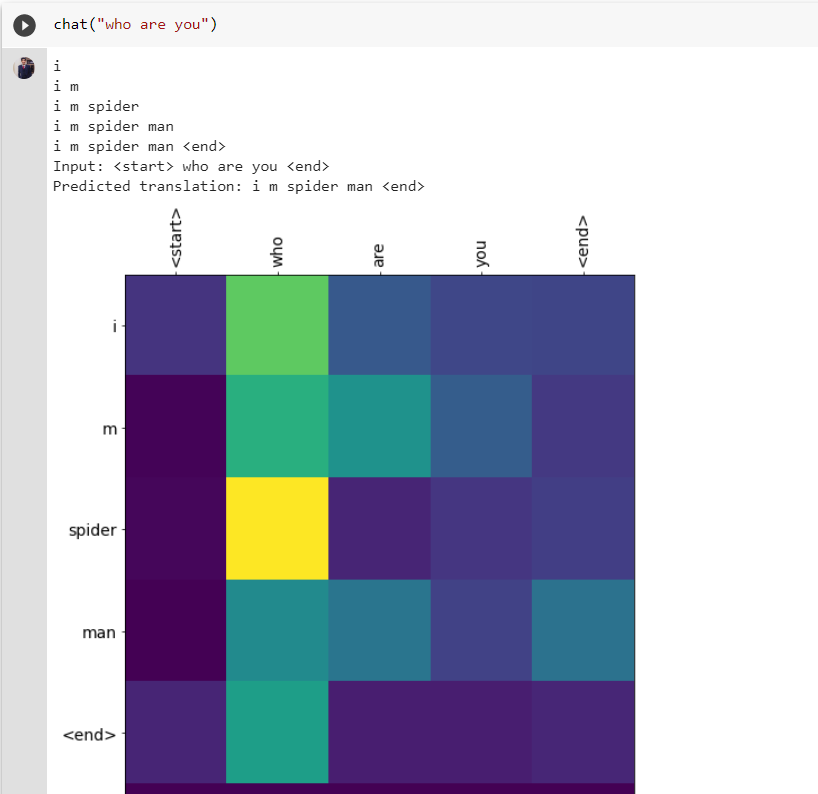


Figure 50Graph: Who are you?

# COMPARISON BASED ON DIFFERENT MODELS RESULT

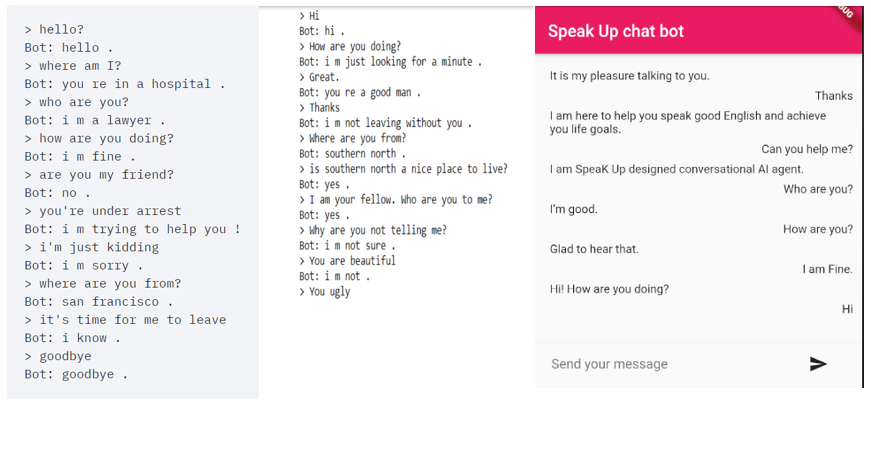
**DL Pytorch Model RASA Model DialogFlow Model**

Figure 51: Comparing Results

This is clearly shown that we have achieved some good and conversable results from Dialog Flow Chat Bot implementation so we are currently deploying it in our mobile application with further improvement due as future work, and we will be working on it to publish a results based research paper.

# CONCLUSION AND FUTURE WORK

Our proposed solution has been thought of, and carried out, by state-of-the-art technologies. Our research led us to develop a solution which caters to our problem statement. We wanted to generate semantically, and syntactically correct sentences based on the word(s) provided by the user. Our solution performs this task effectively, and as showcased in our findings; we are achieving great results for our model through intrinsic evaluation.

Our future goal is to implement dictionary for new vocabulary so that user can increase his knowledge base. Moreover, we will also implement correction of grammatical and spelling errors, writing practices and lessons, webinars, tuitions and online discussions with fellow learners and regular written and spoken assessments. Lastly, we will also measure progress using score so that user can keep track of his progress.

Further improvements to the Language Models could include making these models using Variational Autoencoders (VAE’s). VAE’s could provide a groundbreaking way of sampling and creating new sentences from the kernel space and allow even more flexibility in the constraint-based generation domain.

# REFERENCES:

[1] <http://www.savap.org.pk/journals/ARInt./Vol.4(6)/2013(4.6-13).pdf>

[2] <https://www.dawn.com/news/1051818>

[3] <https://defence.pk/pdf/threads/heres-how-pakistan-ranks-among-worlds-english-speaking-countries.461891/>

[4] <https://www.autismspeaks.org/science-news/cdc-increases-estimate-autismsprevalence->15-percent-1-59-children

[6] <https://www.autismspeaks.org/autism-facts-and-figures>

[7] [http://www.pbs.gov.pk/sites/default/files//tables/POPULATION%20BY%205%](http://www.pbs.gov.pk/sites/default/files//tables/POPULATION%20BY%205%25) 20YEAR%20AGE%20GROUPS%20-%20PAKISTAN.pdf

[8] <https://www.pakistantoday.com.pk/2018/04/02/11th-world-autism-awarenessday-observed-without-govt-support>

[9] https://www.speechpathologygraduateprograms.org/2017/11/top-10-aacaugmentative-and-alternative-communication-devices/ [18] https://machinelearningmastery.com/statistical-language-modeling-andneural-language-models

[10] https://towardsdatascience.com/learning-nlp-language-models-with-real-datacdff04c51c25

[20] <http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf>

[11] <https://pub.dev/flutter/packages>

[12] <https://flutter.dev/docs>

[13] <https://rasa.com/docs/>

[14] <https://pytorch.org/docs/stable/index.html>

[15] <https://dialogflow.com/docs?authuser=6>

[16] <https://patentimages.storage.googleapis.com/41/52/ff/c35d7fc78bf866/US6560983.pdf>

[17] <https://www.semanticscholar.org/paper/Perspectives-for-Evaluating-Conversational-AI-Jadeja-Varia/5ebde6580941d9e0d16ad8cc6ec78aae16912510>

[18] Goldfried MR, Greenberg LS, Marmar C. Individual psychotherapy: process and outcome. Annu Rev Psychol (1990)

[19] Miner AS, Milstein A, Hancock JT. Talking to machines about personal mental health problems. JAMA (2017)

[20] Luxton DD. Artificial intelligence in behavioral and mental health care. Elsevier/Academic Press; (2016).

[21] Richard James Burgess, The art of music production: the theory and practice. Oxford University Press 2013.

[22] www.deepai.org/machine-learning-glossary-and-terms/conversational-agent.

[23] Howard Gardner. 1987. The theory of multiple intelligences. Annals of dyslexia (1987),

[24] <https://www.kdnuggets.com/2019/04/ai-conversational-applications.html>

[25] Deep learning approaches for question answering system, International Conference on Computational Intelligence and Data Science (2019)

[26] Thurlow C L, Lengel L and Tomic A, Computer Mediated Communication: Social Interaction and the Internet, Thousand Oaks, CA: Sage, 2004

[27] Shevat A 2017 Designing bots: creating conversational experiences (O'Reilly Media, Inc)

[28] Abdul-Kader S A and Woods J C, Survey on chatbot design techniques in speech conversation systems International Journal of Advanced Computer Science and Applications, 2015

[29] Azwary F, Indriani F and Nugrahadi D T, Question Answering System Berbasis Artificial Intelligence Markup Language Sebagai Media Informasi KLIK-KUMPULAN JURNAL ILMU KOMPUTER, 2016