HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and Communication Technology

**THESIS**

Apply Machine Learning in Banking Chatbot with RASA Framework

Nguyen Quang Minh

[minh.nq152472@sis.hust.edu.vn](mailto:minh.nq152472@sis.hust.edu.vn)

**Supervisor**: Dr. Nguyen Kiem Hieu

Signature

**Department**: Department of Information Systems

**Institute**: School of Information and Communication Technology

### 

Hanoi, May 2020

**REQUIREMENTS OF THESIS**

Student name: Nguyen Quang Minh

Phone Number: +84829303597

Email: [minh.nq152472@sis.hust.edu.vn](mailto:minh.nq152472@sis.hust.edu.vn)

Class: ICT-K60

Program: Fulltime

**Goal of the Thesis:**

Implement a Chatbot System that can automatically assist customers with a certain set of Online Banking Functions

**Main Tasks:**

* Collect and prepare the dataset as well as designing the scripts for the Chatbot conversation flow
* Conduct Testing for Natural Language Processing tasks with different components to select the best performance one for the system
* Provide an Interface for the Chatbot to interact with customers

**Declaration:**

I – Nguyen Quang Minh – hereby warrants that the Work and Presentation in this thesis are performed by myself under the supervision of Dr. Nguyen Kiem Hieu. All results presented in this thesis are truthful and are not copied from any other work. All references in this thesis - including images, tables, figures, and quotes - are clearly and fully documented in the bibliography. I will take full responsibility for even one copy that violates school regulations.

|  |  |
| --- | --- |
|  | Hanoi, June 2020  Author  Nguyen Quang Minh |

Attestation of the supervisor on the fulfillment of the requirements of the thesis

|  |  |
| --- | --- |
|  | Hanoi, June 2020  Supervisor  Nguyen Kiem Hieu |

# Acknowledgement

I would like to thank my supervisor Dr. Nguyen Kiem Hieu for his guidance and his invaluable comments and contributions. Without his initiation and suggestion this thesis could never have been formed. I also want to send my appreciation to my colleague Bui Van Trung, who gives me countless reviews and encourages me consistently during the process. I am more than grateful to receive unconditional support and guidance from them.

For my friends and lecturers during my 5-year span in Hanoi University of Science and Technology, I am humble and grateful for the knowledge I learn from each and all of you, the connections and the memories I had and will cherish for the lifetime. This University has given me more than I ever asked for and it shapes the person I am today.

For my family, you are the backbone that gives my life structure. I can never make it if it is not for your consistent, tireless support and encouragement. Thank you for pushing me but at the same time giving me room to figure things out by myself. I am forever thankful for it.

# Abstract

In recent years, Natural Language Processing has achieved some marvelous milestones in performance thanks to the arrival of powerful models. Among those milestones, Transformers was considered the ground-breaking model when it was released and created a foundation for later powerful language models. Chatbot building, in its core, took great advantage from this leap and now be able to hold a more fluent conversation and extract better information from users’ intentions. In terms of banking, the need to apply chatbot in their ecosystem is a growing trend. In an ideal world, a chatbot can play the role of virtual assistance and help users execute every task, related to Banking in this case, through the form of texting or voice recording. Among many Chatbot solutions, the RASA framework stands out as an open-source platform with great documentation, stacked with features, frequently updated with the latest technologies and have a strong and active community to support developers. For those reasons, I choose to dig deep into its ecosystem and technology to create a Chatbot that suites such a specific domain as Banking. My Chatbot delivered the most essential use cases while applying Transformers to greatly improve the performance compared to previous versions. It also contains the potential to improve performance by using the data collected from users after being launched and applying language models dedicated to Vietnamese in the future. In the scope of this thesis, I would like to draw an overall picture of RASA framework architecture, as well as the recent features in the new version and how I apply it in my use cases. Also, the theory of Transformers is covered to provide readers with the basic concept of this beautiful model.

Contents

[Acknowledgement 4](#_Toc44041871)

[Abstract 4](#_Toc44041872)

[List of Figures 8](#_Toc44041873)

[List of Equations 10](#_Toc44041874)

[List of Tables 11](#_Toc44041875)

[List of Acronyms 12](#_Toc44041876)

[Chapter 1: Introduction 13](#_Toc44041877)

[1.1. Problems 13](#_Toc44041878)

[1.2. Goals and Scopes 13](#_Toc44041879)

[1.3. Suggested solutions 14](#_Toc44041880)

[1.4. Thesis Structure 14](#_Toc44041881)

[Chapter 2: Requirements Breakdown 16](#_Toc44041882)

[2.1. Current State’s Survey 16](#_Toc44041883)

[2.2. The Overall Use Case Diagram 17](#_Toc44041884)

[2.3. Decomposition/Breakdown Functions 18](#_Toc44041885)

[2.3.1. Transfer 18](#_Toc44041886)

[2.3.2. Top Up Phone 21](#_Toc44041887)

[2.3.3. Frequently Asked Questions (FAQ) Use Case 24](#_Toc44041888)

[2.3.4. Function Search Use Case 26](#_Toc44041889)

[Chapter 3: Technology 28](#_Toc44041890)

[3.1. Artificial Intelligence 28](#_Toc44041891)

[3.1.1. Natural Language Processing 28](#_Toc44041892)

[3.1.2. Machine Learning 28](#_Toc44041893)

[3.1.3. Deep Learning 30](#_Toc44041894)

[3.2. Chatbot – Rasa Framework 40](#_Toc44041895)

[3.2.1. Architecture 40](#_Toc44041896)

[3.2.2. NLU 40](#_Toc44041897)

[3.2.3. Core 42](#_Toc44041898)

[3.2.4. Features Used 47](#_Toc44041899)

[3.3. Interface 48](#_Toc44041900)

[3.3.1. Page-scoped ID (PSID) 49](#_Toc44041901)

[3.3.2. Webhook 49](#_Toc44041902)

[3.3.3. WebView 50](#_Toc44041903)

[3.3.4. Web Plug-ins 50](#_Toc44041904)

[Chapter 4: Development 51](#_Toc44041905)

[4.1. Architecture 51](#_Toc44041906)

[4.2. Tools Used 55](#_Toc44041907)

[4.3. Testing + Results 56](#_Toc44041908)

[4.3.1. Data Overview 56](#_Toc44041909)

[4.3.2. Results 57](#_Toc44041910)

[4.4. Deployment 65](#_Toc44041911)

[4.4.1. Tracker Store 65](#_Toc44041912)

[4.4.2. Facebook Messenger Interface 67](#_Toc44041913)

[Chapter 5: Contribution 72](#_Toc44041914)

[Chapter 6: Conclusion and Future Development 75](#_Toc44041915)

[Bibliography 76](#_Toc44041916)

# List of Figures

[Figure 1: Overall Use Case 14](#_Toc44031154)

[Figure 2: Transfer Use Case Decomposition 14](#_Toc44031155)

[Figure 3: Transfer Activity Diagram 17](#_Toc44031156)

[Figure 4: Top Up Phone Activity Diagram 19](#_Toc44031157)

[Figure 5: FAQ Activity Diagram 21](#_Toc44031158)

[Figure 6: Function Search Activity Diagram 23](#_Toc44031159)

[Figure 8: Transformer Architecture (Source: [2]) 31](#_Toc44031160)

[Figure 9: DIET Architecture (source: [3]) 33](#_Toc44031161)

[Figure 10: Rasa Stack Including NLU and Core (source: [3]) 35](#_Toc44031162)

[Figure 11: Progress Flow Diagram (source: [3]) 39](#_Toc44031163)

[Figure 12: Overall Architecture 45](#_Toc44031164)

[Figure 13: Overall Packages 45](#_Toc44031165)

[Figure 14: Messenger Class 46](#_Toc44031166)

[Figure 15: Rasa Response Class 47](#_Toc44031167)

[Figure 16: Actions Classes 48](#_Toc44031168)

[Figure 17: Chatito Interface 50](#_Toc44031169)

[Figure 18: Chatito Settings when saving generated data 50](#_Toc44031170)

[Figure 19: Confusion Matrix for Conversation Management 54](#_Toc44031171)

[Figure 20: Create database in MongoDB 55](#_Toc44031172)

[Figure 21: Create User in MongoDB 56](#_Toc44031173)

[Figure 22: Configuration for Database in Rasa 56](#_Toc44031174)

[Figure 23: Robo3t Interface when Tracker successfully stored 57](#_Toc44031175)

[Figure 24: Step 1 in Creating Facebook Page (source: [4]) 58](#_Toc44031176)

[Figure 25: Step 2 in Creating Facebook Page (source: [4]) 58](#_Toc44031177)

[Figure 26: Configuration Components in Facebook Application (source: [4]) 58](#_Toc44031178)

[Figure 27: Link the Facebook Page to Messenger (source: [4]) 59](#_Toc44031179)

[Figure 28: Generate Token after Successful Linking (source: [4]) 59](#_Toc44031180)

[Figure 29: Settings for App Secret (source: [4]) 60](#_Toc44031181)

[Figure 30: Retrieving App Secret (source: [4]) 60](#_Toc44031182)

[Figure 31: Configure Webhook (source: [4]) 61](#_Toc44031183)

[Figure 32: Public Localhost using Ngrok 61](#_Toc44031184)

[Figure 33: Messenger Quick Replies Buttons 62](#_Toc44031185)

# List of Equations

Equation 1: Self-Attention Output Vector 25

Equation 2: Weight between two vectors 25

Equation 3: key, value and query from input vector 27

Equation 4: Weight from key and value 27

Equation 5: Final Output Vector 27

Equation 6: Positional Encoding 28

# List of Tables

Table 1: Transfer Function Use Detailed Description 19

Table 2: Transfer Function Main Script 20

Table 3: Top Up Phone Use Case Detailed Description 22

Table 4: Top Up Phone Main Script 22

Table 5: FAQ Use Case Detailed Description 24

Table 6: FAQ Main Script 24

Table 7: Function Search Use Case Detailed Description 26

Table 8: Function Search Main Script 26

Table 9: Parameters in Messenger Webhook (source: [4]) 50

Table 10: Tools Used 55

Table 11: Data Overview 56

Table 12: Stories Designed for Core 56

Table 13: DIET Intent Detection Result 58

Table 14: Sklearn Intent Detection average performance 60

Table 15: DIET Intent Detection with BERT Result 61

Table 16: DIET Intent Detection average performance 61

Table 17: Sklearn Intent Detection average performance 61

Table 18: DIET Intent Detection with BERT average performance 62

Table 19: Entities Extraction Result 62

Table 20: Entities Extraction Average Performance 63

Table 21: Response Selector Result 63

# List of Acronyms

API Application Programming Interface

ML Machine Learning

DL Deep Learning

SVM Support Vector Machine

DIET Dual Intent Entity Transformer

FAQ Frequently Asked Questions

PSID Page Scoped ID

NLU Natural Language Processing

BERT Bidirectional Encoder Representations from Transformers

# Chapter 1: Introduction

## 1.1. Problems

Customer Serving has always been an important aspect in the Service Industry. With the popularity of Smartphone usage, the platform has been delivered for providing better services. In terms of Banking, it has almost become vital for every bank joining the business to develop its own application for Mobiles, hence the whole new service Mobile Banking. Over the years, banks have stacked their applications with more and more features and now almost any service provided by banks can be executed just by tapping on the phone’s screen. This comes with a consequence. Stacking the application with too many features will make it too bulky, user experience (UX) becomes a burden. Moreover, every bank has its own customization of interface, that makes it even harder to unify the experience in case a customer change to another bank or own more than one banking account at once. They need a more robust way to deliver all these services to users.

Holding a conversation by texting has always been a traditional way when it comes to phone usage. It exists before the Smartphone’s era and has proven that it is here to stay since the smartphone comes only to deliver even more platforms to make texting between people easier. So, it is safe to say users are familiar with this form of communication. By delivering the service through texting, users can start using the application right away without getting used to the interface. It does not come as a replacement to the whole interface banks have developed and customized, but as another approach to help users access functions easier. The term of virtual system that can hold a conversation with users automatically is called Chatbot. It functions independently without constant supervision from a human being. The Chatbot ability to hold a conversation frequently and to extract the correct information from users is achieved thanks to the advance of Natural Language Processing researches.

## 1.2. Goals and Scopes

From the existing need of Banking, the Chatbot I develop will deliver some of the most basic functions based on the pre-designed scripts and requirements from Banking services. The goal is to extract the correct information from users in order to help them execute the transaction and be able to hold a conversation in a fluent fashion and handle exceptions effectively.

In order to achieve this goal, the Chatbot needs to:

* Thoroughly acknowledge the domain requirements to construct the scripts for collecting information in each use case.
* Understand correctly inputs from users - be able to predict users’ intention, extract information and responds coherently.
* Have a basic interface that can provide robust and handy usage.

## 1.3. Suggested solutions

With the requirements above for the Chatbot to meet, I will break down the guidelines to develop the Chabot to fulfill these functions.

* Investigate the existed functions on mobile applications, discuss with the business team to come up with different scenarios for the Chatbot to handle.
* Collect logged data, generate simulated examples based on the scripts designed above to enhance the existing data, synthesize and unify from all sources to finalize the dataset.
* Apply Machine Learning and Deep Learning methods for tasks related to language understanding such as predicting users’ intentions or extracting information from sentences. These methods are provided by Rasa Framework.
* Construct a mechanics to manage the Chatbot actions throughout the process of the conversation. These rules and features to control chatbot actions are also supported by Rasa Framework.
* In order to have a robust and friendly interface for demonstration, I adopt the Messenger Platform to deploy the Chatbot.

## 1.4. Thesis Structure

The outline of the thesis can be divided into 4 chapters:

Chapter 1: Introduction

* Main challenges in Banking Chatbot and suggested Solutions

Chapter 2: Requirements Breakdown

* All the Use Cases are presented and analyzed in this chapter.

Chapter 3: Technology

* Breakdown all the theory of technology used in the scoped of the project: from the Deep Learning Architecture to the Rasa Framework and the Facebook Messenger Interface

Chapter 4: Development

* Present the project architecture, tools and libraries used along with the final testing result and Deployment

Chapter 5: Contribution

Chapter 6: Conclusion and Future Development

# Chapter 2: Requirements Breakdown

## 2.1. Current State’s Survey

The need for Chatbot to automate the customers’ supporting process in many fields of the Service Industry is growing larger as it would cut down costs significantly compared to using human resources. In Banking specifically, nowadays almost every bank has developed its own mobile banking application. The challenge now is to raise the application users’ rate up as not every banking customer is interested in registering for the Mobile Banking Service. The reason might start from the customers’ habit of using cash, or their hesitation to trust the consistency of online services. It would be more convincing to persuade users to use the application if the Banking business can deliver a more reliable and easier to use service. So, there is a requirement to build a virtual assistance in the form of Chatbot to help customers use application more conveniently, or support users executing Banking services right on social message platforms. The demand is real for the Chatbot application in the business, coexist with the challenges to make the Chatbot operate smoothly, collect the correct information when users provide it and handle the many exceptions from users effectively. In every Banking service, there are certain fields, or pieces of information that are required to be filled from users in order to execute a successful service. The biggest mission of the Chatbot is to collect all this information from users correctly so the security check can be performed next then the process can be finalized.

## 2.2. The Overall Use Case Diagram

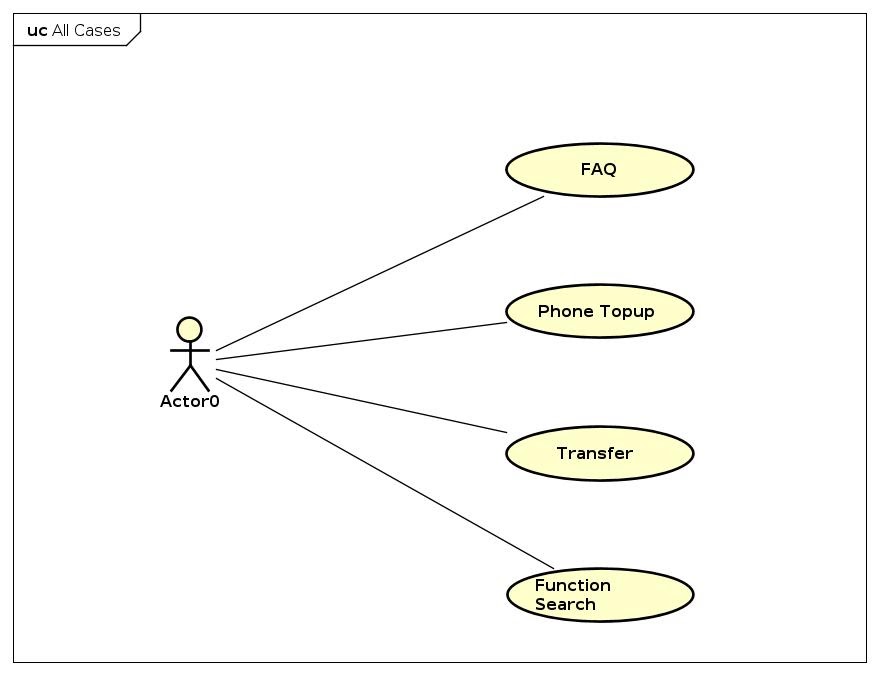


Figure 1: Overall Use Case

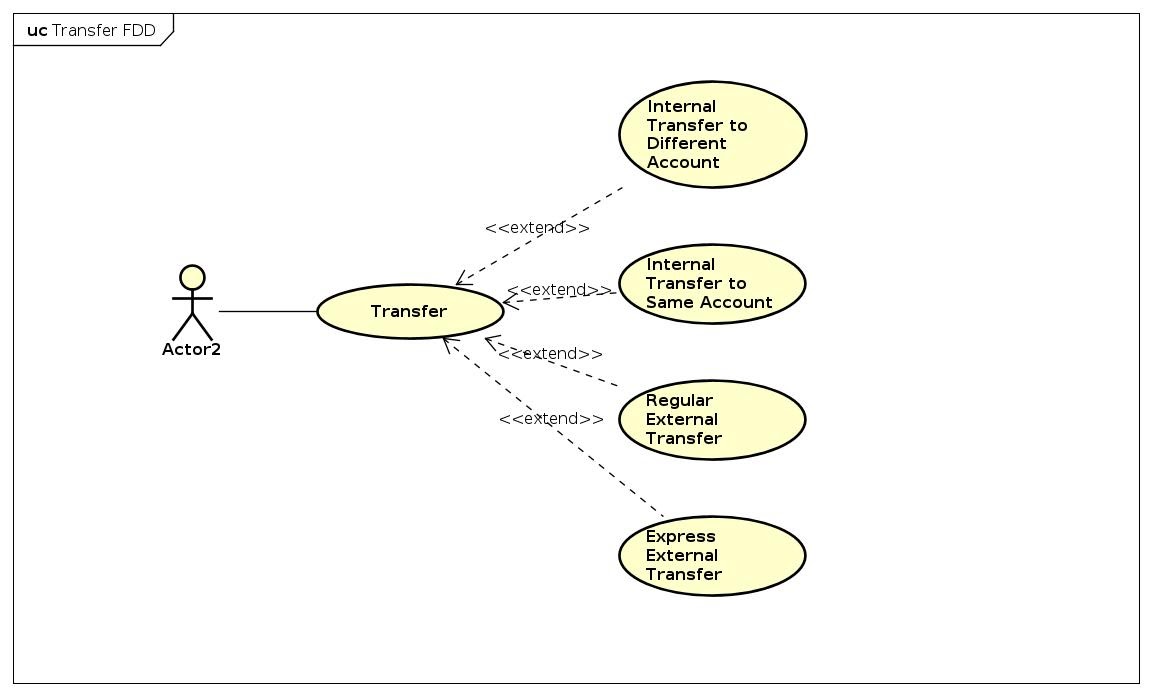


Figure 2: Transfer Use Case Decomposition

## 2.3. Decomposition/Breakdown Functions

### 2.3.1. Transfer

Participated Agents: Customers, AI Server, App Client, App Server, BOT.

Prerequisite Condition:

* + Customers successfully log in to Application/Message Platform
  + Bot has been developed to operate Transfer Function

Expected Result:

* + Bot successfully collects all the required information for Transaction to execute.

Logic Execution:

|  |  |
| --- | --- |
| Use Case ID | 1 |
| Use case | Transfer Money |
| Agent | Customer |
| Description | Support customer transfer money from his/her account to another account |
| Activation | Customer starts sending messages with Chatbot on Facebook Messenger |
| Prerequisites | Customer successfully logged in to Facebook |

Table 1: Transfer Function Use Detailed Description

|  |  |
| --- | --- |
| User | Chatbot |
| User sends messages on Facebook Messenger |  |
| User messages Chatbot with intention related to Transfer | Chatbot activates the Action related to Transfer and asks customers for specific Transfer Type |
| User choose a specific transfer type | Chatbot activates the Process Flow dedicated to Transfer Type user has chosen and asks user for destination account Number |
| User provides valid account number | Chatbot collects account number’s entity and moves on to ask customer the amount of money to transfer |
| User provides valid amount of transfer money | Chatbot collects the amount of transfer money entity and moves on to ask customer for the transfer message |
| User provides the transfer message | Chatbot collects the amount of transfer message entity and moves on the ask customer for the person to pay the transfer fee |
| User chooses the person to pay for the transfer fee | Chabot takes in the information from user, display all the information collected from the user then finish the collecting information process |

Table 2: Transfer Function Main Script

Unexpected Scenarios

* User provides Invalid Account Number. Chatbot is required to detect this misinformation and ask user to fill in again.
* User provides Invalid Amount of Money. Chatbot is required to detect this misinformation and ask user to fill in again.
* User wants to cancel the transaction midway. Chatbot must be able to escape the current flow and exit.

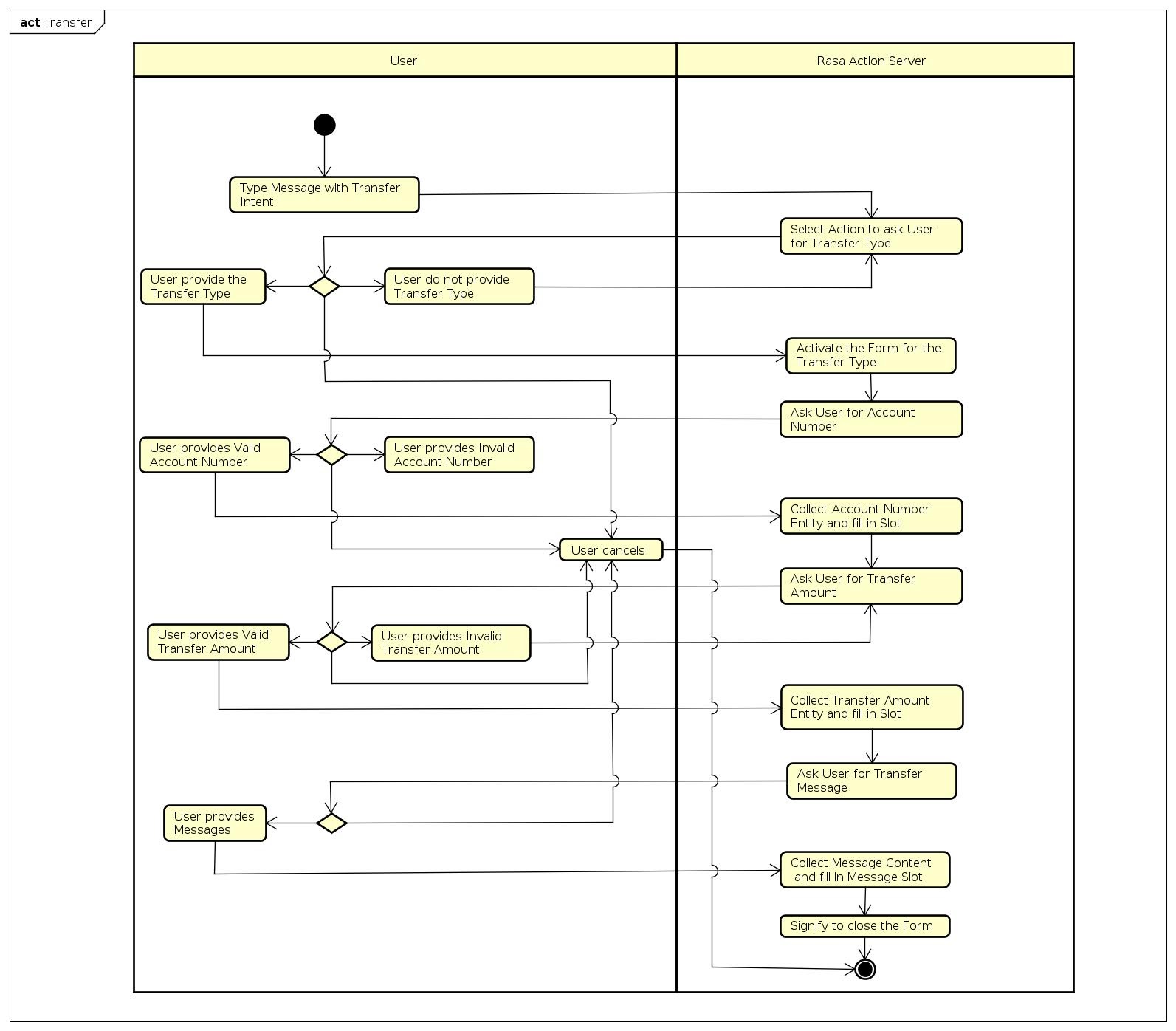


Figure 3: Transfer Activity Diagram

### 2.3.2. Top Up Phone

|  |  |
| --- | --- |
| Use Case ID | 2 |
| Use case | Top-up Phone |
| Agent | Customer |
| Description | Support customer top up his/her phone |
| Activation | Customer starts sending messages with Chatbot on Facebook Messenger |
| Prerequisites | Customer successfully logged in to Facebook |

Table 3: Top Up Phone Use Case Detailed Description

|  |  |
| --- | --- |
| User | Chatbot |
| User sends messages on Facebook Messenger |  |
| User messages Chatbot with intention related to Transfer | Chatbot activates the Process Flow dedicated to Top up use case and asks customers for destination Top Up Phone Number |
| User provides valid phone number | Chatbot collects phone number’s entity and moves on to ask customer the amount of money to top up |
| User provides valid amount of top up money | Chatbot collects the amount of top up money entity, display all the information collected from user then finish the process |

Table 4: Top Up Phone Main Script

Unexpected Scenarios

* User provides Invalid Phone Number. Chatbot is required to detect this misinformation and ask user to fill in again.
* User provides Invalid Amount of Top Up Money. Chatbot is required to detect this misinformation and ask user to fill in again.
* User wants to cancel the function midway. Chatbot must be able to escape the current flow and exit.

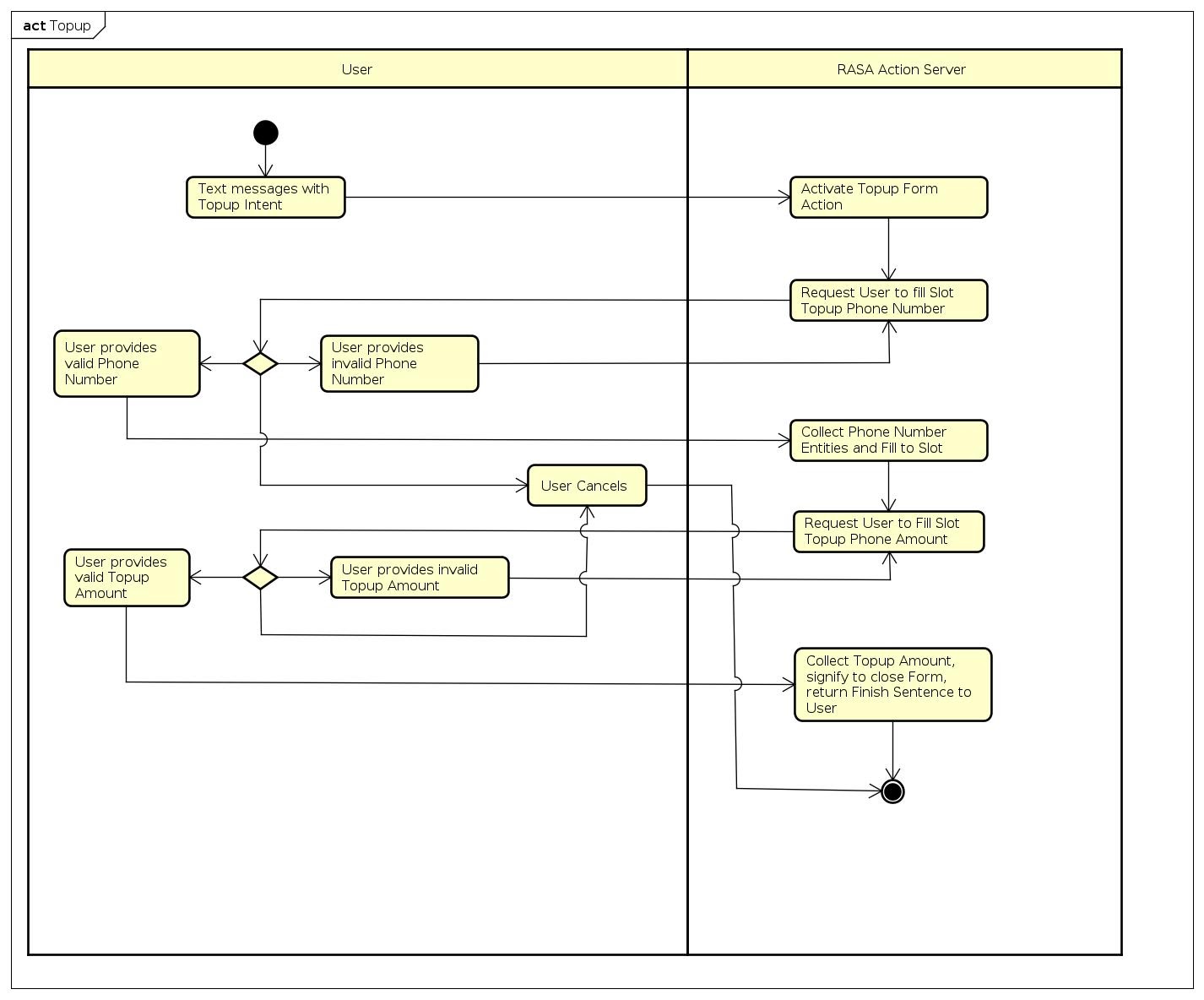


Figure 4: Top Up Phone Activity Diagram

### 2.3.3. Frequently Asked Questions (FAQ) Use Case

|  |  |
| --- | --- |
| Use Case ID | 3 |
| Use case | FAQ |
| Agent | Customer |
| Description | Support customer answer frequently asked questions |
| Activation | Customer starts sending messages with Chatbot on Facebook Messenger |
| Prerequisites | Customer successfully logged in to Facebook |

Table 5: FAQ Use Case Detailed Description

|  |  |
| --- | --- |
| User | Chatbot |
| User sends messages on Facebook Messenger |  |
| User messages Chatbot with intention related to FAQ | Chatbot activates the Process Flow dedicated to FAQ use case and gives out answer if the question is in the database |

Table 6: FAQ Main Script

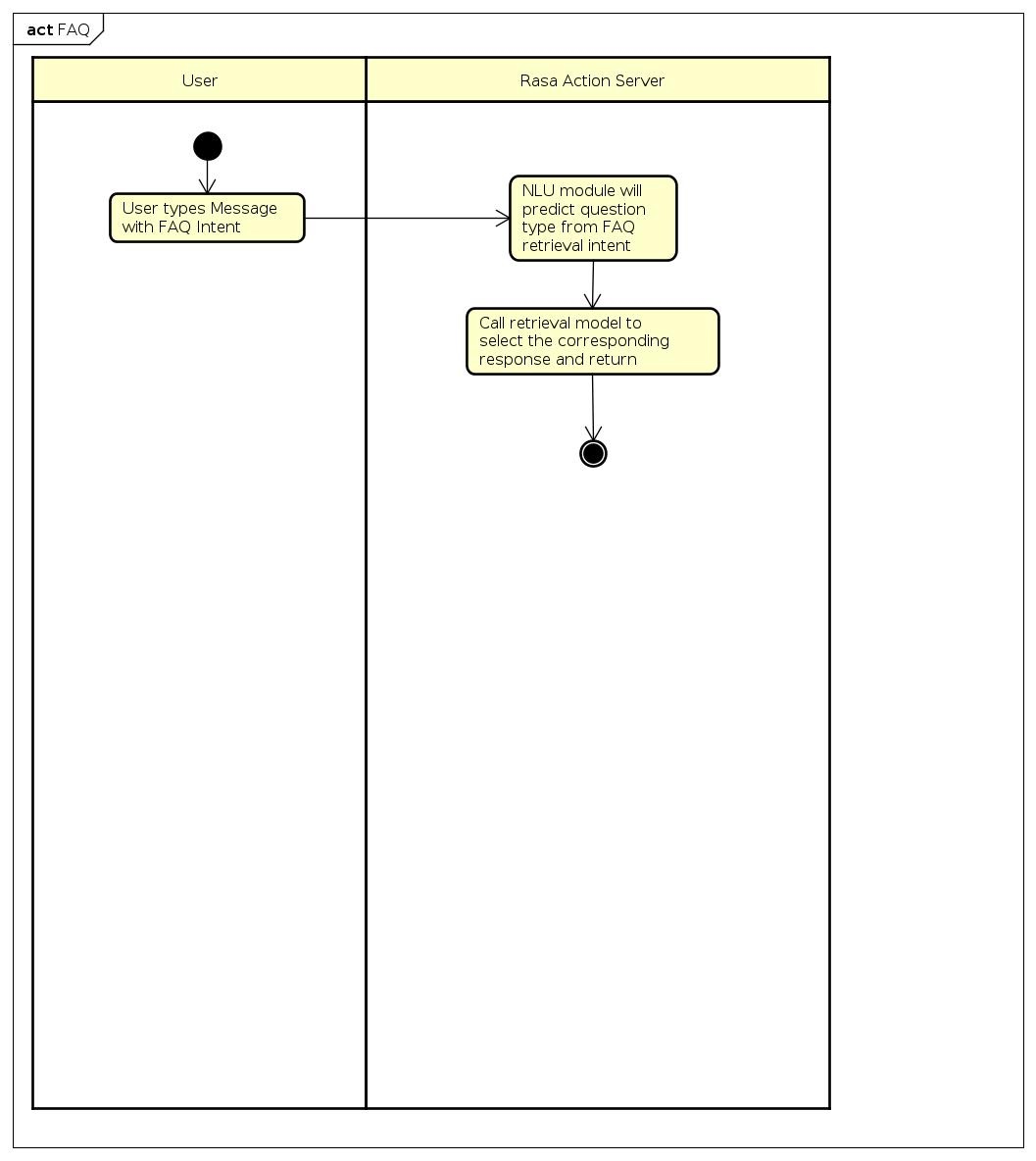


Figure 5: FAQ Activity Diagram

### 2.3.4. Function Search Use Case

|  |  |
| --- | --- |
| Use Case ID | 4 |
| Use case | Search for Function |
| Agent | Customer |
| Description | Support customer find the functions in the application if provided |
| Activation | Customer starts sending messages with Chatbot on Facebook Messenger |
| Prerequisites | Customer successfully logged in to Facebook |

Table 7: Function Search Use Case Detailed Description

Note: This Use Case suits the case when the Chatbot is applied in the Mobile Banking Application.

|  |  |
| --- | --- |
| User | Chatbot |
| User sends messages on Facebook Messenger |  |
| User messages Chatbot with intention related to Function Search | Chatbot activates the Action dedicated to Function Search use case and help users forward to the function’s window |

Table 8: Function Search Main Script

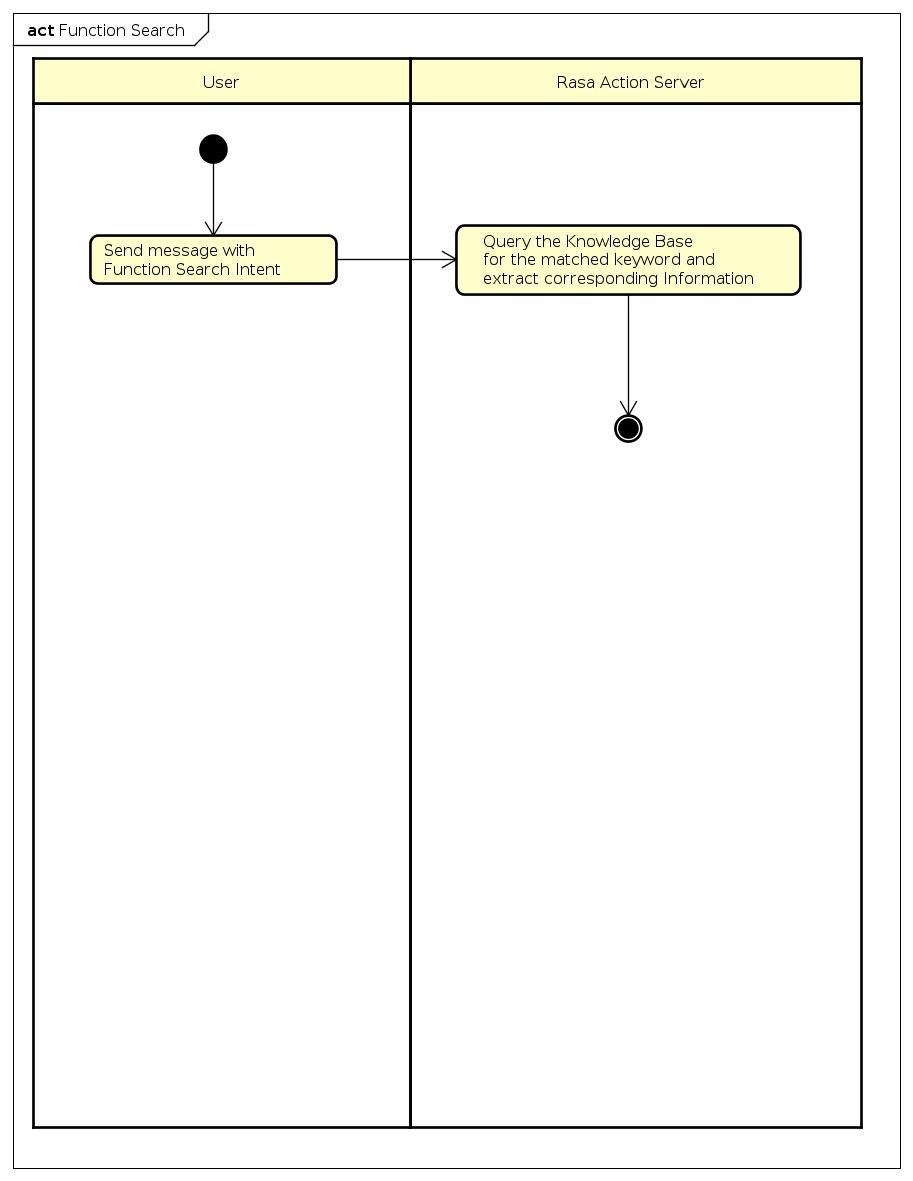


Figure 6: Function Search Activity Diagram

# Chapter 3: Technology

## 3.1. Artificial Intelligence

### 3.1.1. Natural Language Processing

Language is one of the greatest achievements of mankind. It is a major milestone in civilization that levitates the way human beings interact and communicate. It is its sophisticated and complex nature of language that invites challenges for science to replicate language understanding on machines. NLP, therefore, is one of the most challenging and important branches in Artificial Intelligence. Overall, this is a field in which it requires the machine to understand every aspect related to language. This abstract goal can be broken down into many specific tasks:

* Named Entity Recognition: Label every core component in a sentence such as a verb, a noun, an adjective, etc. This task can be further developed to label different classes to suit specific need
* Machine Translation: As straightforward as its name, this task requires the machine to automatically translate a given sentence or paragraph from one language to another. This is one of the most challenging tasks in NLP.
* Paragraph Summary: Summarize a given paragraph by one or a few sentences, the machine is required to capture the core idea of the paragraph.
* Voice Synthetization: Receiving input in form of voice, the machine’s task is to convert this into the corresponding texts/paragraphs.

### 3.1.2. Machine Learning

Tom Mitchell, one of the scientist which has many great contributions in the field of artificial intelligence in general, provides a modern definition for Machine Learning: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.” [1] Simply put, machine learning tries to find a pattern, represented by a formula, behind the data the machine is given to learn. With this pattern served as knowledge learned, the final expectation is for the machine to perform well with new input data it has not seen before. This simulates the learning process of human beings.

There are two major branches in Machine Learning:

* Supervised Learning: The data put in the model for learning includes the labels. Expressed in another way, the machine has already seen the goal - the outcome of what it will learn and need to generalize.

There are two sets of inputs: x = {x1, x2, x3...} and y = {y1, y2, y3...}

Goal: The general formula that can express the relationship between these input sets: y ≅ f(x)

* Unsupervised Learning: The results of the formula are not given. The machine needs to find a guidance for the learning process since it does not have a clear vision of the goal.

Input: x = {x1, x2, x3...}

Output: the formula y ≅ f(x)

In the scope of Machine Learning techniques, Rasa also support the ones for the Intent Classification task. I will train one existing model in Rasa dedicated for comparison with the performance of DIET Classifier. The sklearn intent classifier trains a linear SVM. Linear SVM are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. To classify data, we want to know if we can distinguish a new point in a domain of a certain set of classes. A point is presented as a p-dimensional vector, in this case, needs to be classified in a domain of (p-1)-dimensional hyperplane. There are many hyperplanes that can classify data points. The reasonable goal in this case will be to choose the hyperplane that can represent the maximum distance between two classes. So we choose the hyperplane that the distance from it to the nearest data point on each side is maximized. One common problem is the considering finite-dimensional space cannot linearly separate the existing set of data points. In this case, the proposed solution would be to expand this space into a higher-dimensional one. In this new space, the data can be separated linearly. So the mappings between the points of the original space with the new one needs to be defined, often by a kernel function k(x, y) selected. Specifically, in the case of Linear SVM, given a training dataset of n points of the form . Each y can take the value dedicated to its class, whereas represents a p-dimensional vector. The goal is to find the hyperplane that best classify the group of points x in term of the margin fashion mentioned above: its distance to the nearest point of each class would be maximized compared to every other hyperplane. This model will then be optimized using Grid Search, which is basically exhaustive searching over a subset of hyperparameter space. It also provides rankings of the labels that did not “win”. The Sklearn Intent Classifier needs to be preceded by a dense featurizer in the pipeline. This dense featurizer creates the features used for the classification.

### 3.1.3. Deep Learning

#### 3.1.3.1. Introduction

Deep learning is a smaller branch in Machine Learning. It includes any method or mechanics to construct the features that the machine needs to learn automatically. In another way, this is another thing that the machine will update over the training process instead of them being pre-designed. There has been many successful models and architectures in the fields of Computer Vision and NLP. Many successes achieved in Artificial Intelligence happens in the realm of Deep Learning. The field also levitates its performance thanks to the advance in hardware, specifically the parallel calculations in graphics processing cards. Now we will go explore the theory behind one of the latest milestones in Deep Learning: Transformer.

#### 3.1.3.2. Transformer

Since Transformer alone is a complex and advance architecture, I provide the outline for reader to follow this section:

* Prerequisites
  + Recurrent Neural Network
  + Attention
* Motivation
* Model Structure
  + Self-Attention
  + Multi-head Attention
  + Architecture
* How it is used in the RASA Scope

In the original paper, Transformer is categorized as an approach in Natural Language Processing (NLP) that tackled transduction problems such as Language Modeling or Machine Translation. During research in this specific field, many solutions were proposed and will serve as a good foundation to understand the motivation of Transformer.

##### Prerequisites

###### Recurrent Neural Network (RNN)

RNN is classified as a sequence-to-sequence model which means the input of the model will be a sequence of items and the output of it will also be sequence. The models of this type suit well with the NLP problems in general because of their nature, which consider the context information or the relationship between instances of input. This has always been a key point in solving NLP tasks effectively.

By design, RNN takes two inputs at each time step: an input and a hidden state. Remember that time is just an expression of the sequential flow of inputs. Input instance is put in the model one by one, so it is natural to present it as time step. Hidden state will play the role of “remembering” the context information I mention earlier and will get update in every time step.

Before Transformer, sequence to sequence model used to tackle NLP problems such as Machine Translation uses RNN as its essential component to design overall architecture. In Machine Translation, input language texts need to be encoded into an abstract “language” then will be decoded later into the texts in expected language. These two tasks will be handled by the encoder and decoder, which in fact are RNN models. Now context information, represented as hidden state mentioned above will be passed from encoder to decoder. In other words, all inputs’ information is remembered, to some extent, and are ready to be used by the decoder. The reason I say “remembered to some extent” is because there is a limitation in this method. The context calculated by this way can only carry the information by a certain range and struggles in tasks that deal with long texts or paragraph. This motivates researchers to come up with a new method called Attention.

###### Attention

We will go answer two questions: what needs to pay attention and what will it pay attention to. In the traditional method, the encoder will pass its last hidden state to the decoder. But in this new approach, the encoder is required to pass all its hidden states in every time step to the decoder. The decoder, therefore, have a lot more information to process. This opens the opportunity to increase the performance, but the challenge now is how can the decoder use all the hidden states effectively. So, the answer to the first question is *the decoder* needs to “pay attention” now, and it must pay attention to all the hidden states received from the encoders. I will go into detail what the term “paying attention” mean in this context. So, researchers need to find a way to guide the decoder to attend to hidden states which have valuable information to it and ignore others. This is what Attention does. From the point of view of the decoder, it can use Attention module to assign a score to every hidden state. This score is a reflection to the amount of attention the decoder should pay to every hidden state. The output of Attention module will be a single piece of information (a vector) representing the attention distribution the decoder has on the set of all encoder’s hidden states. This attention information will be used along with the hidden state of the decoder itself to generate the output. That sums up the basic idea of Attention.

##### Motivation

In the prerequisites, I have just overviewed the basics of RNN, which was the state-of-the-art architecture in the domain of sequence-to-sequence models. This architecture, however, contains a bottleneck in design: for the time step i to be executed, it has no other option but waits for the time step i-1 to finish. This recurrent connection prevents the architecture to maximize the use of parallel computation in nowadays graphics cards, which are the hardware dedicated for Deep Learning calculation. On the other hand, convolution neural network (CNN) does not suffer from this type of bottle neck because it allows vectors to be computed in parallel. With these notations in mind, researchers design Transformer as an attempt to capture the advantages of both RNN and CNN. It must model dependencies between time step outputs in a way that does not prevent throughput, which means recurrent connections need to be removed. In the next section I will explain how Transformer is able to achieve this.

##### Self-Attention

Self-attention is a sequence-to-sequence operation and is the essential operation in any transformer architecture.

Here for any input vector i, it will index over the entire sequence and assign a weight for every other input vector.

Equation 1: Self-Attention Output Vector

The weight here, which represents the attention a certain input vector has for all other input vectors, hence the name self-attention, is not simply a parameter, but is derived from a function between and by a form of function. The simplest one can be:

Equation 2: Weight between two vectors

Lastly, the weight is applied SoftMax to map its value to the range from 0 to 1, and to ensure that the weight values of the whole sequence must sum up to 1. For the next input vector, this whole process is repeated.

The learning here is to decide what value each vector should take. The goal is: with those values, meaningful relationships between vectors can be represented. For example, the word “the” in most cases has a vague contribution to the meaning of the whole sentence. The value of this word vector should be able to represent that, which means the dot product between it and every other word should be low or negative. On the other hand, if two words are related, the learning process should be able to assign their vectors so that their dot product would have a high value.

In self-attention design, a notation is inputs are treated as set, not sequence. Considered the motivation mentioned above, this intention is understandable. Transformer does take input positions into account, but how this information is presented comes in later part.

##### Queries, keys and values

These terms derive from the database domain, where information is stored in form of key-value pair. When you want to access to a certain piece of information, you give out a query to search in the database. If the query and the key match, the value of that key can be retrieved. In self-attention, each input vector must play the role of all three: query, key, and value.

Input vector is used as **query** to compare with every other vector to establish the weights for its corresponding output .

Input vector is used as **key** to compare with every other vector to establish the weights for the output of the j-th vector .

Input vector is used as **value** to combine with the weight to establish the final output vector.

For each role, input vector needs to be transformed into a totally different new vector. This transformation is done via three weight matrices , , . Presented in equations:

Equation 3: key, value and query from input vector

The weight is achieved from key and value, after a SoftMax transformation:

Equation 4: Weight from key and value

The final output vector now can be calculated from the weight and the value vector:

Equation 5: Final Output Vector

The √k is taken directly from the original paper. This is a scaling factor to counter the sensitivity of the SoftMax function with large input values. High values can be the reason for the gradient to saturate. This leads to the weight upgrading process to slow down and even stop.

##### Multi-head Attention

In the paper, the use of self-attention is elevated by the proposal of multi-head attention. The idea is, from the point of view of each word, the meaning of every other words can be totally different. Using multiple attentions to describe the relationship between vectors adds more discrimination to the presentation and is almost vital in the case of natural language problems.

For each attention head, it is assigned with a different triple of matrices , , . The same process happens in each head. All the outputs are concatenated and put through a linear transformation to reduce the dimension back to the size of one output vector.

One problem in the architecture when we eliminate the sequential property in RNN is now we lose the information of position in each vector. To make up for that, researchers come up with a way to encode the positional information into every vector. The positional encoding is just another vector, designed to follow a repetitive pattern so that the architecture can recognize the distance between vectors, a good way to present the position.

The formula for the positional encoding:

and

Equation 6: Positional Encoding (Source: [2])

With this formula, it is possible to scale for the input vector even if we do not know exactly the size of it beforehand.

Now we have covered the theory of the most basic and most important component in Transformer, we will move to explore the architecture design of it.

##### Architecture Design

The self-attention is the heart of this architecture. It is almost the definition of any architecture to be qualified as a transformer one. Now we will discuss the design of the layer which wraps around the self-attention. One of the motivations of the design is the ability to stack many layers together. The original Transformer paper follows its predecessor and keeps the encoder-decoder architecture. These encoder-decoder consists of many layers stacked together, with the component of each layer discussed below.

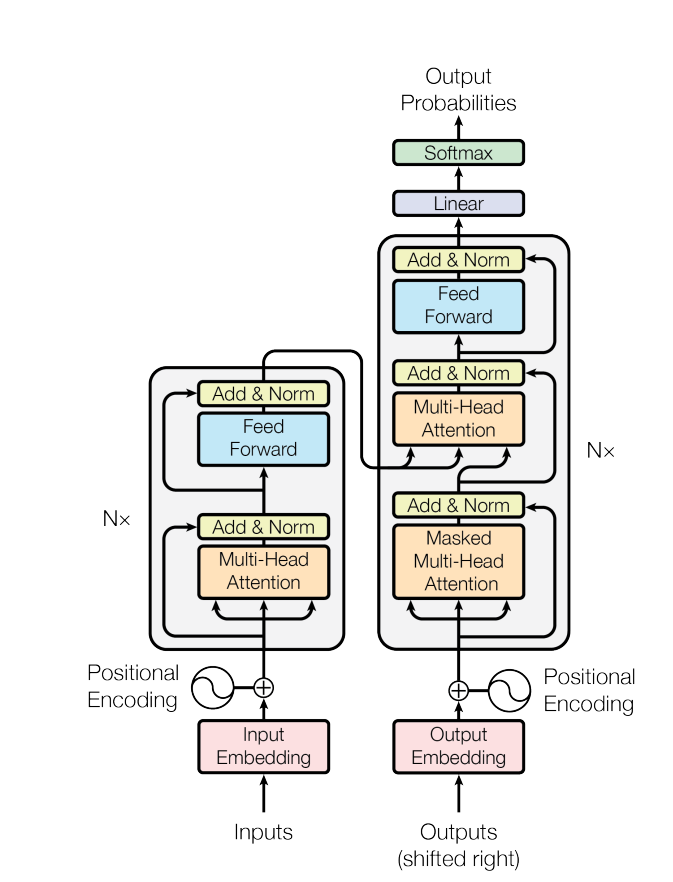


Figure 8: Transformer Architecture (Source: [2])

RNN is now completely replaced by the multi-head attention. The other sub-layer component is the feed forward neural network. The input flows through these two subcomponents inside both encoder and decoder in familiar style: Residual connections are used along with normalization. Normalization is used to pull the values of the outputs to the same scale. The idea of residual is simple: it skips connections between components inside the architecture. For example, input vector can casually skip the multi-head attention and goes straight to the next component. Originally, this method allows neural network to go deeper, have more layers because skipping connections means simplifying the networks, the gradient has to propagate through a shorter path than usual, which proves to be an effective way to counter exploding or saturating gradients.

The difference between the encoder and the decoder is the encoder-encoder attention sub layer in the decoder. As the name suggests, in this sub layer, the output vectors of the encoder which are in the form of the key-value pairs will serve as the inputs to encoder. This process serves the same purpose as in the attention of the RNN. The decoder must distribute its attention to all the information passed to it from the encoder. The query is generated from the self-attention sub layer of the decoder itself. The only difference of the decoder’s self-attention layer compared to the encoder’s is it must mask out the vectors that has not been predicted yet. For example, if the output of the task is to translate an English sentence to its German version and the decoder’s translation process is now at a word in the middle. The process has not finished yet so it cannot see the rest of sentences. So, the decoder can only pay attention to the words that have been translated, and assign a very large value, which will guarantee the value 0 after SoftMax, to mask out the words that have not been translated yet. In the original architecture in which the goal is to build a machine translation system, the output of the decoder goes through the linear layer. This transforms the vector into a much larger one which has the same size as the vocabulary of the language that needs to be translated to. This is called the logits vector. After going through the SoftMax layer, now it is ready to represent the probability distribution of the whole vocabulary in every time step. From it, the word which has the highest probability is the one to translate to.

##### In the Scope of Rasa Framework

In the scope of RASA Framework, the transformer is used in the DIET Classifier, the component which can predict both intents and understand entities from users’ sentences. RASA provides us with the access to configure all the important hyperparameters like the size of the transformer, the number of transformer layers, the number of attentions. This component’s greatest flexibility is it can still perform well without loading a pre-trained language model. As the language model’s availability for different languages other than English is still a limitation, the guarantee of the RASA components to perform well with just the data provided in native languages, without the use of the correspondent language model (Vietnamese in this case) is a big boosting point in performance for the developers’ community. Also, its advantage of Transformer architecture in accelerating training time, which has already been proven compared to RNN architecture, with the optimized implementation from RASA, leaves us with a very robust and at the same time very powerful component in term of natural language understanding tasks.

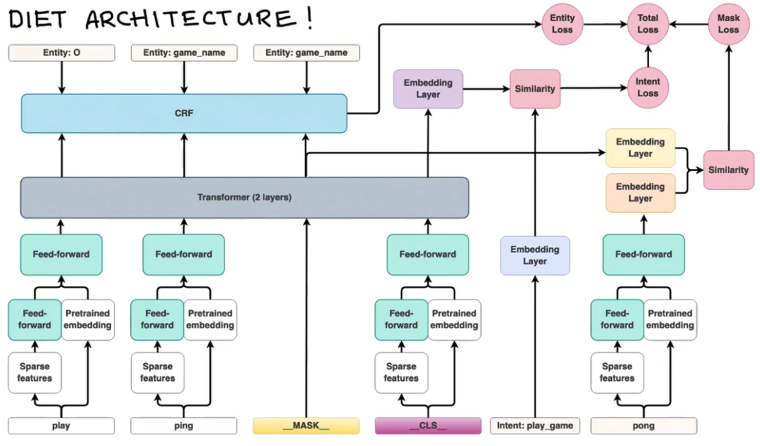


Figure 9: DIET Architecture (source: [3])

Now we go discuss some important points in DIET Architecture. Each token will go in by two directions. One direction is to go through a pre-trained embedding language model, if it is available. The other direction is to go through the featurizer component to produce sparse/dense features. In the scope of this thesis, the branch of pre-trained embedding has been cut off due to the lack of a Vietnamese language model. One input that needs consideration is the “\_\_CLS\_\_”. This plays the role of the summation of the whole utterance, by combining all the sparse features of the tokens and the summary of the pretrained embedding (the method of summary depends on the language model) together. This means the model not only learns every token specifically, but also the representation of the whole sentence in general. This knowledge serves as a powerful guide to predict intent, which is exactly what the model does. It combines the \_\_CLS\_\_’s output from the Transformer with the embedding of the intent to calculate the intent loss.

One last type of input that needs to be considered is the mask. The mask is just a random replacement of a token in the original input. The idea of the model when doing this is it wants to generalize the learning process, by predicting the token being masked. The reason behind this is: the texts in the domain of chatbot can vary greatly compared to the domain of formal texts in newspapers or Wikipedia source, which are usually the sources the language models get the data to train from. The users might tend to use a lot of abbreviations or slangs, so this method of masking is a way for the model to learn these expressions.

The entity loss is calculated by using the tokens’ outputs from transformers with the ground truth entities’ labels and putting it in a Conditional Random Field (CRF). Now we will go overview the theory of CRF.

The core idea of the CRF is this model will consider the values of neighbors to give out the final predictions. By doing this, besides learning the information from the inputs themselves, the model will also learn the relationships among inputs, represented by a transition matrix for each relationship. This extra learning information plays a vital role in the case of language where contexts need to be put into consideration.

So to sum up, if we train DIET Classifier to predict both intents and entities, then there will be in total three types of losses the model would use to optimize: the entities loss, received from the CRF, the intent loss, when we compare the intent label with the \_\_CLS\_\_ output, and the mask loss when you randomly mask out a token and predict it to generalize the model better.

Test performance of the DIET Classifier will be provided in the later section.

## 3.2. Chatbot – Rasa Framework

### 3.2.1. Architecture

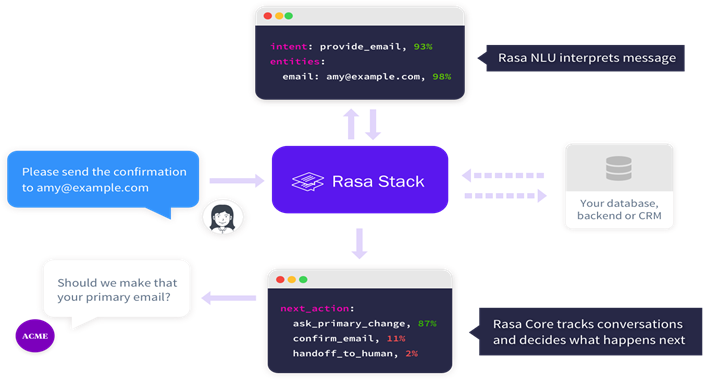


Figure 10: Rasa Stack Including NLU and Core (source: [3])

The above diagram shows the progress flow whenever Rasa Framework receives a message from users. Two main components of Rasa are Natural Language Processing (NLU) and Core. NLU will break down the user’s message to extract the needed information. This information is used by Core to decide which action should be taken place next.

### 3.2.2. NLU

Taken from Rasa Document, “Rasa NLU is an open-source natural language processing tool for intent classification, response retrieval and entity extraction in chatbots”. Three main pieces of information Rasa needs to extract from users’ messages are their intents, entities and the responses mapped to specific messages if configured. Now we will explain what intents and entities are.

Every user’s message contains an intent that he/she tries to convey. The task of identifying this intention is essential for any language-related task and is the key to a quality conversation since knowing the user’s intention in every message means the chatbot is capable of delivering the right action and this will lead to a fluent conversation. DIET Classifier is the component in Rasa that takes charge of detecting users’ intents, along with other components placed on top to support it. Insights of these components will be provided later.

Besides intents, there are certain pieces of information we need users to provide. To extract these is equivalent to do the task Named Entity Recognition (NER) in NLP. These pieces of information are regarded as entities and NLU needs to identify them effectively to fulfill the developer’s designed purposes. Rasa equips its users with many Entity Extractors. In my specific case, I use CRF Entity Extractor and Duckling Extractor. In fact, the CRF mechanism has already been embedded inside the DIET Classifier and the basic of CRF theory has been covered above.

Besides CRF, I also use Duckling to support the task entity extraction. Duckling is a library developed by Facebook that parses text into structured data. This library supported many languages including Vietnamese. However, the number of entities the library can extracted are limited. The mechanics behind it is rule-based, so the confidence it gives to every time an entity is extracted is 1. This library is embedded well with Rasa Framework, so it is just one configuration away from being used. In my specific case, I use Duckling to extract entities such as number, amount of money, duration, time. The entities can be overlapped. For example, an amount of money is also a number. Duckling will return both values.

Intents and entities’ examples need to be provided as input for the training process. Like any Machine Learning (ML) task, the more specific and quality examples are provided, the better NLU will perform.

Besides the above vital pieces of information, response retrieval is handled by NLU when a certain message is met, a specific response can be returned immediately. This is almost the way a traditional if-else system works.

In the configuration of the pipeline, we can see the main flow to process raw user’s input sentences into suitable form for the system to digest. This preprocessing is taken care of by two components: the tokenizer and the featurizer.

The tokenizer chops down the sentences into many tokens, ready to be processed. In this case, on the base of the Whitespace tokenizer given by the framework, we adapt and build a new tokenizer called No Accent Tokenizer to remove all the Vietnamese marks along with tokenizing.

Featurizing is the step of converting all the tokens into vectors. There are two types of conversions: spare and dense. Sparse vectors of every sentence take the size of the whole vocabulary and assign all the tokens in the vocabulary but are not contained in the sentence the value 0. As the size of a single sentence is usually much smaller than the size of the vocabulary itself, most of the values in that vector will be 0, hence the term “sparse”. RASA counters this waste of memory from storing very large vectors with lots of 0 value by only storing the vector by the values and the slots in which those values are presented. This contributes to the model another features called sparse feature. The output of this component is the matrix that has the size equals the number of tokens times the features dimension.

### 3.2.3. Core

According to Rasa’s definition, Rasa Core is a dialogue engine which uses a machine learning model trained on example conversations to decide the bot’s next action. With this engine, Rasa aims to liberate developers from hand-crafted rules that will stack up along the way when the chat bot is getting bigger. Interactive Learning is the method Rasa Team recommends when doing initial design of a bot. The basic idea is user will give feedback to the chatbot step-by-step, point out the action the chatbot should have chosen if it picks the wrong one. This whole process will later be exported as the training data format. This approach seems like a much more natural way to provide the chatbot with meaningful conversations and help it give the correct decisions than designing handcrafted rules which are easily conflicted and harder to debug. In the case of this thesis, due to the lack of testers to conduct interactive learning conversations along with the specific requirements of banking use case, the design of conversations’ flow needs to be taken care of directly by developer and many chatbots actions are customized, which will be broken down into detail in the later part.

Stories is the term defined by Rasa to identify a certain flow in a specific conversation. This flow simulates the basic exchanges between users and the Chatbot. For every intent provides by the user, the matching action should be executed by the Chatbot. This can be used as a standard guideline on how the Chatbot should behave.

One of the most important features the Core Module delivers is the ability to manage the conversation flow. To be more specific, it defines certain policies for the Chatbot to follow. Each policy is built based on certain rules or methods which all aims to decide the correct action for the Chatbot to execute. Almost all policies take the Stories predefined above as a source for reference. However, the way each policy uses the Stories Resources is different. Basic configurations that support these policies includes:

* the number of sentences that are put into account to serve as the context to decide the next action.
* the augmentation factor, which is an optional configuration that will merges different scenarios provided to enhance these Stories. Augmentation is a basic method in the Machine Learning realm to enhance or enrich the existed dataset. So only the policy with Machine Learning based will take advantage of this configuration.

The Policy list is hierarchically divided in term of priority in execution. The list is provided below. The higher number implies higher priority in deciding the Chatbot’s action:

5. Form Policy

4. Fallback Policy

3. Memorization Policy

2. Mapping Policy

1. TED Policy

Form Policy is at the top of the hierarchy. Before explaining this design decision by Rasa, I will go explain the Form Feature in Core Module.

Form Actions will be suitable in the case the user is required to provide certain set of information before a specific action can be executed. In Chatbot Dialogue State Management, there are two styles of developments when it comes to Handcrafted approach - the approach where you manage states by a set of rules predefined instead of letting the Chatbot adapts freely to every situation:

* Finite State: there is a state corresponding to every step the chatbot and the user reaches in a conversation. In each state, there are fixed number of transitions the conversation can be turned to. Overall, this approach is rigid and offers almost no adaption to the user’s input.
* Frame Based: This method provides more flexibility compared to Finite State. The possibilities of the transitions can be described as tree structure. Every slot pre-defined can be filled from anywhere of the conversation.

Form Actions fall into the Frame Based style of development. One of the biggest flexibilities of this method is user can provide more than one piece of information at once and the method will make sure the Chabot acknowledges all these pieces of information without asking the user again. Compared to the Finite State where the Chatbot will give out the output sequentially without the adaption to user’s input, this brings robustness and smoothens user’s experience.

So, the Form Feature supports exactly this. It helps developers define beforehand the set of slots - pieces of information it will collect from the user. It evens support further by defining how these slots can be filled. For example, a slot can only be filled when the entity related to it is met in a specific user’s intent. This solves a lot of overlapped situations when entities can come in many shapes and forms in user’s input and we only want to collect it at the right moment. In this project’s domain, Top Up Phone or Transfer is the scenario in which applying Form is appropriate since the Chatbot must collect a predefined set of information from User in order to help the customer execute the extraction. This also means once a Form Action is activated, the Chatbot’s only aim will be to collect all the information in the set defined. Now the design decision from Rasa to put Form Policy at top priority is clear. the Form Action cannot finish until the user provides all the information that the chatbot needs. This strict policy delivers a certain amount of guarantee that the Chatbot will collect all the needed information, but it also brings a potential weak point in user experience. The user might be asked repeatedly by chatbot to provide the information even when he/she no longer wants to execute the current task or cannot fulfil the requirement for different reasons. In order for user not to fall into this vicious loop, a configuration can be done so that user can escape the Form action at will. In term of Conversation Management, we will go into detail how Rasa handles the conversation flow, with the core idea coming from Frame Based approach above.

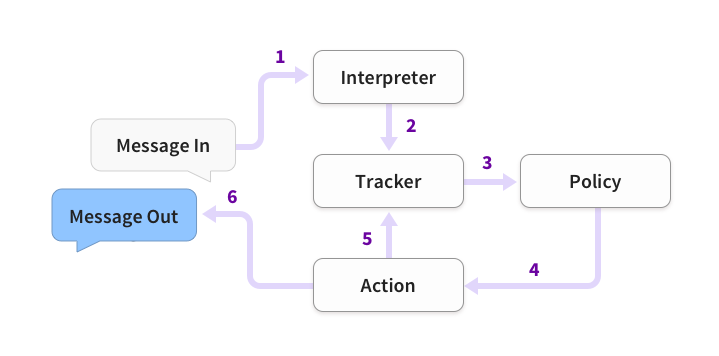


Figure 11: Progress Flow Diagram (source: [3])

The Diagram demonstrates how Rasa manages the conversation every time a new user’s message arrives.

* Interpreter is the first module receiving and processing User’s messages. The NLU is what powers this module and it will return the intents along with all the entities it successfully extracts from the messages.
* Tracker is basically how the Chabot memorizes the conversation. For every new state, it will be logged to Tracker. As the name implies, the information saved can be tracked by future steps to support decision making.
* The Policy we discuss is the component that decides which action the Chatbot should take next. The information the policy processes comes from the Tracker.
* The Action chosen by the Policy will also be logged into the Tracker. The Action will give out the final message to the user.

The Fallback Policy determines the condition in which the Chatbot will not take any action. This scenario occurs when at least one of the following conditions are met:

* The intent threshold is not met. This is equivalent to the Chabot not able to detect the user’s intention with a reliable enough confidence to execute any action.
* Ambiguity happens. This is when the confidence scores the Chatbot provides for two or more intents are too close to each other. In this case, it is also not possible to decide which is the correct intent from user.
* None of the dialogue policy decides the next action with a reliable enough confidence. Simply put, the Chatbot cannot decide the next action to take by the component it uses for this task.

Memorization Policy is activated when the scenario has been defined in the Stories provided. Simply put, the Chabot is in a situation it has already met and know exactly what to do next. The action given by this policy, therefore always comes with the confidence 1.

Mapping Policy is when for a certain intent from user, specific action from the chatbot is expected, and will be activated as soon as this intent is detected. As strong as this configuration might be, it still comes after in priority compared to the Form Policy.

TED Policy is powered by the Transformer. The architecture behind this component is pre-defined. The training steps of this policy are as follow:

* The user input including the intent and the entities, the previous system actions, slots and active forms are all concatenated for each time step and represented in form of input vector to be put into pre-transformer embedding layer.
* This embedding vector will be fed into the transformer.
* The output of the transformer then goes through a dense layer to create embeddings for the dialogue.
* All the actions will go through a dense layer to create embeddings for the system actions
* The similarity between the dialogue embedding and the embedded system actions is calculated. The action that has the embedding with the highest similarity score to the dialogue embedding is selected.

This process is executed repeatedly, every time the Chatbot receives new a new message and has to decide what action to do next.

Rasa also supports developers to configure some of the most important hyper-parameters in the TED Policy the same way as in the DIET Classifier such as:

* the number of epochs: each epoch represents one time the entire dataset goes through the model in the training process. By saying going through it means one forward pass and one backward pass through the entire model.
* hidden layers’ sizes: the number of feed forward layers and the output dimension/size of the output vectors to represent dialogues and intent. These layers will be placed before the transformer in the training process.
* number of transformer layers: correspond to the number of transformer blocks used in the model
* transformer size: the output vector’s size of the transformer.
* weight sparsity: this modifies the complexity of the feed forward layers in the model. The weight is in range 0 to 1. 0 means the layers’ function as a standard feed forward type. The closer this parameter is to 1, the less intense the learning process will be for the feed forward layers.

This wraps up the basic theory behind the policies in Rasa. This is the core component to manage the actions of the chatbot. In the scope of the project, Form Policy plays a vital role since the requirement of the Banking functions suits the nature of this policy. It makes sure the user will provide all the information we need in order to proceed with his/her next action. TED Policy holds the potential to bring better robustness and improvisation to the system. Unfortunately, the specific use cases in this project has not been thoroughly tested to examine the performance of this policy.

### 3.2.4. Features Used

Retrieval Action is an experimental feature in the new version of Rasa that is applied in the project. It is another approach from the traditional method to deal with cases such as FAQ or Small Talk from users. All the FAQ comes in all shapes and forms will be combined into a single intent but still distinguished into that intent’s types. The examples of corresponding responses for each type is treated as input of a separate dedicated model in order to decide what response the chatbot should return when a new sentence comes. By saying separate, the Response Selector Training process really is conducted separately from the NLU Intent and Entities Extraction Training. The responses are also defined in a different file instead of being put together in all other component in one domain configuration file. One notation is that when the responses are changed, the Response Selector Component needs to be changed again.

Knowledge Base Actions is another experimental feature that is used in the project for the Function Search Use Case. A dedicated intent is defined for this feature that will activate the function once the chatbot meets it. A knowledge is pre-defined and serves as a small database for the chatbot to refer to and get the correct information related to the topic. This is not machine learning based and works based on the matching keywords, so the flexibility and adaptation is low to new examples.

Both Retrieval Actions and Knowledge Base Actions is applied to the system and solved a quite serious problem. Before these two features are provided, a lot of Intents have to suffer from too little examples. This leads to extremely unbalanced distribution in the Database. One intent can have a lot of examples while another intent can only be given a few examples due to it only serves a narrow purpose. This affects the way the model learns from the dataset. Even though the Transformer has a reputation for distributing its attention effectively, which in theory can deal with the unbalanced behavior from the training dataset, applying the Transformer alone cannot seem to distinguish the intent with few examples with confidence in a small experiment. The Knowledge Base Actions still bears some limitations but overall, this is a good trade off compared to affecting the performance of the entire system.

## 3.3. Interface

Now when Rasa serves as the core to guide the Chabot performance, there are many options when it comes to giving the Chatbot an interface. One of the most direct ways a Chatbot can be deployed in the Banking domain is through the Mobile Application. In fact, this is an inevitable development direction if the end goal is to better support Banking customers. However, another potential direction will be to deploy the Chabot to a popular Messaging Platform. This is an easier way to approach customers since it does not require the users to install or setup any extra application, besides the platform they have already used. On the other hand, the biggest challenge of this approach is almost all functions related to Banking are not ready to be served outside the scope of its own Banking Application. This idea holds great potential as it will levitate the role of the Chatbot. If the Chatbot can assist users executing Banking functions right in the Messaging Platform, this will be a breakthrough in usability and convenience. However, in terms of security, this brings great challenges of users’ and transactions’ management and validation. The merger of Banking platform with Messaging Platform still awaits further investigation in term of possibility and suitability to the users’ behaviors. In the domain of this project, I will deploy the Chatbot to Facebook Messenger, which is one of the most popular social platforms in Vietnam.

The Messenger Platform of Facebook provides every function a Chatbot can ask for. A handful of features, API, web plugins, web view support, along with the ability to transfer more data types than just texts like images, videos, audio files, etc. makes this platform becomes an almost perfect environment to deploy any Chatbot.

### 3.3.1. Page-scoped ID (PSID)

When a User initiates a conversation on Facebook whether by clicking on the Get Started button, or sending a message, he/she will be given an ID by Messenger Platform in the scope of the Page he/she sends the message to. This identifies the user so the chatbot can pick up a conversation with a user from where it stopped last time. Also, from the users’ perspectives, PSID means Pages have to be given the permission from users to start sending messages to them.

### 3.3.2. Webhook

Prerequisite Terms:

* Application Program Interface (API) is a software intermediary that allows two applications to talk to each other.
* Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers. HTTP works as a request-response protocol between a client and server.
* GET Method: GET is used to request data from a specified resource.
* POST Method: POST is used to send data to a server to create/update a resource.

Now when we interact with users through the Facebook Messenger Platform, the first thing we want to control is when users will send a message to the Chatbot. In the scope of Web Development, this is considered as an Event. As of using traditional APIs, if we want to know if a new Event happens, we must call to the server regularly to check. This is a passive method and cannot be applied to real-time feature. In order to solve this problem, Webhook is another type of API that an application will use if it wants to notify other application about the Event in real time. This saves the application from a passive position and can provide response to that Event in time. Webhook events are sent by the Messenger Platform as POST requests to Rasa webhook.

With that basic idea of Webhook, now we will go into detail how to setup one. Every webhook requires a callback URL. Facebook requires verification that we really control the domain that hosts our application. In order to prove this,

1. We provide Facebook with the webhook URL along with a developer generated verify token via the app dashboard.
2. The Messenger Platform will try to verify our webhook by sending a GET request to the callback URL with the parameters listed below:

|  |  |
| --- | --- |
| Parameter | Description |
| hub.mode | Set to subscribe |
| hub.verify\_token | The custom verify token that the developer provided |
| hub.challenge | Generated by the Messenger Platform. Contains the expected Response |

Table 9: Parameters in Messenger Webhook (source: [4])

1. The callback URL responds with the value of the hub.challenge sent. The URL should validate that the hub.verify\_token matches with the token that was entered in the app dashboard.

In the scope of our application, the main Webhook Events we should focus on is the *messages* Event and *messaging\_postbacks* Event, which when we subscribe, we will get notification whenever we receive a message or a Button Click Event from users. Rasa provides us with the basic configuration to setup our Webhook to match the requirements from Facebook.

### 3.3.3. WebView

WebView is another strong feature that Facebook Messenger supports. It provides the ability to view a Website right in the scope of the Messaging Window instead of being transferred to the Web Browser and forced to leave the Messaging Application. This helps the Chatbot provides more complex features than the ones limited by the nature of messages. In our case, when the Banking Applications are applied, this potentially will be the platform to develop those applications.

### 3.3.4. Web Plug-ins

In another approach, instead of accessing the WebView from Messenger, in reverse we can plug the Messenger Button inside the scope of our Web so users visiting the Website can easily start a conversation with Customer Service or Chatbot right away with the Messenger Platform. The Users’ Facebook Information can also be controlled and retrieved if needed.

# Chapter 4: Development

## 4.1. Architecture

The proposed Architecture is 3-tier. This is a client-server software architecture pattern in which the User Interface, the Webhook, the Rasa Server are three main components and are developed independently with each other. This independence provides freedom to develop and update module without effecting other components and the whole product.

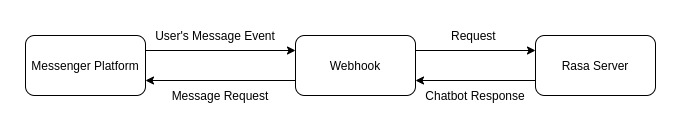


Figure 12: Overall Architecture

Users interact with the Chabot through the Messenger platform. Whenever a message is sent, webhook will be notified by a POST request from the Messenger. Webhook will then transfer the users’ message to the Rasa Server and retrieve the Chatbot’s Response. This response is sent to the Messenger Platform by the Webhook using the Messenger’s API.

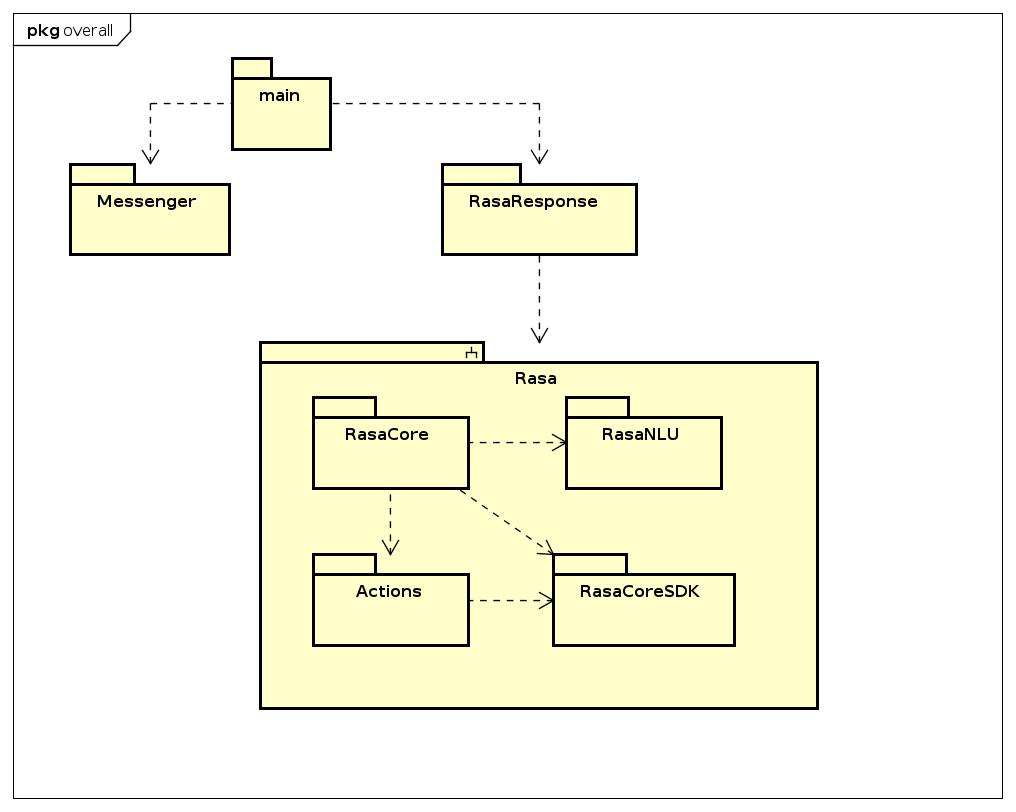


Figure 13: Overall Packages

In the system, from the main package, two packages Messenger and Rasa Response are imported. Messenger Package plays the role of sending the messages to the Platform using the API provided by Facebook. Rasa Response Package’s sole purpose is to retrieve the corresponding answers from the Rasa Chatbot using the API provided by the Framework.

Inside Rasa, there are many sub-packages connected to each other. When Rasa receives a message, Rasa Core will transfer this message to Rasa NLU in order to get the intents and entitles extracted from users’ messages. From this information, Rasa Core can decide which actions the Chatbot should execute next and call the Actions Package. Actions are executed through an endpoint URL provided by Rasa Core SDK. This package supports developing custom actions besides the ones provided by the Framework. Furthermore, it also deals with defining corresponding events and the Tracker related to storing information for later retrieval.

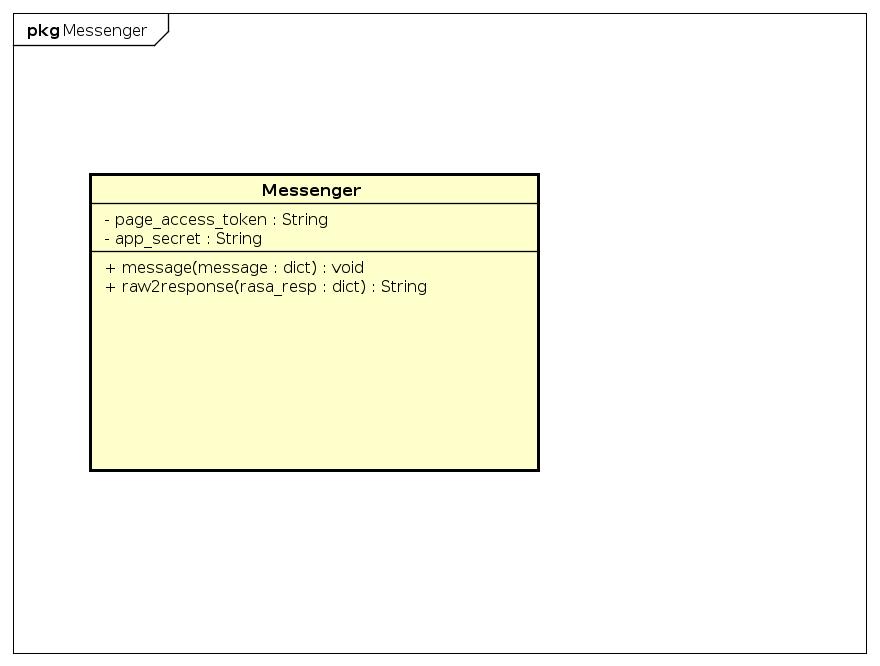


Figure 14: Messenger Class

Messenger Class:

* Attributes:
  + Page\_access\_token: The token used to verify the Facebook Page.
  + App\_secret: the verifcation code used when setting up the Messenger Webhook.
* Operations:
  + Message (message): the operation to send the message to Facebook Messenger, with the attribute is the user’s message.
  + Raw2response (rasa\_resp): convert the output given by Rasa to valid response that can be consumed by Facebook Messenger.

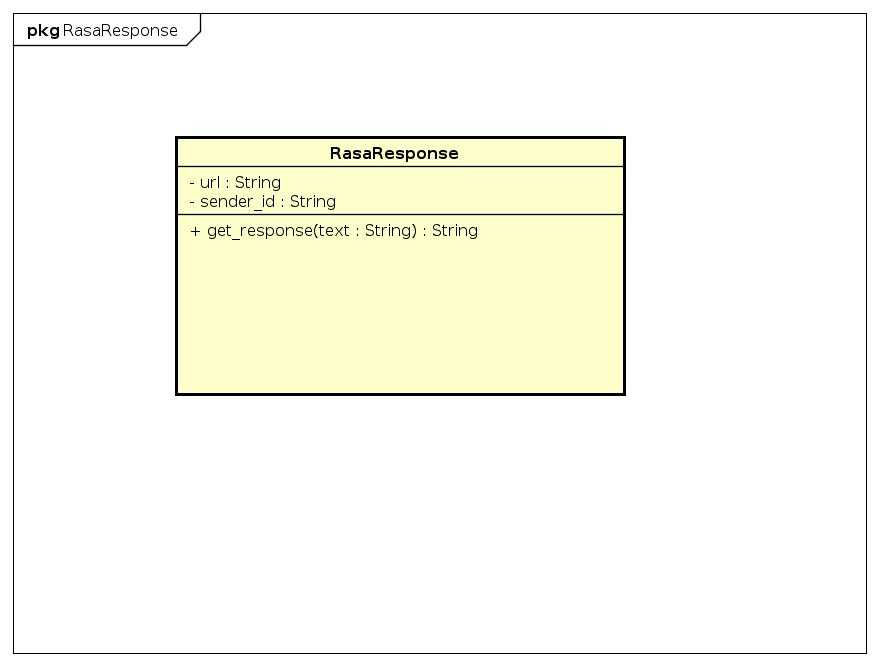


Figure 15: Rasa Response Class

Rasa Response Class:

* Attributes:
  + Url: the URL of the Chatbot Server
  + Sender\_id: PSID of the Facebook Messenger User
* Operations:
  + Get\_response (text): retrieve the answer from the Chatbot corresponding to the message in “text” from the user.

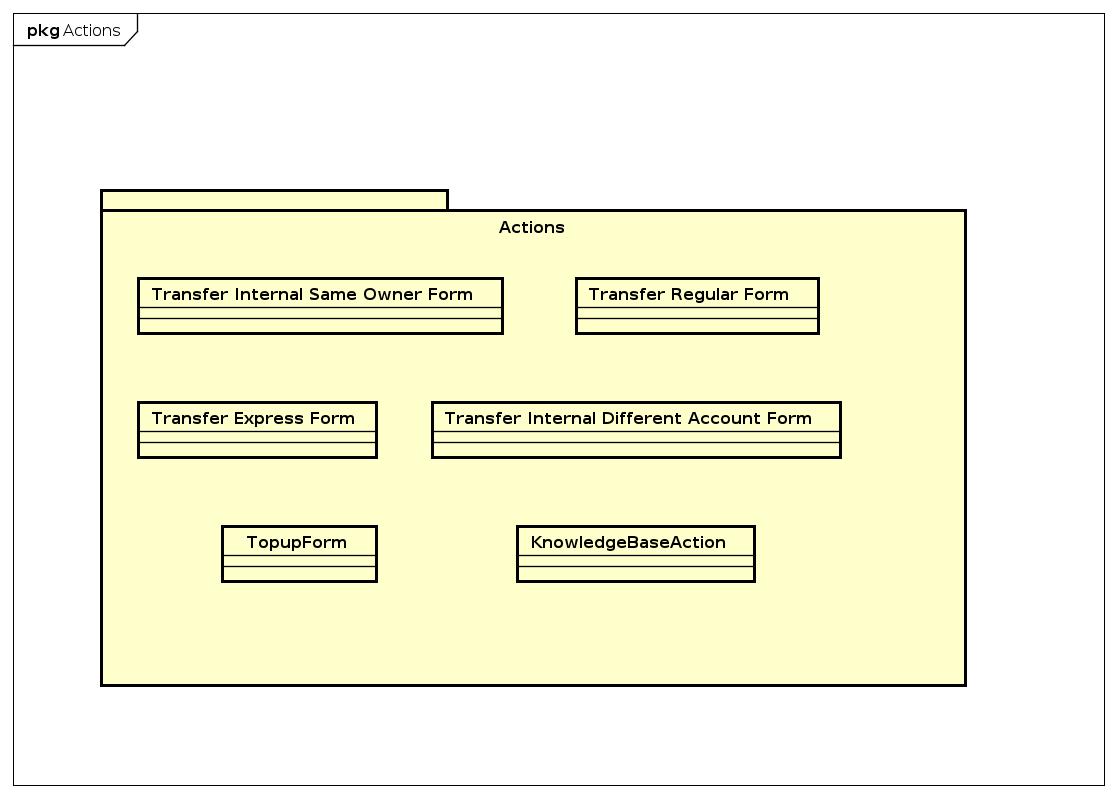


Figure 16: Actions Classes

Actions Package consists of 6 classes. Every class corresponds to a set of actions dedicated to handle a function. In each function, there is a predesigned flow of process to collect information from user and handle exceptions when user provides incorrect information, cancels, etc. In the case of FAQ, the process to select corresponding response for user’s question is handled directly by the DIET Classifier Component.

## 4.2. Tools Used

|  |  |  |
| --- | --- | --- |
| Purpose | Tool | URL |
| IDE | Visual Code | <https://code.visualstudio.com/> |
| Programming Language | Python 3.7 | <https://www.python.org/> |
| NLU | Rasa NLU | <https://github.com/RasaHQ/rasa> |
| Dialogue Management | Rasa Core | <https://github.com/RasaHQ/rasa> |
| Web Server | Flask | <http://flask.pocoo.org/> |
| Interface Platform | Messenger | <https://developers.facebook.com/docs/messenger-platform> |
| Library to connect with Facebook Messenger's API | Fbmessenger | <https://pypi.org/project/fbmessenger/> |
| Support using MongoDB Database | Robo3t | <https://robomongo.org/> |

Table 10: Tools Used

## 4.3. Testing + Results

### 4.3.1. Data Overview

|  |  |  |
| --- | --- | --- |
| Main Intent | Number of Sub Intents | Number of examples |
| FAQ | 7 | 300 |
| Top Up Phone | 4 | 282 |
| Transfer | 4 | 192 |
| Transfer Internal Different Owner | 1 | 80 |
| Transfer Internal Same Owner | 1 | 50 |
| Transfer Internal | 1 | 84 |
| Transfer External Express | 1 | 55 |
| Transfer External Regular | 1 | 65 |
| Function Search | 1 | 60 |
| General (Greet, Thanks, etc.) | 7 | 1000 |
| FAQ Response | 0 | 1 corresponding answer for every question |
| Total |  | 2168 |

Table 11: Data Overview

|  |  |
| --- | --- |
| Story Type | Number of Stories |
| Transfer External Express | 10 |
| Transfer External Regular | 10 |
| Transfer Internal Different Owner | 10 |
| Transfer Internal Same Owner | 11 |
| Transfer in General | 13 |
| Top Up Phone | 12 |

Table 12: Stories Designed for Core

In total 2168 examples, based on the scenarios designed, there are expected outputs that contain information the Chatbot wants to receive from users. The simulated data is generated based on this expectation. [Chatito](https://rodrigopivi.github.io/Chatito/) is a dedicated tool to generate training data for Rasa. We can configure combinations of components to create sentences based on the expected entities, intents. The percentages of certain sentences’ structure can also be configured.

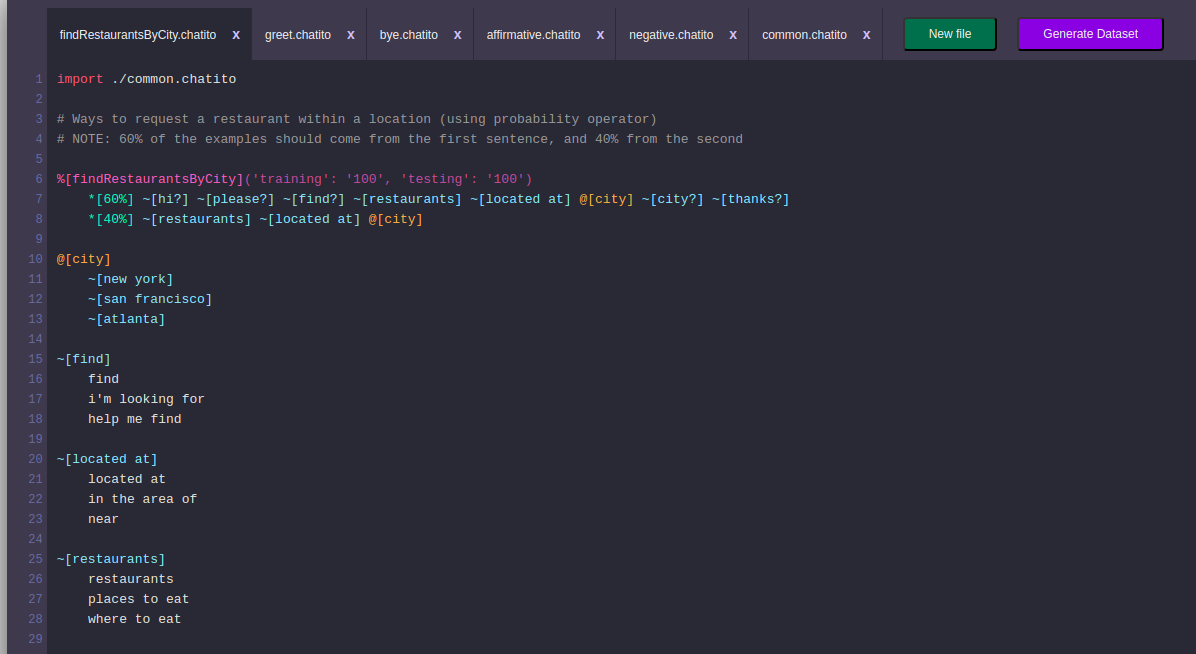


Figure 17: Chatito Interface

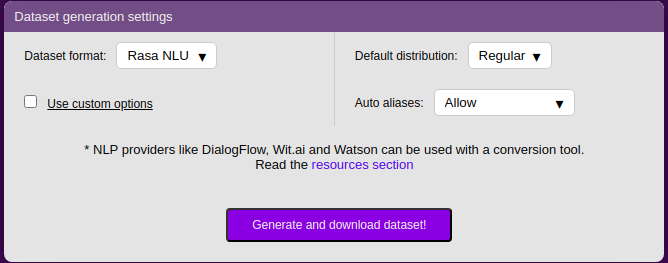


Figure 18: Chatito Settings when saving generated data

### 4.3.2. Results

#### 4.3.2.1. Intent Detection

Note: I only include Intents with over 20 test examples.

DIET Classifier Performance

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Intent | Precision | Recall | F1-Score | Intents confused with | Number of examples |
| thanks | 0.94 | 0.96 | 0.96 | greet: 8 times  ok: 3 times | 332 |
| Top up phone | 0.95 | 0.89 | 0.92 | Reselect phone number: 2 times | 47 |
| Query knowledge base | 0.92 | 0.65 | 0.76 | Greet: 12 times  Enter data: 3 times | 58 |
| FAQ | 0.92 | 0.95 | 0.94 | Greet: 5 times  Query knowledge base: 4 times | 269 |
| Enter data | 0.96 | 0.97 | 0.97 | Greet: 4 times  Cancel: 1 time | 260 |
| Transfer External Regular | 0.94 | 0.87 | 0.90 | Transfer External: 4 times  Transfer External Express: 2 times | 63 |
| Transfer Internal | 0.94 | 0.98 | 0.96 | Transfer Internal Same Owner | 82 |
| Transfer Internal Different Owner | 0.98 | 0.93 | 0.96 | Transfer External Express: 2 times  Transfer Internal: 3 times | 79 |
| Greet | 0.93 | 0.92 | 0.93 | FAQ: 5 times  Enter data: 5 times | 455 |
| Cancel | 0.9 | 0.6 | 0.72 | Greet: 4 times  Transfer External: 1 time | 20 |
| Transfer | 0.92 | 0.86 | 0.89 | Transfer External Express: 2 times  Transfer External Regular: 1 time | 30 |
| Transfer External | 0.8 | 0.97 | 0.88 | Transfer External Regular: 1 time | 39 |
| Transfer External Express | 0.86 | 1. | 0.92 |  | 53 |
| Transfer Internal Same Owner | 0.95 | 0.9 | 0.92 | Transfer Internal: 2 times  Transfer External: 2 times | 50 |

Table 13: DIET Intent Detection Result

Sklearn Classifier Performance

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Intent | Precision | Recall | F1-Score | Intents confused with | Number of examples |
| thanks | 0.92 | 0.94 | 0.93 | greet: 16 times  faq: 3 times | 332 |
| Top up phone | 0.9 | 0.77 | 0.83 | faq: 5 times  Enter data: 4 times | 47 |
| Query knowledge base | 0.79 | 0.9 | 0.67 | faq: 11 times  greet: 5 times | 58 |
| FAQ | 0.86 | 0.92 | 0.88 | Greet: 15 times  Thanks: 3 times | 269 |
| Enter data | 0.95 | 0.95 | 0.95 | Transfer External Express: 4 times  Greet: 1 time | 260 |
| Transfer External Regular | 0.88 | 0.86 | 0.88 | Transfer External: 4 times  Transfer External Express: 3 times | 63 |
| Transfer Internal | 0.92 | 0.94 | 0.93 | Transfer Internal Different Owner: 4 times  Transfer External Express: 1 time | 82 |
| Transfer Internal Different Owner | 0.8 | 0.92 | 0.86 | Transfer External Express: 1 time  Transfer Internal: 3 times | 79 |
| Greet | 0.89 | 0.91 | 0.9 | FAQ: 5 times  Thanks: 9 times | 455 |
| Cancel | 1. | 0.75 | 0.86 | Enter data: 1 time  Thanks: 2 times | 20 |
| Transfer | 0.93 | 0.9 | 0.91 | Transfer External Express: 1 time  Transfer External Express: 1 time | 30 |
| Transfer External | 0.86 | 0.92 | 0.88 | Transfer External Regular: 2 times  Transfer Internal: 1 time | 39 |
| Transfer External Express | 0.86 | 0.86 | 0.86 | Transfer Internal Different Owner: 3 times  Transfer External Regular: 2 times | 53 |
| Transfer Internal Same Owner | 0.9 | 0.94 | 0.92 | Transfer Internal Different Owner | 50 |

Table 14: Sklearn Intent Detection average performance

With the original architecture of Transformers, many language model has been proposed with different modifications to achieve state-of-the-art results in a wide variety of NLP tasks. In the scope of this thesis, I will apply the BERT pre-trained language model to the existing DIET Classifier to examine if it can improve the performance for the particular dataset in this domain. BERT’s key contribution is to apply the bidirectional training of Transformer. In theory, this technique will add a deeper sense of language and flow to the traditional Transformers architecture. Using the pre-trained model is the core idea of Transfer Learning, with the main purpose of adding a pre-determined knowledge base from that model to the specific domain that needs to be solved. The pre-trained language model is obtained from Hugging Face’s Transformers library - a well-known open source library that supports many Transformers-related language models. The component applies language model specific tokenization and featurization to compute sequence and sentence level representations for each example in our existing dataset.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Intent | Precision | Recall | F1-Score | Intents confused with | Number of examples |
| thanks | 0.94 | 0.93 | 0.93 | greet: 9 times  faq: 3 times | 332 |
| Top up phone | 0.95 | 0.78 | 0.86 | Query Knowledge Base: 3 times  Enter data: 2 times | 47 |
| Query knowledge base | 0.69 | 0.7 | 0.7 | faq: 3 times  greet: 7 times | 58 |
| FAQ | 0.93 | 0.9 | 0.91 | Greet: 17 times  Thanks: 3 times | 269 |
| Enter data | 0.97 | 0.97 | 0.97 | Ok: 2 times  Greet: 2 time | 260 |
| Transfer External Regular | 0.92 | 0.79 | 0.85 | Transfer External: 4 times  Transfer Internal Different Owner: 4 times | 63 |
| Transfer Internal | 0.79 | 0.9 | 0.84 | Transfer Internal Same Owner: 2 times  Transfer External Express: 3 times | 82 |
| Transfer Internal Different Owner | 0.79 | 0.75 | 0.77 | Transfer External: 4 times  Transfer Internal Same Owner: 9 times | 79 |
| Greet | 0.9 | 0.92 | 0.91 | FAQ: 11 times  Thanks: 12 times | 455 |
| Cancel | 0.67 | 0.6 | 0.63 | Greet: 4 times  Transfer: 2 times | 20 |
| Transfer | 0.78 | 0.7 | 0.76 | Transfer External Express: 3 time  Transfer External Express: 3 time | 30 |
| Transfer External | 0.78 | 0.85 | 0.81 | Transfer Internal Different Owner: 3 times  Transfer Internal: 3 times | 39 |
| Transfer External Express | 0.73 | 0.77 | 0.75 | Transfer Internal: 6 times  Transfer Internal Same Owner: 1 time | 53 |
| Transfer Internal Same Owner | 0.68 | 0.78 | 0.72 | Transfer Internal Different Owner: 4 times  Transfer Internal: 3 times | 50 |

Table 15: DIET Intent Detection with BERT Result

Performance Comparison Between Models

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Precision | Recall | F1-Score | Number of Test Examples |
| Macro Average | 0.81 | 0.8 | 0.8 | 1880 |
| Weighted Average | 0.93 | 0.93 | 0.93 | 1880 |

Table 16: DIET Intent Detection average performance

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Precision | Recall | F1-Score | Number of Test Examples |
| Macro Average | 0.85 | 0.69 | 0.73 | 1880 |
| Weighted Average | 0.89 | 0.89 | 0.89 | 1880 |

Table 17: Sklearn Intent Detection average performance

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Precision | Recall | F1-Score | Number of Test Examples |
| Macro Average | 0.69 | 0.66 | 0.67 | 1880 |
| Weighted Average | 0.88 | 0.88 | 0.88 | 1880 |

Table 18: DIET Intent Detection with BERT average performance

Observation:

The Pre-Trained Language model applied to DIET Classifier turns out to hurt the module performance since it cannot generalize well with the existing data in our domain. Compared with the Machine Learning method, DIET Classifier Component for Intent Predicting performance was better by a noticeable margin. The optimal option is to train the DIET component from scratch with our existing dataset. One thing to notice is the model has been trained on the existing dataset but has not yet to be tested on real world examples. One key thing to put into consideration is the nature of intents in this domain relate very closely to their correspondent key words. In another way, the model does not have to learn a too complex pattern in certain intents in order to achieve a good result. This result can change if the model encounters more complex intents.

#### 4.3.2.2. Entities Extraction

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Entity | Precision | Recall | F1-Score | Number of Examples |
| Function | 1. | 0.83 | 0.9 | 42 |
| Benefit Type | 0.97 | 0.99 | 0.98 | 1722 |
| Attribute | 0.95 | 0.41 | 0.58 | 223 |
| Me | 0.56 | 0.27 | 0.36 | 33 |

Table 19: Entities Extraction Result

Observation:

Entity with a fixed pattern such as phone number, account number or card number can be extracted effectively using Regex Entity Extractor

Simple entity such as benefit type only toggles between a fixed set of values and will receive a high score

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Precision | Recall | F1-Score | Number of Examples |
| Micro Average | 0.95 | 0.71 | 0.81 | 725 |
| Macro Average | 0.79 | 0.64 | 0.7 | 725 |
| Weighted Average | 0.93 | 0.71 | 0.785 | 725 |

Table 20: Entities Extraction Average Performance

#### 4.3.2.3. Response Selector for FAQ Use Case

Note: Only consider cases with over 15 examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Responses | Precision | Recall | F1-Score | Number of examples |
| Change Number Register | 0.9 | 0.93 | 0.92 | 49 |
| Busy line | 0.93 | 0.89 | 0.92 | 28 |
| Fee | 0.98 | 0.97 | 0.97 | 69 |
| Forget Password | 0.92 | 0.96 | 0.94 | 89 |
| Transfer Limit | 0.94 | 0.94 | 0.94 | 19 |

Table 21: Response Selector Result

#### 4.3.2.4. Dialogue Management

There are 68 pre-designed stories corresponding to 4 main Use Cases, each represents a conversation flow the Chatbot is expected to follow. After conducting the test, we have the result on how the Chatbot performed. A conversation is considered as Fail if one action is mis-predicted by the Chatbot.

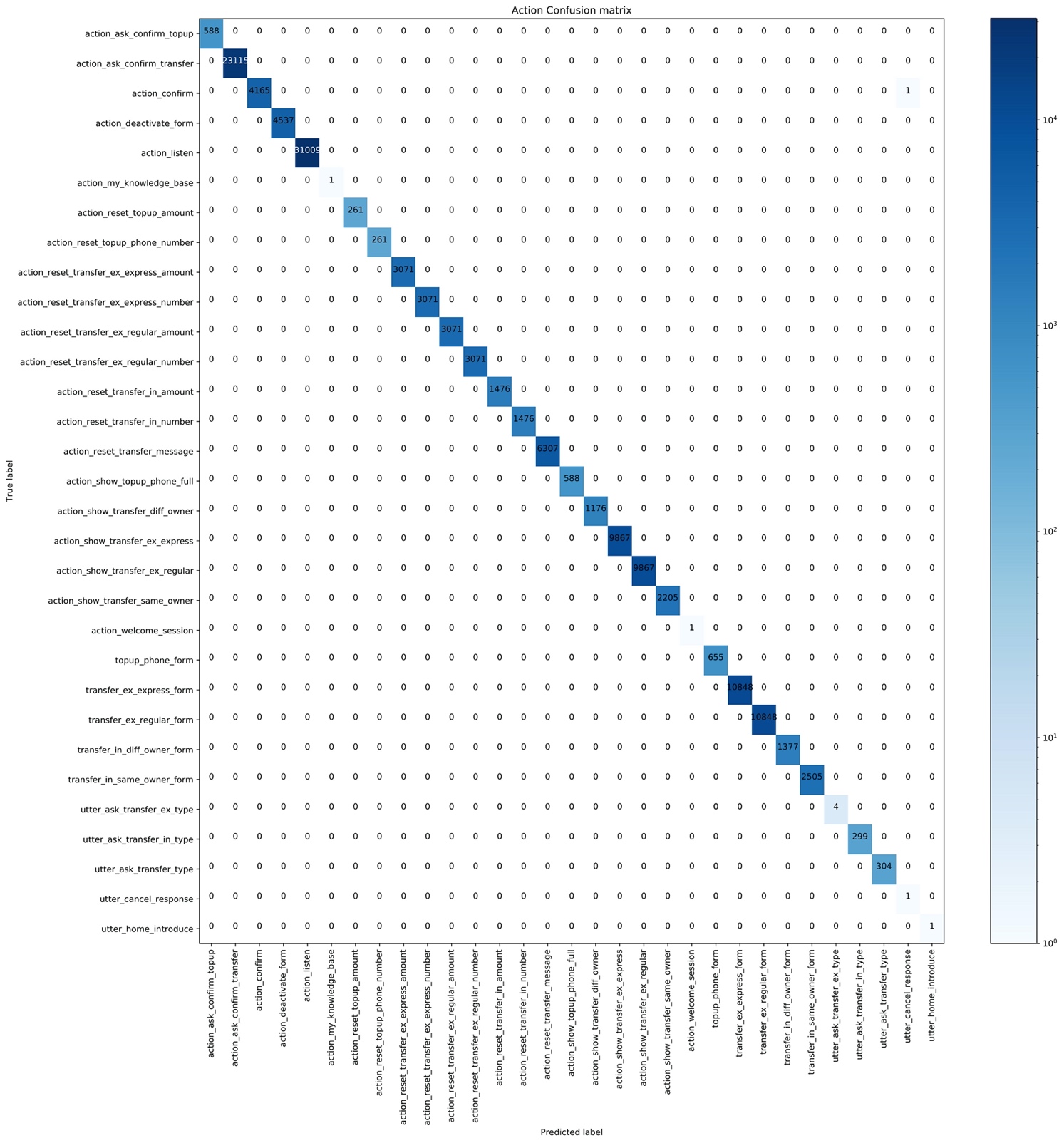


Figure 19: Confusion Matrix for Conversation Management

Above is the confusion matrix for the performance of the Chatbot in Dialogue Management. The Chatbot is considered to perform well if it executes all the expected actions at the same time. On the diagonal line, there are the number of correct actions the Chatbot executed in the scope of 68 tested stories. There is only one action that the Chatbot mis-predicted that leads to a failed story. This test proves that the Chatbot can follow the pre-defined scripts provided by the developer. Further testing on how adaptive the Chabot is to new and unseen situations need to be conducted in the future.

## 4.4. Deployment

### 4.4.1. Tracker Store

Retrieving users’ texts and chatbot’s actions are important. Users’ examples can later be used to improve the model through training. Saving Chabot’s actions can help the debugging process. All the information needed will be defined by Rasa as events, each event corresponds to either a Chatbot’s action or user’s message. Rasa supports a handy way to configure the framework with a database. Besides, Rasa is compatible with many popular database systems such as PostgreSQL, Oracle, Mongo dB, etc. In my specific case, I choose to connect the project with MongoDB. According to the main page’s definition, MongoDB is a document-oriented database, classified as NoSQL. It has many different terms compared to traditional database systems in the SQL branch. Some of them includes storing data in key-value pair, wide column, graph, document, etc. This helps escalating querying data in some cases and is overall better for scaling if the system needs to deploy over many servers. Below is the demonstration to connect Rasa to MongoDB locally. Prerequisite is MongoDB and Robo3t has been successfully installed in the system.

First, create a database corresponding to storing Rasa’s Tracker

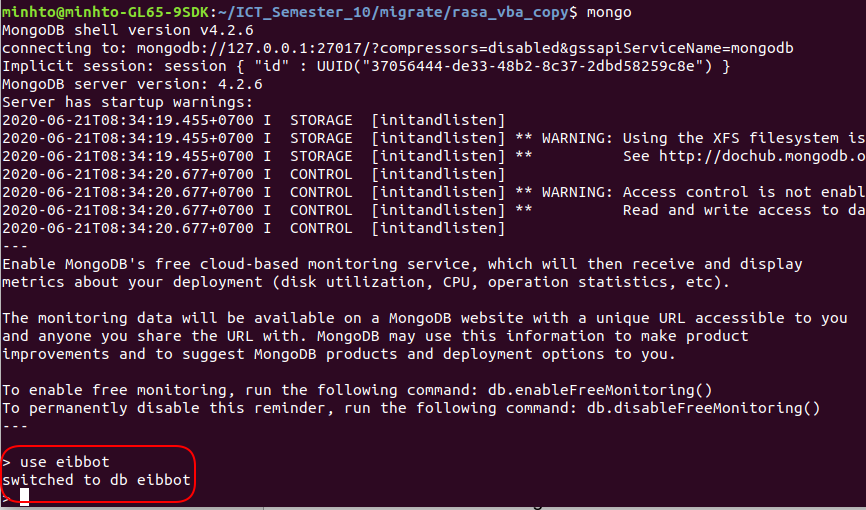


Figure 20: Create database in MongoDB

Next, create a user to access to the database just created

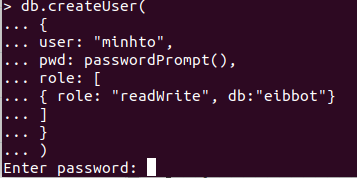


Figure 21: Create User in MongoDB

After these two steps, we provide the information just created to the configuration file in Rasa

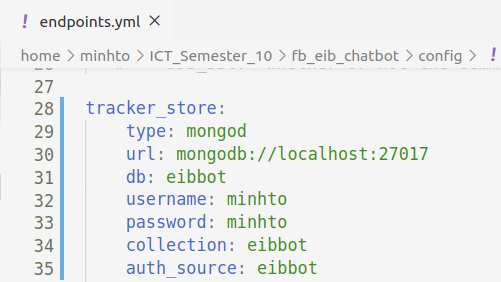


Figure 22: Configuration for Database in Rasa

With this, we finish connecting Rasa to a database system. When we run the rasa server and conversations have been conducted, Tracker for every user’s conversation can be stored and retrieved.



Figure 23: Robo3t Interface when Tracker successfully stored

### 4.4.2. Facebook Messenger Interface

To Deploy the Chatbot to Facebook Messenger Platform, first, we need to create a Facebook Page

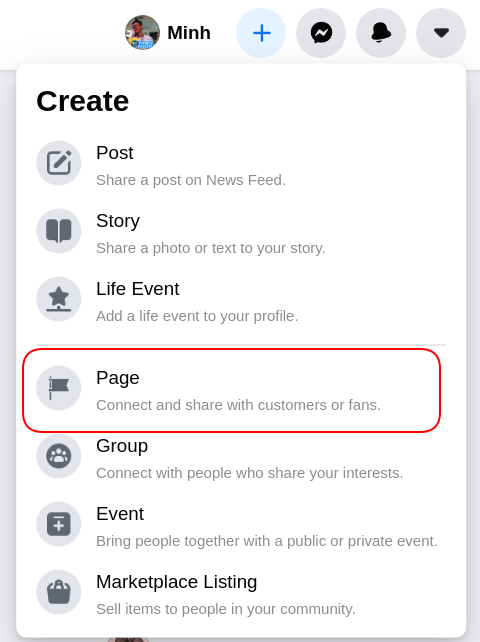


Figure 24: Step 1 in Creating Facebook Page (source: [4])

Fill in Basic Information Required to Initiate a Facebook Page

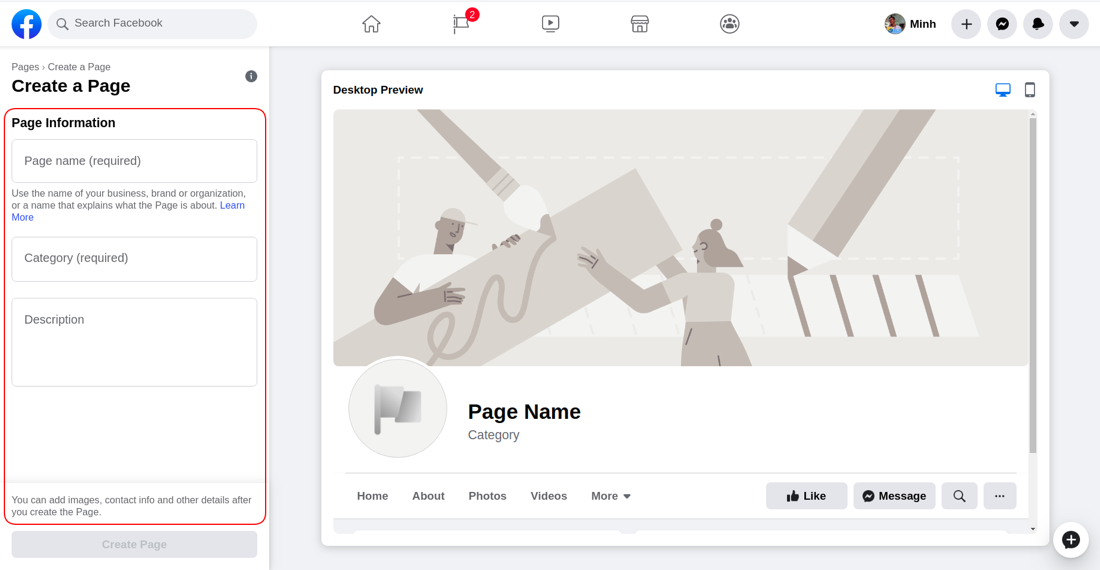


Figure 25: Step 2 in Creating Facebook Page (source: [4])

After we have our Facebook Page initialized, the next step is to create the Application on [Facebook Page for Developers](https://developers.facebook.com/apps/)

Next, we will setup our Application to use the Webhook and link the Page we just created with the Messenger.

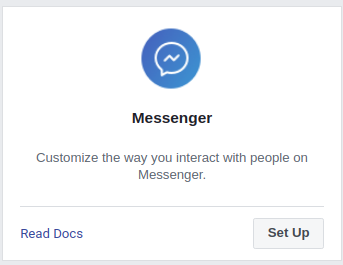
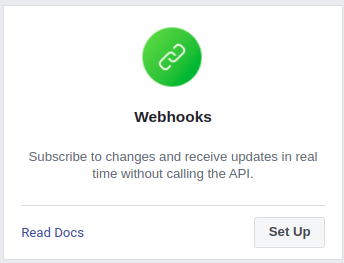


Figure 26: Configuration Components in Facebook Application (source: [4])

To link our Page with Messenger, find the Setting Section:

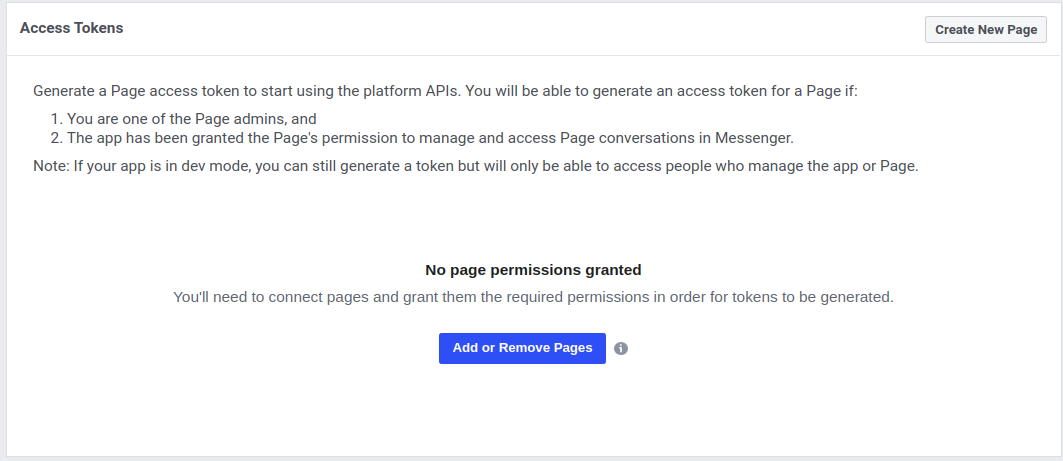


Figure 27: Link the Facebook Page to Messenger (source: [4])

After the Page has been linked up, we need to generate the Page Token. This Token will then be used as a field in the credentials file in Rasa.



Figure 28: Generate Token after Successful Linking (source: [4])

One more thing we need to achieve for our credentials file in Rasa is App Secret. We can get this piece of information from the Setting Section:

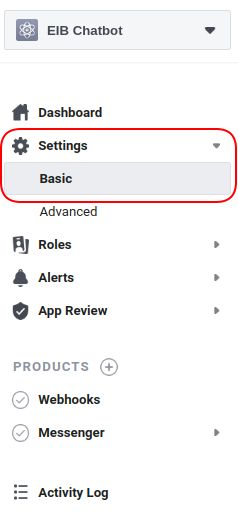


Figure 29: Settings for App Secret (source: [4])

Then click Show to retrieve the App Secret

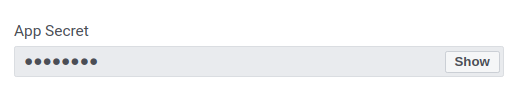


Figure 30: Retrieving App Secret (source: [4])

The next thing we will go configure is the Webhook. For this, we need to provide Facebook with a URL and a Verify Token to claim ownership you have over that URL. This Verify Token is the last field after the Page Token and the App Secret that Rasa requires in the credentials file to link up the Chatbot with Facebook.

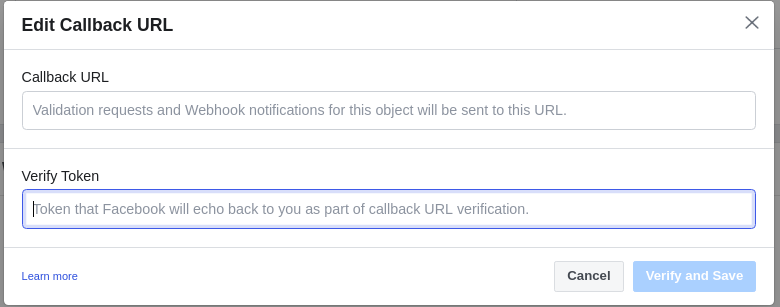


Figure 31: Configure Webhook (source: [4])

For Demonstration Purpose Only, I will use [Ngrok](https://ngrok.com/) to earn a public URL for my local server.

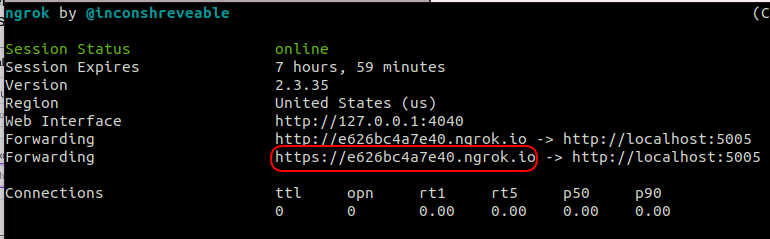


Figure 32: Public Localhost using Ngrok

# Chapter 5: Contribution

The first important thing when I approach a problem is to understand it fully. In the scope of Banking, in order to develop a Chatbot that can satisfy the specific requirements, understanding the domain knowledge is extremely important. This includes identifying with each Use Case, which fields of information the Chatbot needs to collect. This step is the foundation for designing the flow the Chatbot would follow later.

Data, which is one of the most important aspects defining the performance of a Machine Learning project comes from two sources. The first one is from logged data of real customers’ messages collected from another application. The second source is the data simulated based on the design which represents the expected information the Chatbot would like to receive.

The next important thing is to design the scripts for the Chatbot to follow. For most use cases, the Chatbot requires the users to provide it with some information. So, the easier the Chatbot can make the users understand what it wants, the smoother the conversation will be. In order to do this, for each sentence the Chatbot returns, it must be able to express its intention as clear as possible for users to know which step to follow. Moreover, Facebook Messenger provides the Quick Replies in its interface, which is a helpful and convenient way for Chatbot to communicate with users. Quick Replies, when being used effectively, can serve as a guide for users to provide fields of information the Chatbot would like to receive. This feature is applied in many cases in this project and greatly improved the fluency of conversations.

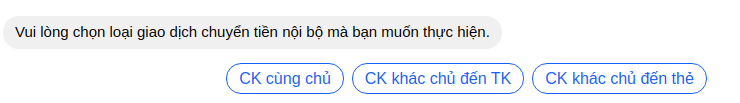


Figure 33: Messenger Quick Replies Buttons

Rasa Framework is a broad and powerful framework with handful of features. New features and components are updated frequently so this is both a bliss and a challenge for developers to keep up with the new technology. In fact, DIET Classifier with Transformers at its core is only supported in the recent version but has greatly improved the performance of Intent Prediction. Understanding the theory behind this powerful architecture means knowing the meaning behind every configuration to the hyperparameters in the architecture. This also means the potential to adapt future Vietnamese Language Model when possible to power the Chatbot in Natural Language Understanding Aspect.

Form Actions, Retrieval Actions and Knowledge Base Query are the features that are used to adapt to specific Use Cases in the project. Besides Retrieval Actions and Knowledge Base Query which are new and still under experimental trial, Form Actions has been a great feature and well developed by Rasa and suits well with the requirements of Banking use case. Understanding the source codes of this components is the base to add customization for the adaption of specific requests.

First, Form Actions let developers define the slots it will collect. Each slot represents a piece of information or entity. Condition can be considered so different set of slots will be initiated. There are two functions dedicated to the Entity Extracting Process: extract\_other\_slots() and extract\_requested\_slot(). As mentioned above, instead of collecting the information sequentially, Frame Based method mentioned above means the Chatbot can collect the information the moment it appears in the user’s sentence. This is exactly what the extract\_other\_slots() handles. After the entities are collected, there will be an extra validate() function to check if the value in the entities are suitable to be used. All these functions at some point need some customization by the developer to suit the specific use case. To check for validation of information provided by users, many string patterns are defined using Regular Expression (Regex), which works effectively to filter misinformation from users.

The reason Rasa Framework is one of the most complete solutions for Chatbot is the Framework’s concern does not just stop at how to make the Chatbot generates the correct output but also extends to the learning process of the Chatbot. By defining its own term dedicated to the dialogue's state and provides hand-on configuration to connect to various database systems, the conversations’ history is effectively stored and later can be extracted as training data for the NLU model through the complete API set. Besides storing data, Rasa also supports developers connect with many Message Platforms with ease.

Different components in NLU for predicting Intents and Extracting Entities are thoroughly tested to choose the most suitable ones for the available dataset. Many current powerful language models are supported by Rasa but cannot perform well on the project due to the incompatibility in language.

# Chapter 6: Conclusion and Future Development

To conclude, with Rasa Framework, we can provide a complete and production ready for Chatbot that can adapt to any specific domain. With the Banking function, the Chatbot can handle the conversation if customers provide correct information. The exceptions are handled to some extent but only when the Chatbot is really exposed to more cases from customers in real situations can the Chatbot be really improved.

The ability to improve the performance is also a strong point of Rasa. This is not only due to the nature of Machine Learning based method but also the active updates from the framework with the latest models and NLU methods. When a Vietnamese language model is compatible and can be embedded to the system, this promises to better the Chatbot’s performance. Also, the project’s scope delivers the solution for the Chatbot aspect only. In order to have a complete Banking application with full functionalities on a Messaging Platform requires huge effort to expose the existing Banking service. But the potential of this idea is worth considering since it can elevate the application’s convenience and users’ experience. Equipping the Chabot with a powerful model and designing the stories in an intuitive way so the users can follow the conversation with the Chatbot easily are equally important for the contribution of the Chatbot performance overall. More stories/scenarios can also be added with ease in the future if needed.

# Bibliography

|  |  |
| --- | --- |
| [1] | T. Mitchell, "Introduction," in *Machine Learning*, 1997. |
| [2] | Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, "Attention Is All You Need," 2017. |
| [3] | "Rasa Docs," [Online]. Available: https://rasa.com/docs/. |
| [4] | "Facebook Messenger Developer," [Online]. Available: https://developers.facebook.com/docs/messenger-platform/. |