TURKISH NATURAL LANGUAGE UNDERSTANDING SYSTEM

Senior Design Project II

Berfin Elif Erdoğan

Ezgi Nihal Subaşı

2021

MEF UNIVERSITY
FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

TURKISH NATURAL LANGUAGE UNDERSTANDING SYSTEM

Senior Design Project II

Berfin Elif Erdoğan

Ezgi Nihal Subaşı

Advisor: Dr. Şeniz Demir

2021

MEF UNIVERSITY
FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

Project Title	: Turkish Natural Language Understanding System
Student(s) Name	: Berfin Elif Erdoğan - Ezgi Subaşı
Date	: 05/06/2021

I hereby state that the design project prepared by Berfin Elif Erdoğan - Ezgi Subaşı has been completed under my supervision. I accept this work as a “Senior Design Project”.

05/06/2021
Dr. Şeniz Demir

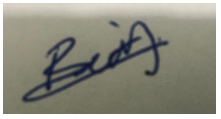
I hereby state that I have examined this senior design project by Berfin Elif Erdoğan - Ezgi Subaşı. This work is acceptable as a “Senior Design Project”.

05/06/2021
Prof. Muhittin Gökmen

Head of the Department of
Computer Engineering

ACADEMIC HONESTY PLEDGE

In keeping with MEF University Student Code of Conduct, I pledge that this work is my own and that I have not received inappropriate assistance in its preparation. I further declare that all resources are explicitly cited.

NAME	DATE	SIGNATURE
Berfin Elif Erdoğan	05/06/2021	
Ezgi Subaşı	05/06/2021	

ABSTRACT

TURKISH NATURAL LANGUAGE UNDERSTANDING SYSTEM

Berfin Elif Erdoğan
Ezgi Subaşı

MEF UNIVERSITY
Faculty of Engineering
Department of Computer Engineering

Advisor: Dr. Şeniz Demir

JUNE, 2021

As the relationship of humanity with computers increased, communication with computers gained importance. Chatbots, luminescent examples of human-computer interaction, are useful tools for engaging in a dialog with computers. With the help of this project, a Turkish chatbot will be designed to facilitate people's abilities and to overcome challenges that they face in their daily lives. The chatbot will be able to understand input of the user by using the natural language understanding component and will respond to them or take actions accordingly. Due to the busy daily life, people may forget to do some of their work or want to speed things up by spending less effort. In such cases, the chatbot will assist those people by receiving their commands and performing some actions on behalf of them. In this century where technology is flowing rapidly, the help of chatbots cannot be ignored.

Keywords: natural language understanding, chatbot, daily life assistance, turkish, human-computer interaction

ÖZET

TURKISH NATURAL LANGUAGE UNDERSTANDING SYSTEM

Berfin Elif Erdoğan
Ezgi Subaşı

MEF ÜNİVERSİTESİ
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

Tez Danışmanı: Dr. Şeniz Demir

HAZIRAN, 2021

İnsanlığın bilgisayarla ilişkisi arttıkça bilgisayarla iletişim önem kazandı. İnsan-bilgisayar etkileşiminin ışıldayan örnekleri olan sohbet robotları, bilgisayarlarla diyalog kurmak için yararlı araçlardır. Bu projenin yardımıyla, insanların yeteneklerini kolaylaştırmak ve günlük yaşamlarında karşılaştıkları zorlukların üstesinden gelmek için bir Türk chatbot tasarlanacak. Chatbot, doğal dil anlama bileşenini kullanarak kullanıcının girdilerini anlayabilecek ve onlara yanıt verecek veya buna göre eylemler gerçekleştirecektir. Yoğun günlük yaşam nedeniyle insanlar işlerinin bir kısmını yapmayı unutabilir veya daha az çaba harcayarak işleri hızlandırmak isteyebilir. Bu gibi durumlarda, chatbot bu kişilere komutlarını alarak ve onlar adına bazı eylemler gerçekleştirerek yardımcı olacaktır. Teknolojinin hızla aktığı bu yüzyılda chatbotların yardımı göz ardı edilemez.

Anahtar Kelimeler: doğal dil anlama, sohbet robotu, günlük yaşam yardımı, türkçe, insan-bilgisayar etkileşimi

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

NLU	Natural Language Understanding
SDP	Senior Design Project
ML	Machine Learning

1. INTRODUCTION

Our project is a travel agency chatbot based on Turkish Natural Language Understanding System. Natural Language Understanding (NLU) is a method that provides human-computer interaction by communicating with a human. NLU system understands the meaning of the text by using grammatical rules and common syntax to perform a conversation. Each sentence has an intent for the bot to understand the user sentence, and intents can have entities which are variables of the sentence. NLU systems determine which intent that input sentence has with intent classification, then find an appropriate answer for the response of the bot by using machine learning or deep learning methods. With the help of this project, users will be provided assistance when traveling such as ordering event or flight tickets, booking a hotel room and etc. Since conversation occupies a huge place in human communication, the use of chatbots is becoming widespread with the development of technology.

1.1. Motivation

The reason why we want to work on NLU is that we are aware of the importance of chatbots nowadays. Since there is very little work done in the Turkish language of the NLU side, we aim to develop a more clever conversation system. On the other hand, as Machine Learning is an important sub-branch of Computer Science that we want to deal with in the future, it motivates us that we will also go into this field while working on NLU.

Besides, we believe that this chatbot will provide great convenience to people who want to travel. Instead of dealing with tasks such as searching for events at the traveled city or buying tickets before traveling, it will save people time to handle them with the help of a chatbot.

1.2. Broad Impact

With the development of technology, people started to reach the things they want to do faster. However, the number of options that we have started to increase, and being able to do the desired things started to get complicated with this development. For this reason, and since people communicate by talking or texting in daily life, the use of chatbots has become widespread to speed things up. At this point, our project will help people who want to travel to quickly meet their needs and do what they want as a Turkish language chatbot. Besides,

since there are few Turkish language chatbots working with the NLU system, this project will contribute to this language option.

1.2.1. Global and Environmental Impact

With the development of smart assistants, daily life has become much easier. The ability to get things done without the need for manpower saves a lot of time. People are forced to use different websites to plan a trip. But with the automatization of this process, people will be able to make reservations much more easily.

1.2.2. Legal Issues Related To The Project

There are not any health security issues for the Turkish Natural Language Understanding System. In the future, if a company wants to use this chatbot security and protection issues may arise. In addition, the data of the user using the chatbot must also be protected.

2. PROJECT DEFINITION AND PLANNING

2.1. Project Definition

With the development of technology, the need for people to communicate with the machine has emerged. NLU, as the branch of NLP, is a method that aims to increase human-computer interaction by understanding natural human language. In the process of understanding, it is aimed to understand the intent and respond to people in order to provide communication even if there are mispronunciations. NLU can be used in dialog systems such as chatbots which we will be planning to do. After the NLU part, the system does NLG (which stands for natural language generation) to provide an answer as you can see in the Figure 1. NLU systems also deal with two key parts, which are intent and entity. Basically, the intent is what the sentence really wants to mean and the entity is the possible inputs that intents the same meaning. These features facilitate understanding in the training part. We believe that communication is important for humanity, so we are interested in making a chatbot system that will make our lives easier by communicating.

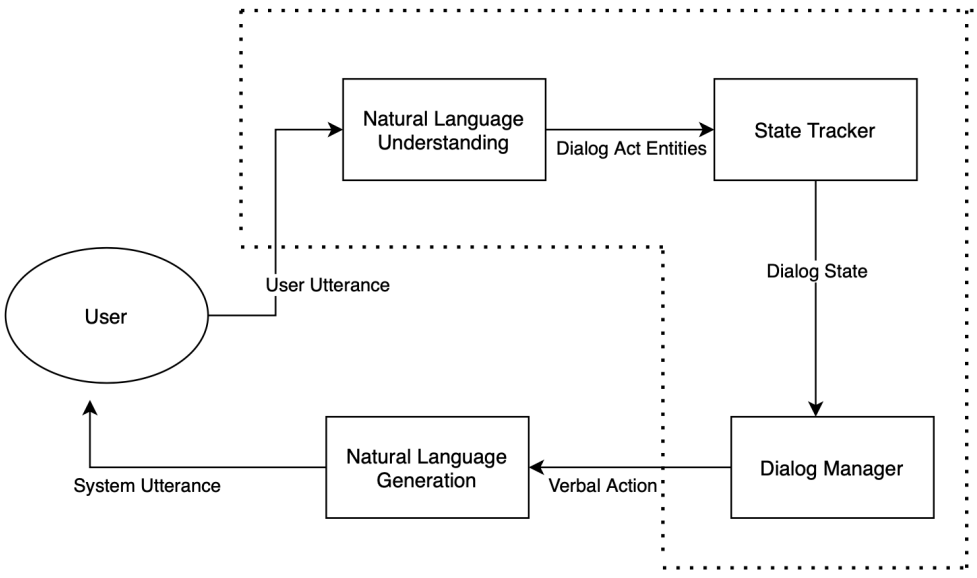


Figure 1. Diagram for understanding chatbot systems

Since the beginning of the project we have hard time finding a dataset. Besides not having many datasets in Turkish, there are not very different domains in English either. We came to the conclusion that we can combine different datasets to make a travel agency chatbot

to plan the users entire day. We have combined the intents in hotel, travel, weather and event chatbots. Since all the datasets are held in English, we translated them into Turkish with the help of Google API. And fixed the problems manually.

Dataset Domains	**Number of intents**		
travel	19	34	
Hotel	36	21	
event	17	10	6
weather	3	6	

Table 1. Dataset matrix

2.2. Project Planning

Planning is necessary for the project to be catch-up and working accurately on delivery time when making a long-term project. At the beginning of the project, we divide the tasks into a timeline as you can see in the table 2. Then we decided how much time we would devote to each task according to the difficulty of the task. Finally, we tried to follow this plan as best we could throughout the project.

Table 2. Sample project plan for 14 weeks.

Task	**Responsible Person**	**Weeks**													
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Preparing Proposal	Berfin Ezgi														
Literature Survey	Berfin Ezgi														
Preliminary Study in Rasa Framework	Berfin Ezgi														
Data Preparation	Berfin Ezgi														
Implementation of System	Berfin Ezgi														
Report and Project Presentation	Berfin Ezgi														

2.2.1 Aim of the Project

The aim of this project is to make a chatbot using the Turkish language based on Natural Language Understanding. With this chatbot, people who plan to go on a trip with this chatbot can buy tickets, book a hotel or go to the event, etc.

2.2.2 Project Coverage

This project covers Natural Language Understanding and Machine Learning topics, Python for the programming language, and to do the implementation of the chatbot with RASA's framework. Also, the Machine Learning model we use in this project is the DIET Classifier that RASA uses by default.

2.2.3 Use Cases

When the user runs the program, the chatbot does not write a message until the user enters a sentence as input. After the greeting with the chatbot, the user can make operations on 5 domains which are searching flights, hotels, events, and restaurants or asking weather. After the user has asked the operation to search, if the chatbot finds what the user wants, it starts the transactions for the user to book. The user can navigate between these operations regardless of order, optionally can ask about the weather and creating a plan for the trip. The chatbot responds at every stage in accordance with what the user wants to do and takes care of transactions such as reservations in the background.

2.2.4 Success Criteria

The success criteria of this project is depending on the error-free running of the program of chatbot and its ability to provide appropriate responses. If the program understands the intent of the user's sentences and can send a meaningful response depending on its properly trained model and test stories, the chatbot becomes successful.

2.2.5 Project Time and Resource Estimation

To complete this project, we had 14 weeks in total. Preparation of the proposal took 3 weeks, in this time we determined the content of the project and what we'll do. Literature survey took from week 4 to week 7. Since we couldn't find any Turkish dataset, the dataset preparation lasted from the 5th week to the 12th week as we translated and corrected the datasets we found in other languages. In between the 7th and 10th weeks, we studied the

RASA framework in order to implement our chatbot. From week 11 to week 14 we worked on implementing the system. Lastly, we prepared reports and presentations between the 7th to 9th weeks and the 12th to 14th weeks separately.

2.2.6 Solution Strategies and Applicable Methods

For implementing Natural Language Understanding based chatbot, there are various language models that can be used such as BERT, XLNet, etc. While doing a literature survey, we read articles for these language models, as well as researched other frameworks for making a chatbot. Finally, we chose RASA because it is both comfortable to use and flexible.

2.2.7 Risk Analysis

The biggest risk we encountered while doing this project was the wrong predictions of the model due to the fact that the intents and stories were not defined properly. Because the model made wrong predictions, the wrong answers were returning as a response to the chatbot, which disrupted the flow of conversation. The most important thing to do to prevent this is to create the model well and define the stories and dataset properly.

2.2.8 Tools Needed

In this project, we used Python programming language and the RASA framework as software tools. For hardware tools, we both used Mac Computers.

3. THEORETICAL BACKGROUND

We did some literature research to do this project. Having an idea about previous studies on our subject, being aware of the methods used by other people made the subject more understandable for us, expanded our perspective and vision.

3.1. Literature Survey

CASA-NLU: Context-Aware Self-Attentive Natural Language Understanding for Task-Oriented Chatbots

In classical Natural Language Understanding methods include intent classification (IC), slot labeling (SL), and dialog management (DM) to the dialog systems. These kinds of systems evaluate phrases as isolated then direct them to the DM for issues in context management. Since contextual information plays an important role in predicting and understanding accurately, this method is not sufficient enough. This model works on context management problems for better performance in understanding.

CASA-NLU model has three sections such that signal encoding, context fusion, and ICSL predictions as shown in Figure 5.

Signal Encoding:

In order to improve the encoder performance of utterance encoding, the directional self-attention-based encoder (DISAN) adds to absolute position embedding. DISAN occurs by word-level token2token (t2t) and sentence level source2token (s2t) that are multi-dimensional attentions. The output of the DISAN unit contains intent, dialog act, and slot label history that will be combined in context fusion.

Context Fusion:

This section concatenates vectorized contextual signals for the aim of receiving the current feature vector. If the context window is large than desired, the concatenating process becomes difficult. In order to solve this problem, source2token layer is added to each current context's component.

IC-SL Predictions:

In IC-SL predictions, this article uses a joint IC-SL model for training with improved IC performance by adding secondary IC loss function.

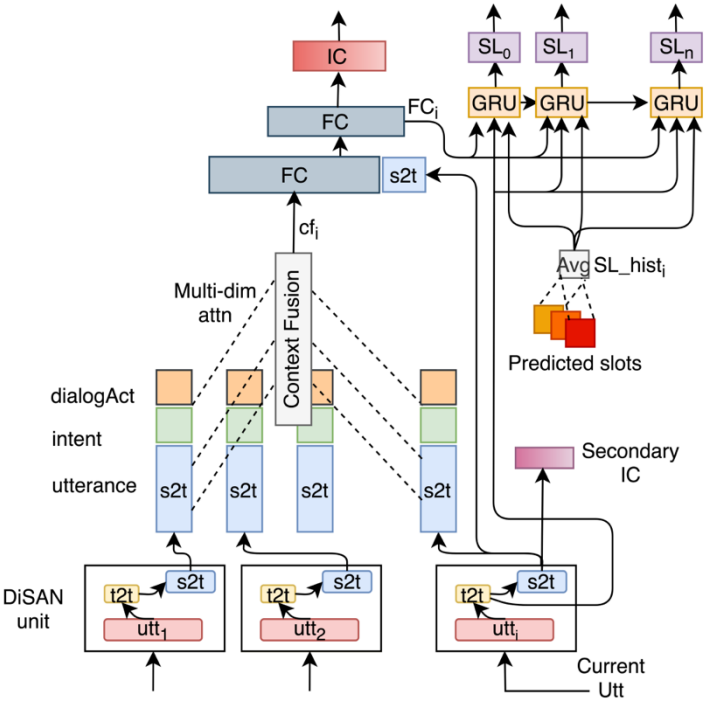


Figure 2. CASA-NLU model architecture for joint IC-SL

This project improves the NLU concept by focusing on the context management problem that occurred by intent classification and slot labeling. Using the contextual signals in addition to the current utterance as a method, the importance of contextual information role in predictions of intent and slot labels proved by showing the efficiency of 7% IC accuracy from Cable dataset over CGRU-NLU baseline [1].

Graph-Based Semi-Supervised Learning for Natural Language Understanding

Finding a sufficient number of labeled datasets is difficult. The most important benefit of using a semi-supervised learning technique in NLU tasks is helping to deal with insufficient labeled data. In this work, the connection between labeled and unlabeled data is demonstrated in a transductive graph for finding similar utterances with the paraphrase detection model.

In this article, since the paraphrase detection model used, nodes represent utterances, and edges represent paraphrase relations while labeled and unlabeled data are displayed on the

graph. Considering the created graph, each utterance's intent classification (IC) corresponds to node classification. It also uses an NLU Slot Gated Model (SGM) to obtain slot labels.

Transductive model in this work implements auxiliary models in the NLU model pseudo-labeling pipeline after learns paraphrase patterns among utterances. As seen in Figure 6, the first phase of pretraining of the model is provided through a structure.

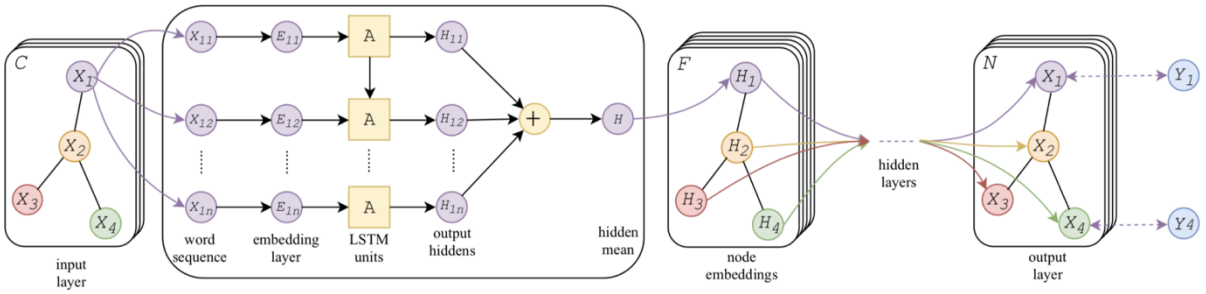


Figure 3. TGCN model architecture

Due to the limitations of this model in some cases during training, they used the transductive model in the inductive semi-supervised learning pipeline as in Figure 7 to generalize the unseen data. After completing the training process using a graph, the modeling is completed by applying paraphrase detection.

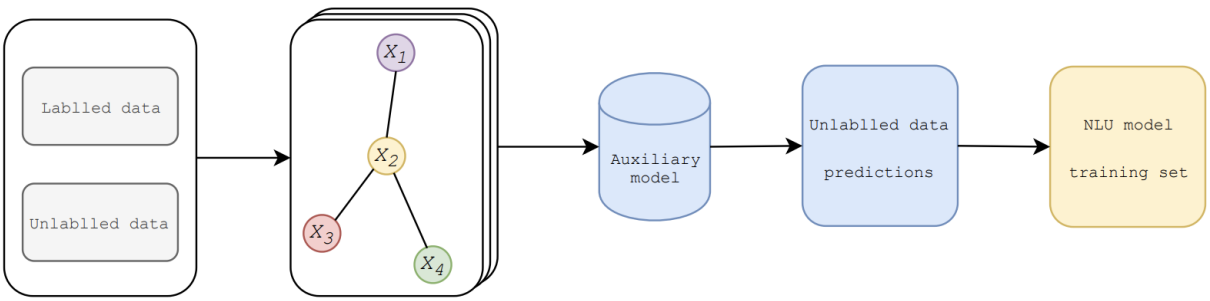


Figure 4. Inductive semi-supervised learning pipeline

This experiment proves applying inductive graph-based semi-supervised learning is an efficient method by reducing the error rate of the NLU model by 5% in SNIPS dataset [2].

Rasa: Open Source Language Understanding and Dialogue Management

Rasa is an open source python library for constructing conversational software with minimal (or no) initial training data. It consists of two parts: Rasa NLU and Rasa Core. Dialogue management problem can be handled as a classification problem. At each iteration, Rasa Core predicts which action to take from a predefined list. On the other hand, Rasa NLU is a tool for natural language understanding. It combines a number of natural language processing and machine learning libraries in a consistent API.

First a message is received and passed to Rasa NLU to extract the intent, entities, and the other structured information. Then the conversation state saved in the tracker which receives a notification that a new message has been received. In step 3, the policy receives the current state of the tracker and chooses which action to take next. Then chosen action is logged by the tracker and executed. If the predicted action is not 'listen', go back to step 3. After the first step all the remaining steps are performed by Rasa Core.

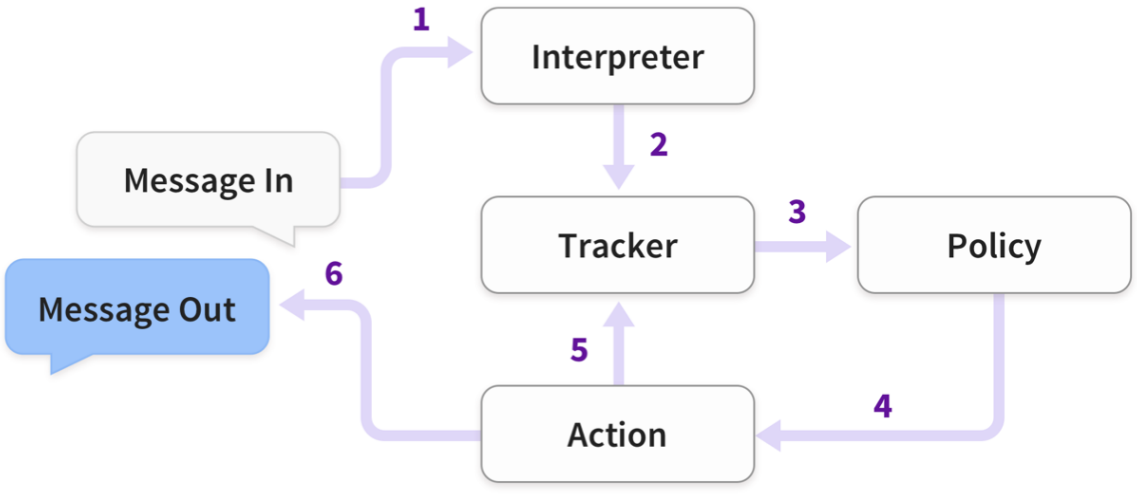


Figure 5. Set of steps Rasa applies when a message received

Rasa NLU's and Rasa Core's training data formats are human-readable. A list of utterances with intents and entities is required by Rasa NLU.

```
{
  "text": "show me chinese restaurants",
  "intent": "restaurant_search",
  "entities": [
    {
      "start": 8,
      "end": 15,
      "value": "chinese",
      "entity": "cuisine"
    }
  ]
}
```

Figure 6 . json syntax

```
## intent: restaurant_search
- show me [chinese](cuisine) restaurants
```

Figure 7. markdown syntax

To specify training dialogues, also known as stories, Rasa Core uses markdown. The body of a story is a series of events. The general format is where entities are pairs of key-values separated by commas.

Rasa Core has the ability to visualize the training dialog graph. The story graph has actions as nodes. User utterances that exist between the execution of two actions are used in labeling edges.

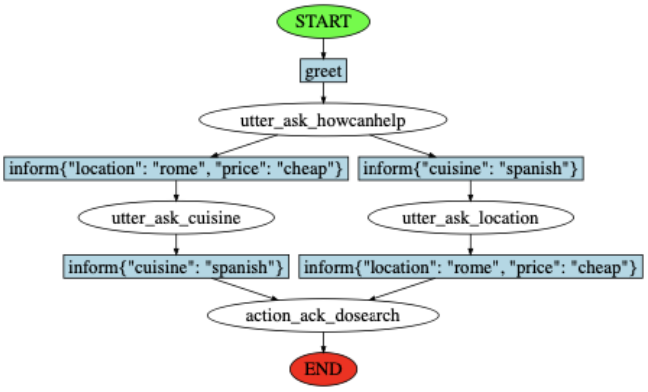


Figure 8. Story graph

Rasa NLU as well as Core are both in active development. They serve as a platform for non-specialist developers to make applied research in conversational AI accessible and will never be 'done' [3] .

Deep Natural Language Understanding of News Text

This paper summarized applications of abductive reasoning to the deep understanding of a paragraph of news text, which is a 75-word paragraph from the New York Times article “How Uber Uses Psychological Tricks to Push Its Drivers’ Buttons” , where the subtle meaning of words and phrases are resolved by backward chaining on a hand-authored knowledge base of 80 axioms. The goal was to explore the scope of the axioms that were required, and to determine whether the correct interpretation of the whole paragraph which is a proof graph can be derived using recent advances in incremental abductive reasoning. Particularly they used modified version of incremental Etcetera Abduction which treats the entire paragraph as a single input sequence while ignoring sentence boundaries.

Even though they were able to run the entire 75-word paragraph with the proof graph and generate the right interpretation. "How will it do in the next paragraph, and the next?" is an ongoing issue [4].

NLU Datasets:

Evaluating Natural Language Understanding Services for Conversational Question Answering Systems

Even though there is no universal model for chatbot design, this paper proposed an universal chatbot architecture. And it contains three main parts as shown in Figure 12 : Request Interpretation, Response Retrieval and Message Generation.

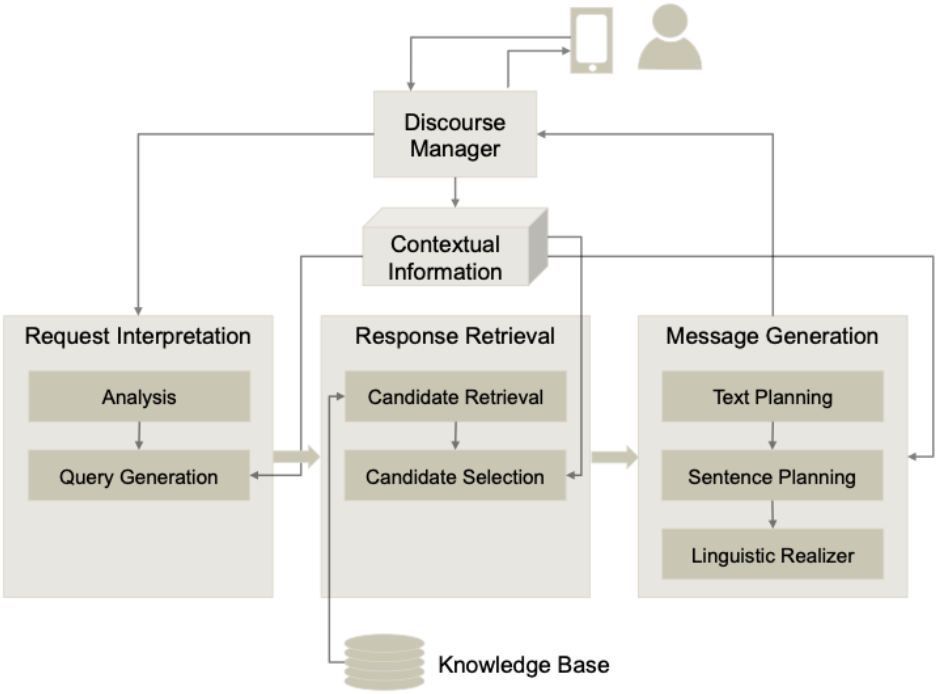


Figure 9. General Architecture for Chatbots

The main objective of NLU services is from unstructured natural language input, such as chat messages, extract structured semantics by adding user-defined labels to messages. A comparison of the basic functionality provided by the most common NLU services is shown in Table 2. All of them follow the same basic principle, excluding Amazon Lex: the user can train a classifier to identify intents and entities depending on example data.

Service	Intents	Entities	Batch import
LUIS	+	+	+
Watson	+	+	+
API.ai	+	+	+
wit.ai	+	+	O
Lex	+	O	-
RASA	+	+	+

Table 3. Intents and Entities for each service

Two corpora,one consisting of annotated questions and one consisting of annotated questions with the corresponding answers, is also presented. The Chatbot Corpus is based on questions

gathered by a Telegram chatbot in production use, answering questions about public transport connections while the StackExchange Corpus is based on data from two StackExchange platforms: ask ubuntu and Web Applications.

Based on these corpora, the paper conducted an evaluation of some of the most popular NLU services such as LUIS, Watson Conversation, API.ai, wit.ai, Amazon Lex and RASA.

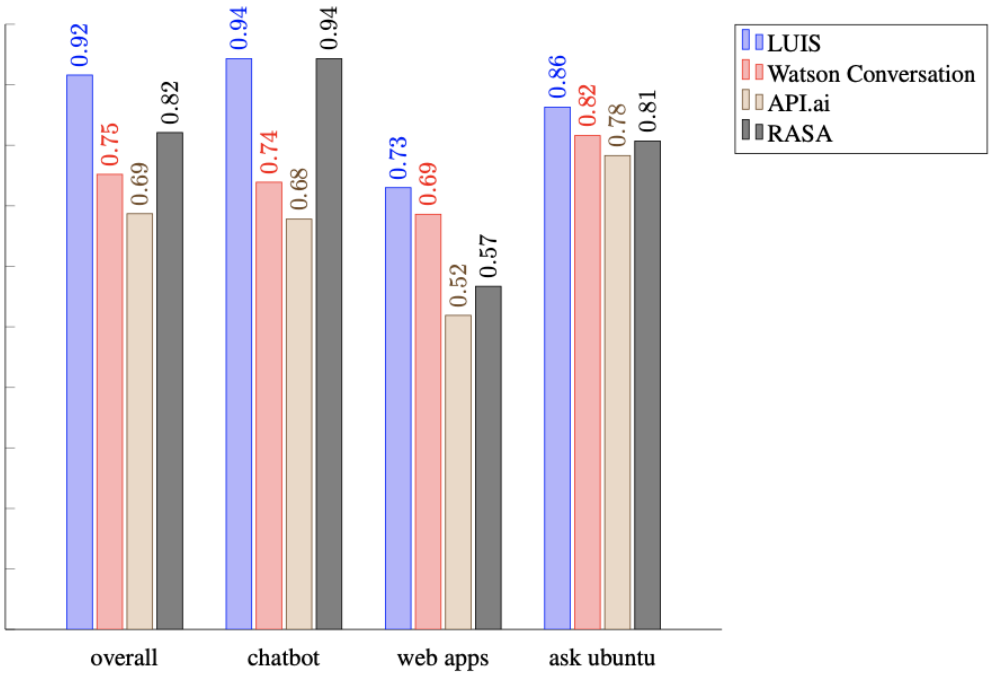


Figure 10. F-scores for different NLU services, grouped by corpus

Even the LUIS showed the best result for two corpora, the paper states it might be possible to achieve better results with RASA due to its open source solutions adaptability [5].

Effective Crowdsourced Generation of Training Data for Chatbots Natural Language Understanding

In this Paper, an end-to-end solution is proposed for collecting training data using crowdsourcing is presented.The end-to-end pipeline follows Dataset Generation, Dataset Enrichment and finally Model Training.

With the support of crowd workers, this paper implemented a generic approach to producing training data and then examined the workflow of the approach as well as the creation of high quality crowdsourcing tasks. By conducting an experiment gathering data for 9 different intents, they tested the approach.

The experiment collects training data for 9 different intents from 3 different domains, which are travel information, meeting scheduling, and software development conducted and shown in Table 3.

		TRAVEL	MEETING	SOFTWARE
READ	Intent	Ask for navigation	Availability check	"How to" questions
	Parameters	source, destination, mode of transport	time, alternative time, place	progr. language, OS, package/tool
CREATE	Intent	Purchase a ticket	Create a meeting	Deploy software
	Parameters	source, destination, trip purpose, date	time, participants, place, duration	action, OS, memory requirement
UPDATE	Intent	Modify a ticket	Modify a meeting	Modify software
	Parameters	source, destination, date	time, participants, place, duration	error, progr. language, OS, package/tool

Table 4.Training data

In each domain the paper tested the approach with 3 types of intents: *1) read* - where users retrieve some information, *2) create* - where users intend to perform a new transaction, *3) update* - where users intent to edit information.

		BRAINSTORMING	VALIDATION		
		Accuracy, %	Accuracy, % T = 0.33	Accuracy, % T = 0.66	Accuracy, % T = 1.00
TRAVEL	Read	90,15	97,57	98,92	99,58
	Create	94,39	97,14	98,7	99,63
	Update	79,36	86,53	90,31	94,87
MEETING	Read	88,25	91,47	96,46	97,78
	Create	98,82	99,47	99,73	99,85
	Update	81,85	92,46	95,87	99,2
SOFTWARE	Read	89,83	93,71	97,09	98,21
	Create	90,79	94,17	96,49	97,16
	Update	84,5	89,59	93,72	97,63
	Mean	**88,66**	**93,57**	**96,37**	**98,21**

Table 5. Training data generation pipeline results.

The paper states achieving 97 percent accuracy for read and create intents shows that the method can already be used for obtain large corpus of training data, as this level of accuracy has been achieved for all 3 domains.

And Rasa NLU, which's intent classifier is based on support vector machines (SVM), is trained with the data that collected using end-to-end pipeline and reported how the performance differs with training set size and intent.

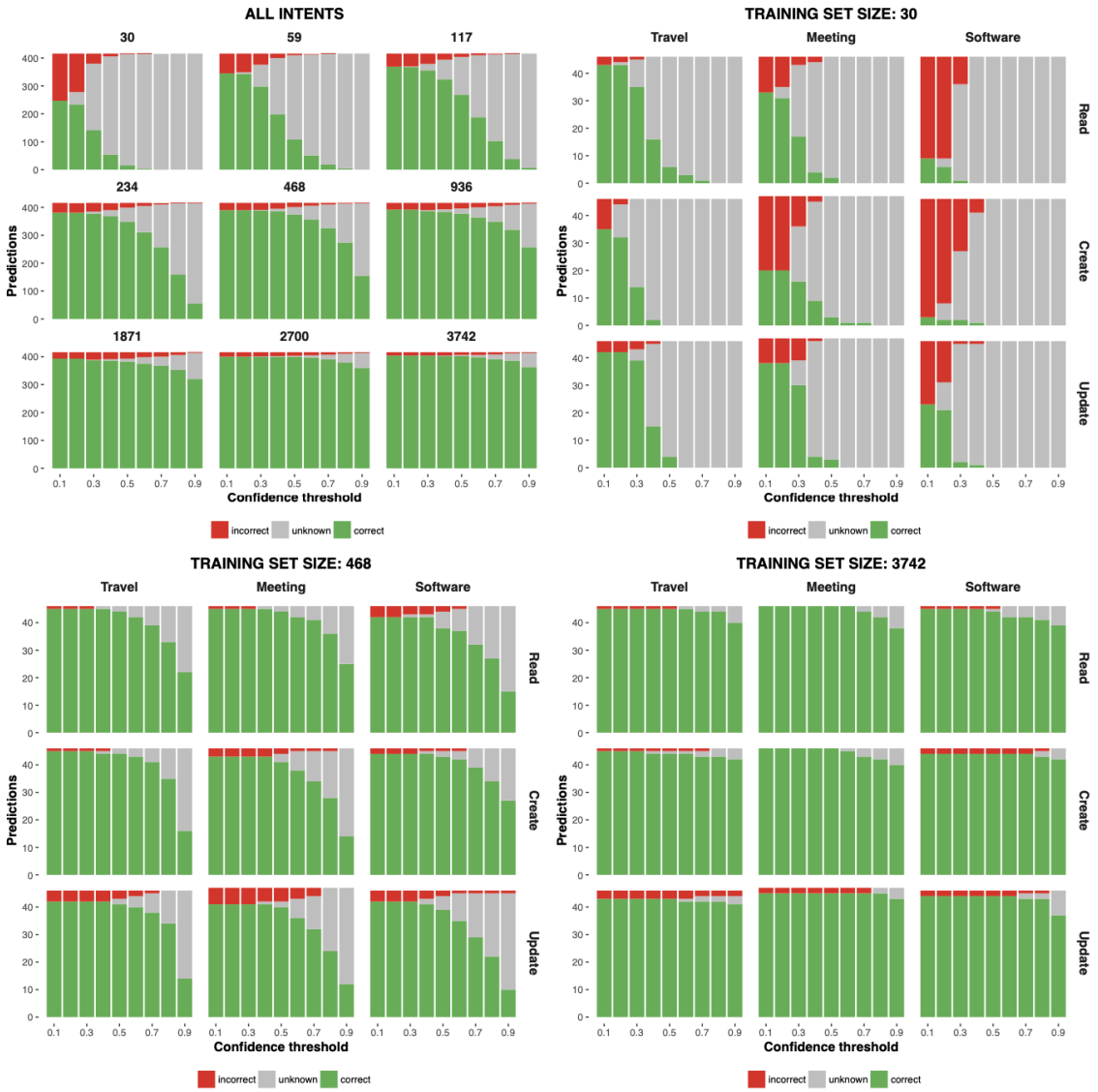


Figure 11. Performance comparison of NLU models trained with data of a different size.

Figure 11 shows that various intents' performance is not consistent, such as Travel Read performance is over 90 percent with a very limited training data set of only 30 training examples, while Software Create is below 10 percent. All test requests were correctly classified with training size 3742 for intent Meeting Create.

The model has very low efficiency, looking at the combined performance (the bottom part of the figure) with training data of less than 117 request samples (less than 90 percent).

Improving the training data 16 times allows us to achieve an accuracy level of 95 percent. The further increase in dataset size only slowly improves its efficiency [6].

Available Language Models:

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

BERT is one of the available language models which uses unsupervised learning in the pretraining part. It stands for Bidirectional Encoder Representations from Transformers since it applies a Transformer that learns from connections between words or sub-words in a text. While classical models use the input as left-to-right or right-to-left, the Transformer encoder takes all of the sequences at once for a bidirectional approach.

This implementation considers pre-training and fine-tuning that are key steps. In pre-training, BERT uses Masked LM (MLM) and Next Sentence Prediction (NSP) one after another. In order to begin the training, first, the MLM process masks 15% of the inputs randomly in each sequence for predicting masked tokens by considering only non-masked words. After that, for the aim of understanding relationships of the sentences, BERT uses Next Sentence Prediction (NSP) with selecting two sentences 50% of the time tagging by isNext, and 50% of the time tagging notNext in each pretraining depending on whether the next sentence is.

To summarize, BERT provides a deeper understanding by combining left-to-right and right-to-left context analyzes. According to test results on GLUE datasets, BERT has 82.1% on average that is the highest score [7].

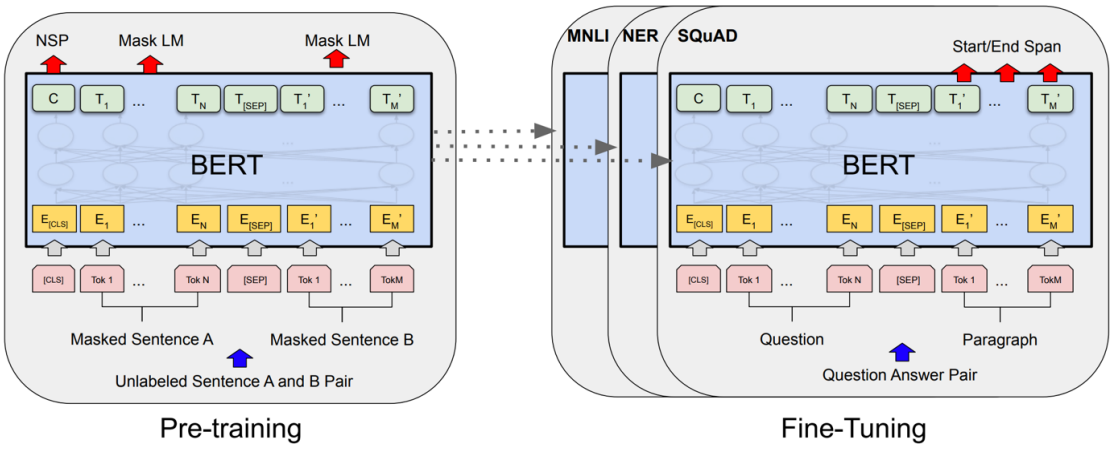


Figure 12. Pre-training and Fine-tuning steps of the BERT

XLNet: Generalized Autoregressive Pretraining for Language Understanding

XLNet is a Generalized Auto-Regressive Model that takes BERT's bi-directional context modeling to the next level. BERT loses the connections between masked words using the mask system in the pretraining that causes inconsistency in finetuning. For example, incorrect prediction can occur when one of the two-word proper nouns is masked. XLNet beats BERT over 20 tasks with the autoregressive pretraining model that allows all possible permutations of the factorization order by maximizing.

In comparison with the BERT, BERT uses unsupervised learning for predicting randomly masked inputs in the pretraining stage, thus deducing the meaning of the word from the previous and next words in a bidirectional way. This approach creates errors in words that have a common meaning. For example, while masking New in the word New York and guessing Francisco creates output like Francisco York which is not true, thus this issue lowers the accuracy of the model.

Rather than BERT, XLNET uses the autoregressive model according to probability distributions to cover this problem. While doing this it gathers information from all positions on both back and forward sides [8].

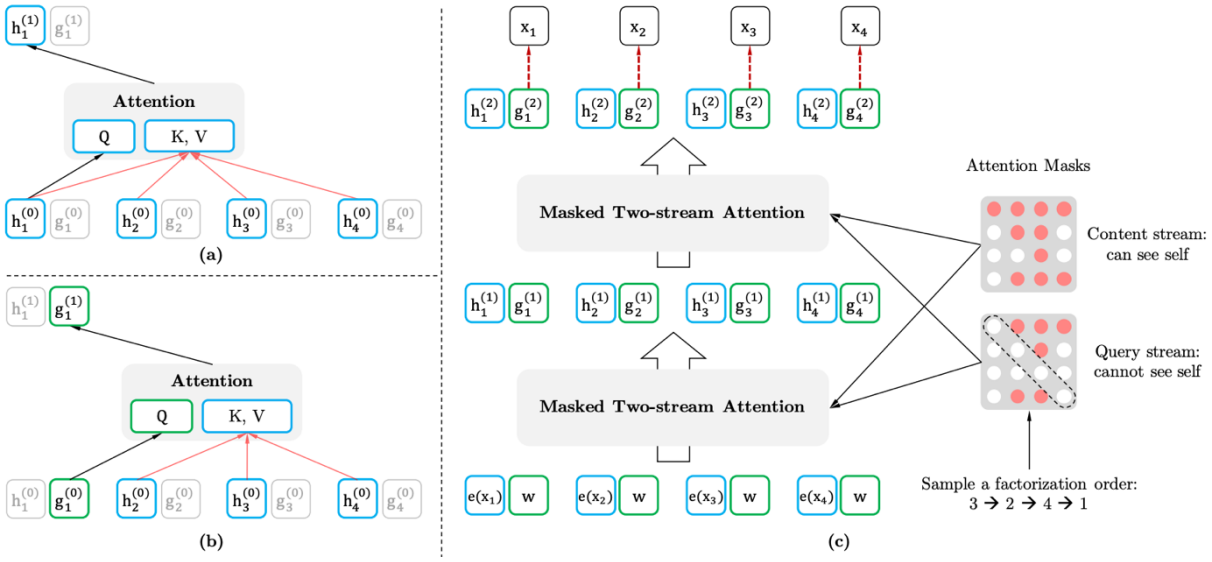


Figure 13. Architecture that used in XLNet

3.2. Solution Method - RASA

Many methods are used to create smart assistants, but Rasa is the most popular. Rasa draws inspiration from a variety of sources to build a conversational AI chatbot. The most important reasons besides it is open source is the Rasa API uses famous machine learning libraries and deep learning frameworks such as Tensorflow, scikit-learn, and Keras, and both of these libraries are components of the Rasa application, and the text classification is loosely based on the fastText approach.

4. ANALYSIS AND MODELLING

4.1. System Factors

The most fundamental factor that will affect the system is the Machine Learning model used by RASA in the training process. In addition, the number of intents and entities in the dataset and the number of story samples may be among the factors that affect. Training the model is highly complex work and if the amount of the features used in training increases, the operation of the system begins to slow down. RASA is a framework that works in the memory of the computer, so it cannot deal with more than the computer can handle, which may cause a slowdown or false results.

4.2. How System Works

Our project is a chatbot builded with the help of RASA. In this project, firstly we created a Natural Language Understanding model to help Rasa to understand user messages, and then train a model. And we taught the chatbot how to respond to user messages using stories and rules. We continued with defining the universe that chatbot lives such as what user inputs it should expect to get, what actions it should be able to predict, how to respond, and what information to store. We finished with re-training a neural network on our example stories and NLU data.

4.3. Modelling

In this Senior Design Project, as we explained before in 2.2.3 Use Cases chapter, our use cases are " Search Flights", "Search Hotels", "Search Events", "Search Restaurants" and "Ask Weather". After the user searches the domain the chatbot returns the results.

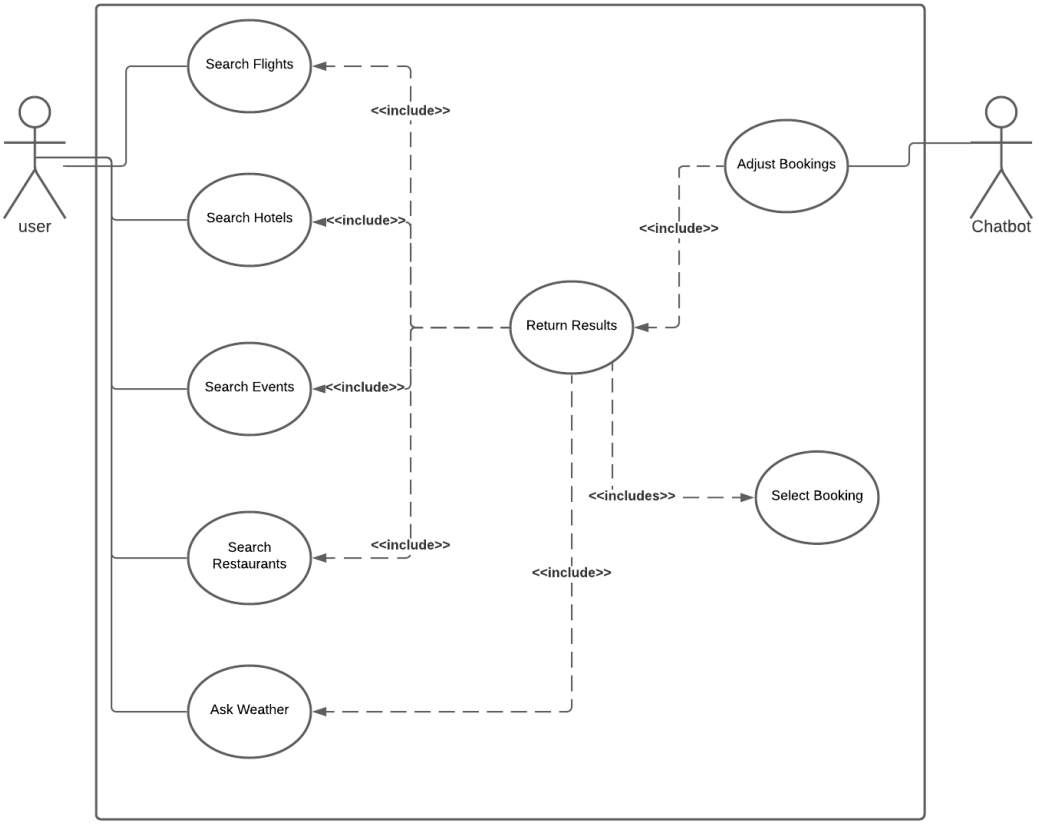


Figure 14. Use Case Diagram of the project.

4.3.1. System Architecture

RASA is a platform for creating conversations with AI. It has two components; NLU and Core. NLU tries to understand what the user wants to say and detects contextual information. Core provides the chatbot to give the most appropriate answer by considering the history of the conversation.

Rasa NLU:

Since NLU tries to understand the input sentence, it uses intent classification and entity extraction. In this example, the program finds the proper intent for the sentence then extracts each word for finding the entity by using recognition.

Rasa Core:

Core is basically a dialog engine. It uses a machine learning model trained on example conversations to decide what to do next. It provides the chatbot to return a response by

combining the intent and entity information. Chatbot continues the conversation by going to the next action after the reply is returned.

Components of Rasa:

__init__.py	An empty file that helps python to find your actions
actions.py	Code for custom actions
config.yml	Configuration of your NLU and Core models
credentials.yml	Details for connecting other services
data/nlu.yml	Your NLU training data
data/stories.yml	Your stories
data/rules.yml	Rules for conversation flow
domain.yml	Your assistant's domain
endpoints.yml	Details for connecting to channels like messenger
models/<timestamp>.tar.gaz	Your initial model

Rasa Flow Diagram:

If we examine the diagram, after NLU determines intent and entity, it sends these entries to the core and core returns a response. Conversation can be improved by using ai methods. Here, Core tries to get a result by making use of the example of the stories. For this reason, it is important that the dataset and stories that are given to the program should be properly defined. Because they affect the accuracy of the training.

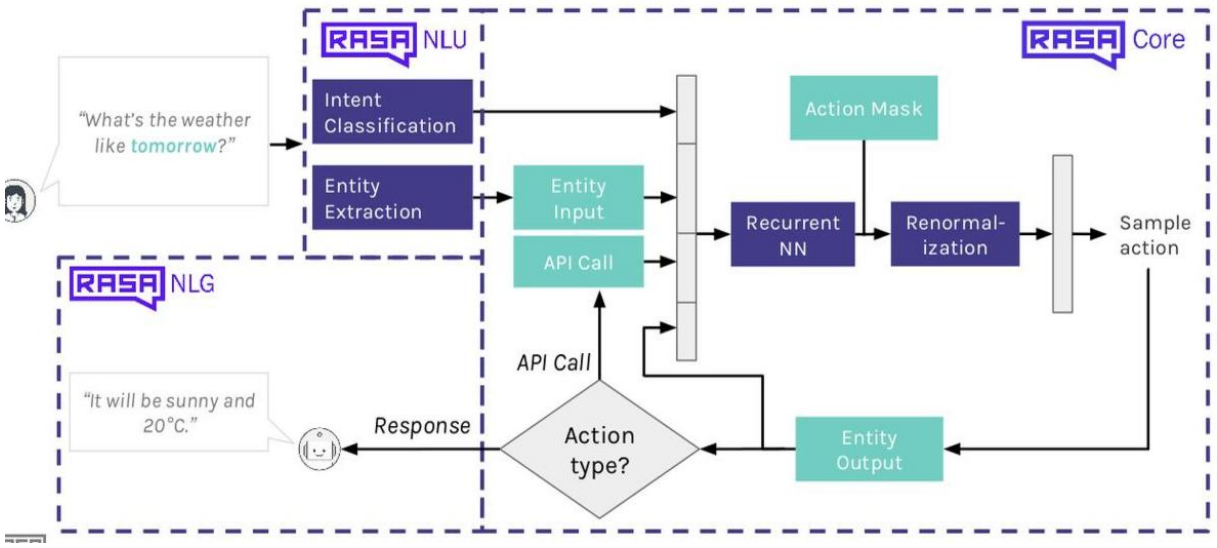


Figure 15. The architecture of RASA

4.3.2. UML (Unified Modeling Language) Diagrams

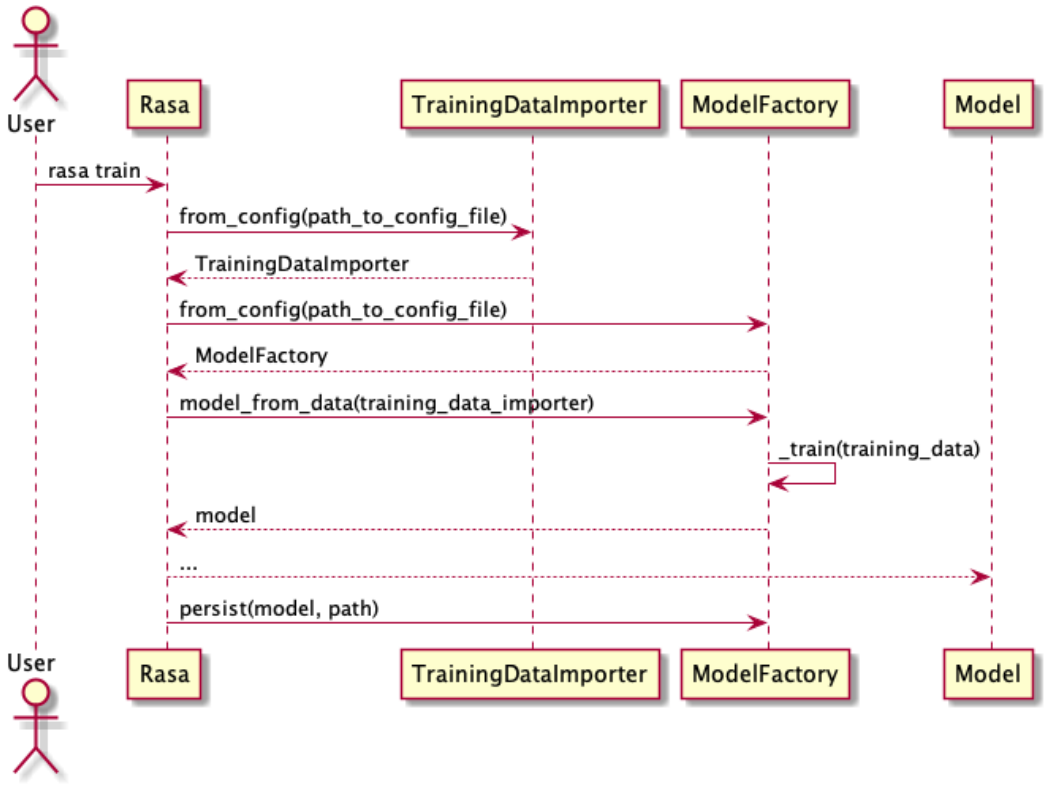


Figure 16. UML Diagram of rasa train workflow

Figure 16. shows how a model can be trained using a Training DataExporter when the rasa train is runned.

5. DESIGN, IMPLEMENTATION AND TESTING

5.1. Design

We designed our system based on the architecture of the RASA. Our program is a chatbot that can be conversable by giving input from the terminal. After creating our dataset with intents and entities, we started to set up the system according to the RASA architecture. First of all, we prepared dialogues that could be examples and added them to the stories section of the system. In RASA, we created rules for the responses to be given when certain intents come. After that, we added the responses of the chatbot to the domain part. Since the entities of the intents may be variable, it was necessary to fill the slots with the entities entered by the user. For this reason, we added codes in Python language to the actions section to fill the slots. Thus, our system has been designed to make predictions and respond according to the dynamics of the conversation. After we implemented the conversation progress, we designed an interface using the Flask API, therefore the user can talk to the chatbot comfortably from an appropriate interface.

5.2. Implementation

We implement our code according to the RASA architecture. First, we prepare a dataset that contains intents with entities in "data/nlu.yml". Since intents and entities will be used when trying to understand what the user wants to say, it was very important to be appropriate, we took care to make it accordingly. In the end, we have 93 intents 29 entities in total and average 7 sentences for each intent as you can see in the Table 6.

intent name	# of sentences	entity name
enquire_flight_info	1	
enquire_flight_details	10	
enquire_hotel_details	4	
desired_city	4	city
desired_room	4	person
faqbreakfast	5	
faqcancellationpolicy	4	
faqcheckin	5	

faqcheckinage	4	
faqcheckout	3	
faqchildrenage	4	
faqearlycheckin	3	
faqlatecheckin	3	
faqlatecheckout	4	
faqroomcount	3	
faqbabycot	8	
faqbreakfasttime	3	
faqrefundable	4	
faqreceptiontime	4	
faqparking	10	
faqparkingcost	4	
faqpartialcancel	3	
sss	1	
faqpetpolicy	6	
faqreservaconfirm	5	
faqnearestairport	5	
faqmodifyreservation	3	
faqmeetingroom	4	
sssecurity	4	
faqluggage	7	
faqcontactnumber	6	
enquire_pnr	4	pnr_num
enquire_train_cancel	6	
enquire_train_fare	3	
enquire_train_quota_info	2	
enquire_train_route	7	train_num
enquire_train_seat	4	
enquire_train_station	4	
enquire_train_station_code	5	city

enquire_train_station_name	5	station_code
enquire_trains_class_info	2	
goodbye	17	
greet	22	
mood_great	10	
mood_unhappy	9	
inform_age	3	age
inform_city	6	city
ask_time	14	city
ask_which_events	28	event,date_time,city
weather	17	weather
weather_address	7	city
weather_address_date_time	9	date_time, city
inform_class_code	4	class_num
inform_destination_airport	5	dcity
inform_destination_station_name	4	station
inform_flight_date	6	flight_date
inform_no_of_adults	4	adult
inform_no_of_children	6	children
inform_no_of_infants	4	infant
inform_pnr	4	pnr_num
inform_quota_code	4	quota_code
inform_seating_class	7	class_num
inform_source_airport	4	scity
inform_source_station_name	5	station_name
inform_station_code	4	station_code
inform_train_date	4	train_date
inform_train_number	5	train_num
main_menu	4	

plan_my_trip	3	
activities_offered	5	
more_info_zipline_tour	5	event
add_to_mytrip	4	
contact_us	1	
references	1	
about_istanbul	3	city
tips_and_tricks	3	
other_activities	4	
selected_list	5	
desired_event	5	
selected_event	16	event
got_total_tickets	3	ticket
got_email	3	mail
selected_no	9	
restaurant_search	33	
telling_category	23	category
telling_cuisine	25	cuisine
telling_datetime	10	date_time
telling_location	20	place
telling_location_cuisine	19	place, cuisine
telling_numpeople	17	person
telling_phoneno	10	phone_no
affirm	63	
deny	30	
Average	7,698924731	

Table 6. Entities for each intents

After preparing the dataset with intents and entities (if it needs), we need to utilize responses for user's input sentences. The responses to be sent by the chatbot are kept in a file named "domain.yml". In this file, we have defined the intents, entities, slots and forms if needed,

actions and created the utters of chatbot. Since the entities are variable, it is determined whether the entities will affect the conversation by filling slots. If any slots will not be filled by the entity of the user's input creating a response and using it in the story is sufficient. In occasions where entities will be used from a user's sentence, slot filling will be applied. Thus, entities can contribute to the training process, programs can return a response with an entity or operations can be performed using the entity (called and API, search on database, etc).

For the aim of filling the slots, firstly, the slots are defined with necessary features as in Figure 17. Instead of explaining all of them, let's go from the example we created for booking an event.

```yaml
slots:
  event:
    type: text
    influence_conversation: false
  ticket:
    max_value: 9.0
    min_value: 1.0
    type: float
    influence_conversation: false
  mail:
    type: text
    influence_conversation: false
```

Figure 17. A few examples of slots

After that, these slots are assigned in the form by creating a form structure. In this structure, which entity will fill which slot is specified with the type of each entity as in Figure 18. For this example, they are event, ticket and mail.

```yaml
forms:
  event_form:
    event:
      - type: from_entity
        entity: event
    ticket:
      - type: from_entity
        entity: ticket
    mail:
      - type: from_entity
        entity: mail
```

Figure 18. An example of form

After the necessary definitions are made, the python code named "action.py" is used to fill the slots. As seen in Figure 19, the form able to call validation method with its specified name, finds the entities from the user's sentence and fills the slots accordingly.

```python
class ValidateEventForm(Action):
    def name(self) -> Text:
        return "event_form"

    def run(
        self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict
    ) -> List[EventType]:
        required_slots = ["event", "ticket", "mail"]

        for slot_name in required_slots:
            if tracker.slots.get(slot_name) is None:
                # The slot is not filled yet. Request the user to fill this slot next.
                return [SlotSet("requested_slot", slot_name)]

        # All slots are filled.
        return [SlotSet("requested_slot", None)]
```

Figure 19.Example action function

In "domain.yml" there are responses to be given by chatbot. Chatbot's responses should start with "utter_" and if an action is to be done using a slot, the name of the response should be "utter_ask_ <slotname>" as in Figure 20. If the response is to be given with slots, they are added to the sentence in the response utter with the form of "{slotname}".

```
utter_ask_event:
- text: "Hangi etkinliğe gitmek istiyorsunuz?"

utter_ask_ticket:
- text: "Lütfen almak istediğiniz toplam bilet sayısını giriniz"

utter_ask_mail:
- text: "Lütfen geçerli bir e-posta adresi giriniz"

utter_event:
- text: "{event} etkinliği için {ticket} adet biletiniz alındı, bilet bilgileriniz {mail} adresine gönderildi.
Yardımcı olabileceğim başka bir konu var mı?"
```

Figure 20. Example responses

Putting the slot's variables into sentences takes place in the function from "actions.py". The function in Figure 21 is called with the name of the utter in the responses from which slots will be invoked. In the function, the slots in the utter are filled. In addition, the names of these actions must be defined in the "domain.yml" file as well.

```
class ActionSubmitEvent(Action):
    def name(self) -> Text:
        return "action_submit_event"

    def run(
        self,
        dispatcher,
        tracker: Tracker,
        domain: "DomainDict",
    ) -> List[Dict[Text, Any]]:
        dispatcher.utter_message(template="utter_event",
                                 event=tracker.get_slot("event"),
                                 ticket=tracker.get_slot("ticket"),
                                 mail=tracker.get_slot("mail"))
```

Figure 21. Action function for responses

In order for RASA to predict the flow of conversation and give a response according to this prediction, it was necessary to create sample stories. For this reason, we tried to improve the train by adding sample stories to the "data / stories.yml" file. Stories are created according to the flow of the conversation and if slots are used, the defined forms are used here. Based on the example in Figure 22, two different story possibilities are created according to the user saying yes or no while booking an event. In stories, the program understands the intent from

the user's sentence and calls the corresponding form as action, and this form remains in the loop until the user enters the correct entities to fill the slots.

```
stories:
- story: ordering event ticket for yes 1
  steps:
  - intent: desired_event
  - action: event_form
  - active_loop: event_form
  - slot_was_set:
      - requested_slot: event
      - requested_slot: ticket
      - requested_slot: mail

- story: ordering event ticket for no 1
  steps:
  - intent: desired_event
  - action: event_form
  - active_loop: event_form
  - slot_was_set:
      - requested_slot: event
      - requested_slot: ticket
      - requested_slot: mail
```

Figure 22. Example stories

In RASA, when some specific intents come, you can return special answers to them, and we did this by defining it in the "data / rules.yml" file. Since we cannot write all the possibilities of when the user will want to buy a flight or when a concert ticket, we have set rules according to the user's wishes in this file. For the aim of doing this, RASA has a feature called rules which is defined in "rules.yml". According to this feature, the program can return answers to certain patterns. As seen in Figure 23, the chatbot can automatically return a response when the user makes a greet or farewell. Apart from that, at any point of the conversation, when the user asks for something different (for example, hotel reservation), chatbot can respond to this by activating the form in line with these rules.

```yaml
rules:
  - rule: user greet
    steps:
      - intent: greet
      - action: utter_greet

  - rule: user bye
    steps:
      - or:
          - intent: selected_no
          - intent: goodbye
      - action: utter_goodbye

  - rule: activate event form
    steps:
    - intent: desired_event
    - action: event_form
    - active_loop: event_form

  - rule: submit event form
    condition:
    - active_loop: event_form
    steps:
    - action: event_form
    - active_loop: null
    - action: action_submit_event
```

Figure 23. Example rules

Finally, test stories are needed in order to train the artificial intelligence method in RASA. In order to do this, in the "test_stories.yml" file, sample sentences and stories are defined together with which intent and actions should be called according to the flow of the conversation in the test stories. One of the test stories we wrote for the booking an event example is shown in Figure 24.

```
stories:
- story: ordering event ticket 1
  steps:
  - user: |
      selam
    intent: greet
  - action: utter_greet
  - user: |
      Konsere gitmek istiyorum
    intent: desired_event
  - action: event_form
  - active_loop: event_form
  - slot_was_set:
      - requested_slot: event
      - requested_slot: ticket
      - requested_slot: mail
  - action: utter_anything_else
```

Figure 24. Example test stories

When all these are combined, the preparations in the RASA section have been completed. In order to run the program, you need to “rasa run actions” on local 1 and “rasa shell” on local 2. Dialog will start after the “rasa shell” command.

As we mentioned in the testing part, we made the configuration of the chatbot by using config light. Thanks to this configuration, the accuracy score of our program has increased while predicting the intent of the user and responding to the text message as well.

Integration with the API is possible in Rasa so that users can talk to the chatbot. We used Rasa Core, Actions, and Flask API for the deployment of the chatbot. Rasa Core handles the conversation flow, utterances, actions and Rasa NLU extract entities and intents. Rasa Core is a server that returns all the responses in the form of a JSON response. These responses are important for the dialog management part, the assistant can further the conversation with the help of this. Rasa Action Server is required to use tools such as custom actions, forms, and responses that allow the assistant to predict the next reply of the user. It works when our assistant predicts a custom action, the Rasa server sends a POST request to the action server with a JSON payload including the name of the predicted action, the conversation ID, the contents of the tracker, and the domain contents. When the action server finishes running a

custom action, it returns a JSON payload of responses and events. Apart from these, we used Flask to implement web API. Flask is a web framework for Python which provides functionality for building web applications, including managing HTTP requests and rendering templates.

5.3. Testing

We used pre-trained language model BERT for our pipeline. We compared three different pipeline configurations: a light configuration, a configuration using ConveRT, and a heavy configuration that included BERT.

In each case we're training a DIETClassifier for combined intent classification and entity recognition for 200 epochs, but in the light configuration we have CountVectorsFeaturizer, which creates bag-of-word representations for each incoming message at word and character levels.

In the heavy configuration instead of CountVectorsFeaturizer there is a BERT model inside the pipeline. And we used a utility component that does the heavy lifting work of loading the BERT model in memory which is called HFTransformersNLP . Under the hood it leverages HuggingFace's Transformers library to initialize the specified language model. Also we added two additional components LanguageModelTokenizer and LanguageModelFeaturizer which pick up the tokens and feature vectors respectively that are constructed by the utility component.

In the convert configuration, the ConveRT components work in the same way. They have their own tokenizer as well as their own featurizer.

To save a lot of compute time we used these language models as featurizers, which means that their parameters are not fine-tuned during training of downstream models in the NLU pipeline.

config	precision	accuracy	f1-score
config-convert	0.571	0.607	0.564
config-heavy	0.604	0.616	0.583
config-light	0.609	0.636	0.598

Table 7. Intent Classification Results

config	precision	accuracy	f1-score
config-convert	0.726	0.952	0.714
config-heavy	0.756	0.946	0.681
config-light	0.765	0.955	0.726

Table 8. Entity Recognition Results

As we can see from the tables the models with pre-trained embeddings perform worse and BERT is not the best option. The more lightweight CountVectorsFeaturizer perform better on all the scores listed above.

While testing the three configurations we initialized for the dataset, we evaluated them in two categories: intent classification and entity extractions. To examine how accurately the model predicts, first, we used the confusion matrix for both intent classification and entity extraction. In the confusion matrix, the horizontal axis contains an intent the model predicted, while the vertical axis has the actual intent. The point where the values in the horizontal and vertical axes intersect gives correct classifications. So we can see if the model has made a mistake or not. The same applies while analyzing predictions of entities.

Besides the confusion matrix, the confidence histogram helps us understand the confidence of the model for both correct and incorrect classifications. The blue bars we see in the confidence histogram indicate the correctly predicted intent or entity, while the red bars indicate misses. While there should be no miss in the ideal model, this is an extraordinary situation that cannot always be achieved.

Config-light:

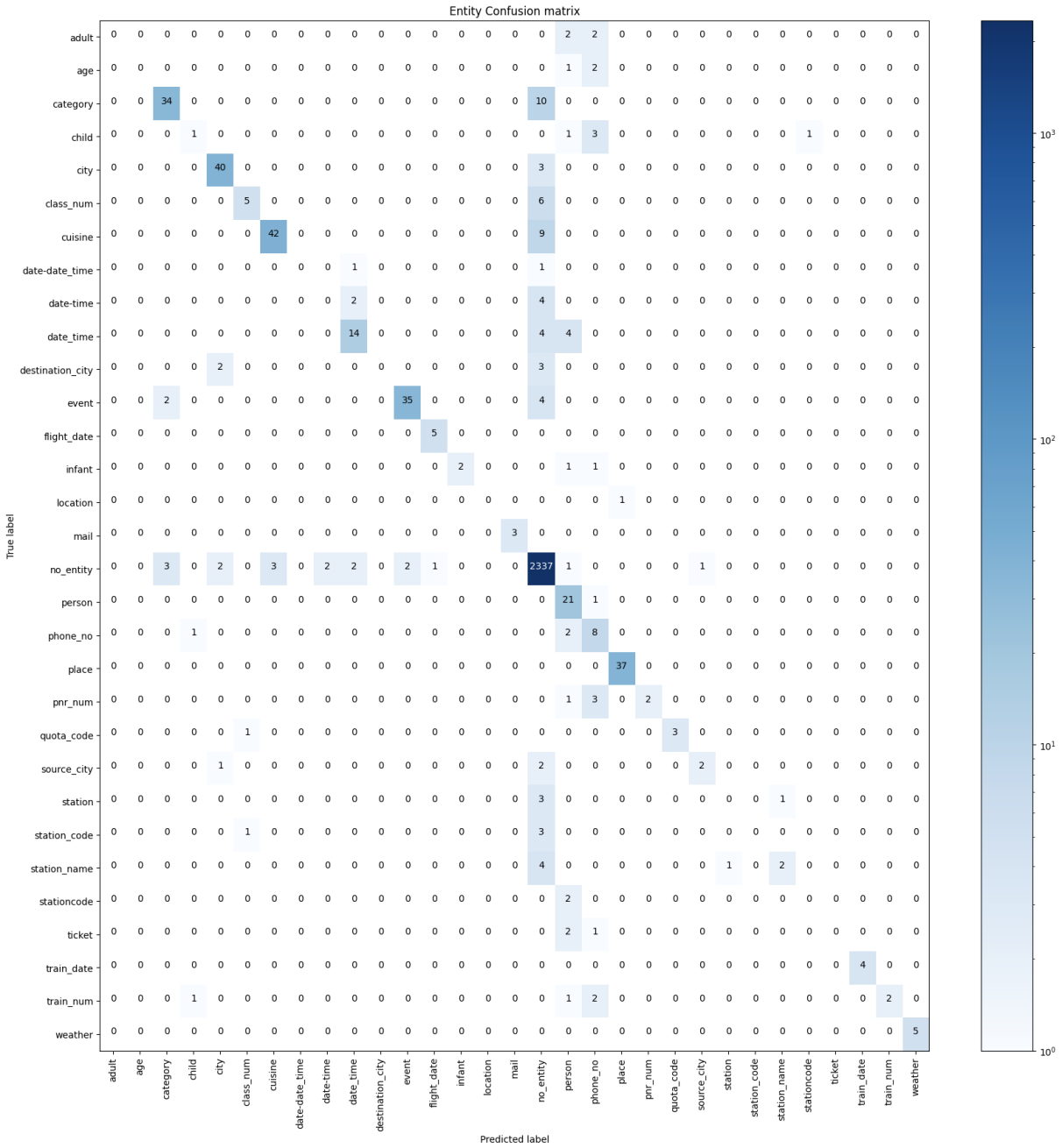


Figure 25. Entity confusion matrix of light configuration

When we look at Figure 25, we see the entity confusion matrix of light configuration. In addition to the not found entities, if we look at train_num and pnr_rum entities as an example, we can observe that in some cases the model predicts they are phone_no entity as a wrong prediction. Still, the model seems to have correctly predicted the majority of entities.

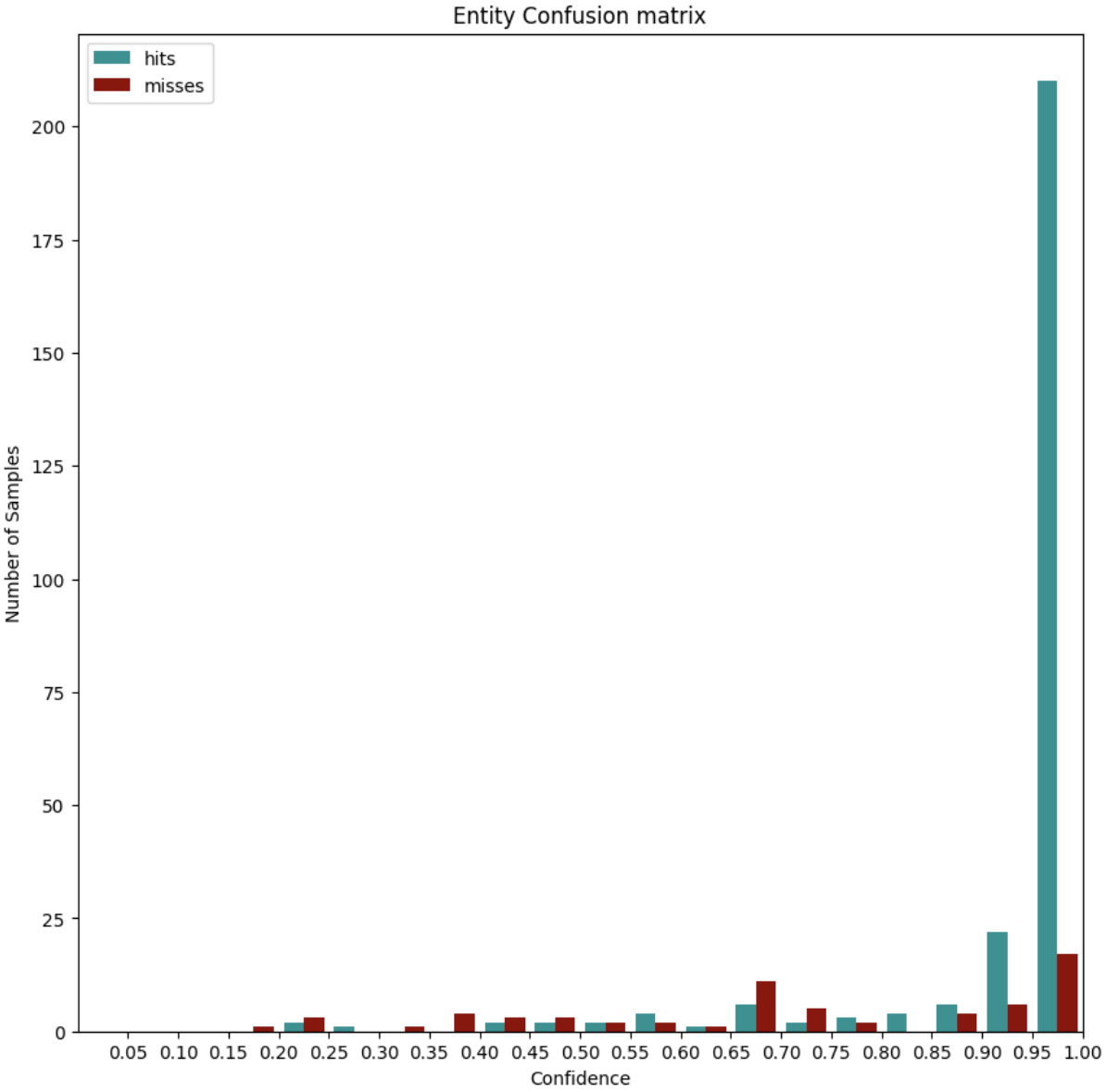


Figure 26. Confidence histogram for entity confusion matrix of light config

When we examine Figure 26, we observe that when it reaches the 200th sample, it has the highest confidence of hits with 0.95.

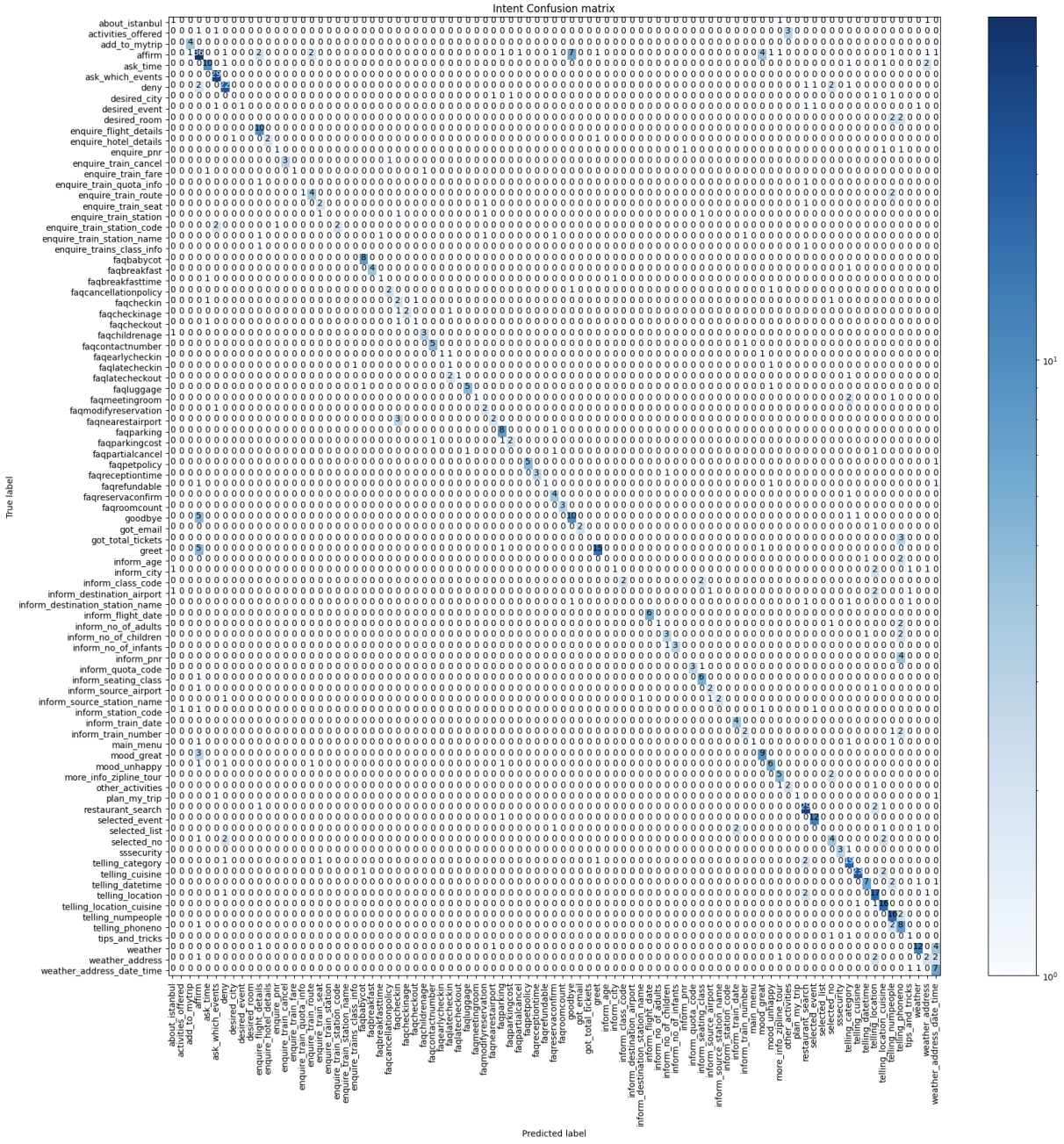


Figure 27. Intent confusion matrix of light configuration

Figure 27 contains the intent confusion matrix obtained from the intent classification of light configuration. Although the intent confusion matrix has mostly made correct predictions, there are also some mistakes. For example, when the model predicts the goodbye and greet intents, it has been determined as the affirm intent which is wrong.

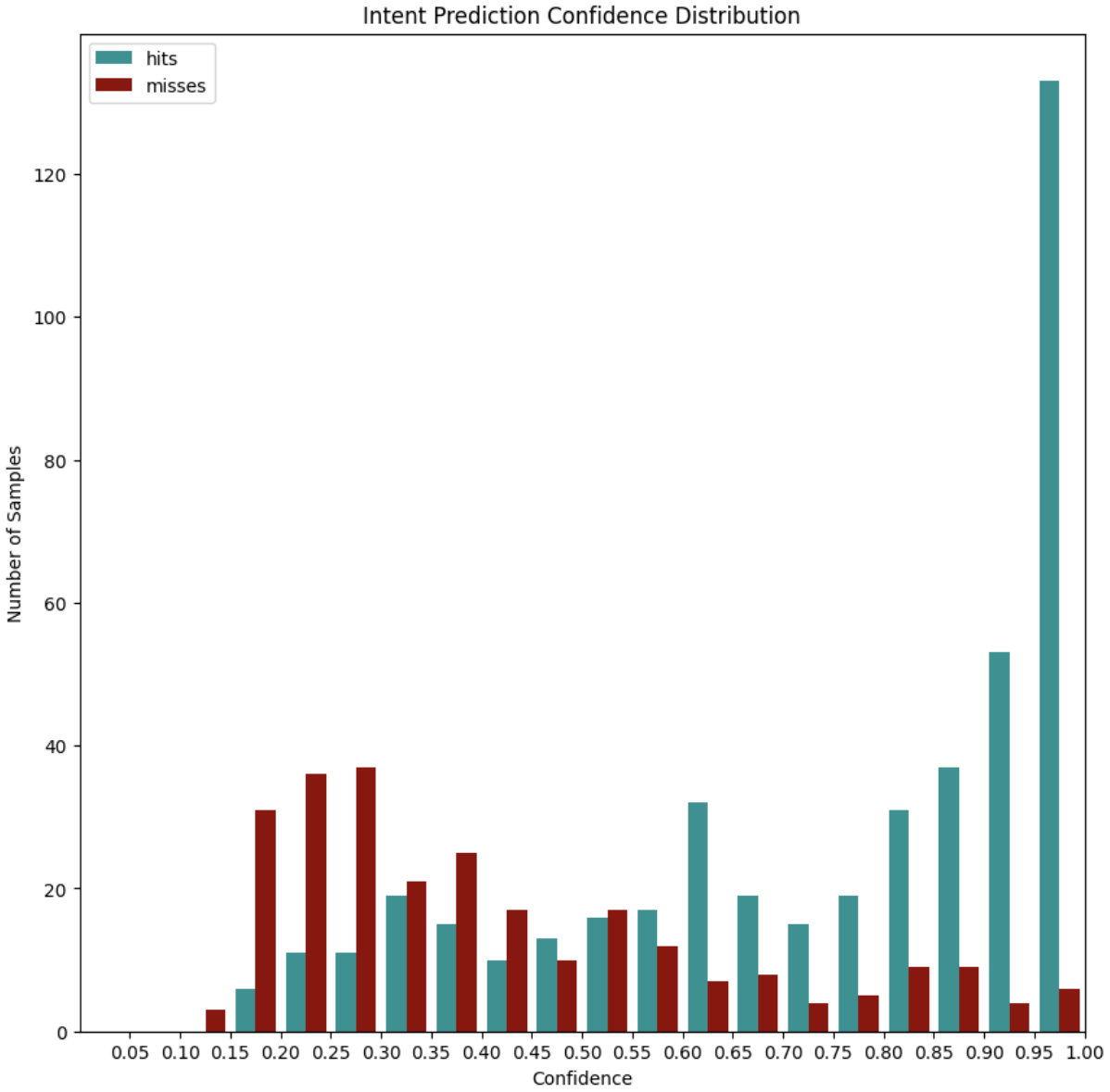


Figure 28. Confidence histogram for intent confusion matrix of light config

Finally, when we examine the confidence histogram of intents in the Figure 28, we see that, unlike the entity, the samples at the beginning make too many misses. However, when the number of samples was over 120, it reached the highest hits of more than 0.95.

Config-convert:

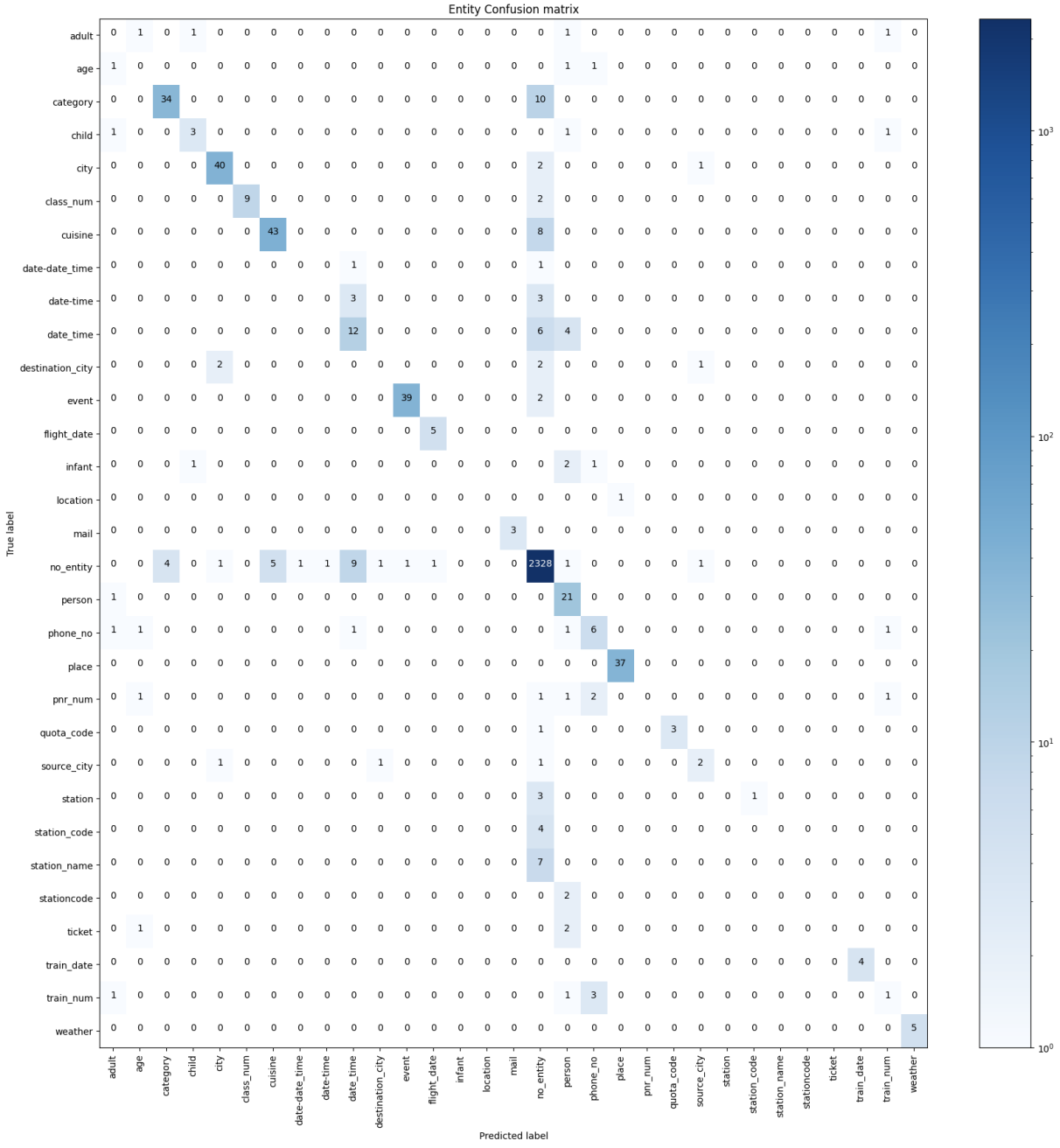


Figure 29. Entity confusion matrix of convert configuration

Figure 29 shows us the entity confusion matrix that results from the entity extraction when we use the convert configuration. Mostly, date-related entities such as date_time, flight_date have been mistakenly predicted by mixing with the person entity. Other than that, entities are mostly predicted correctly, except for no_entities.

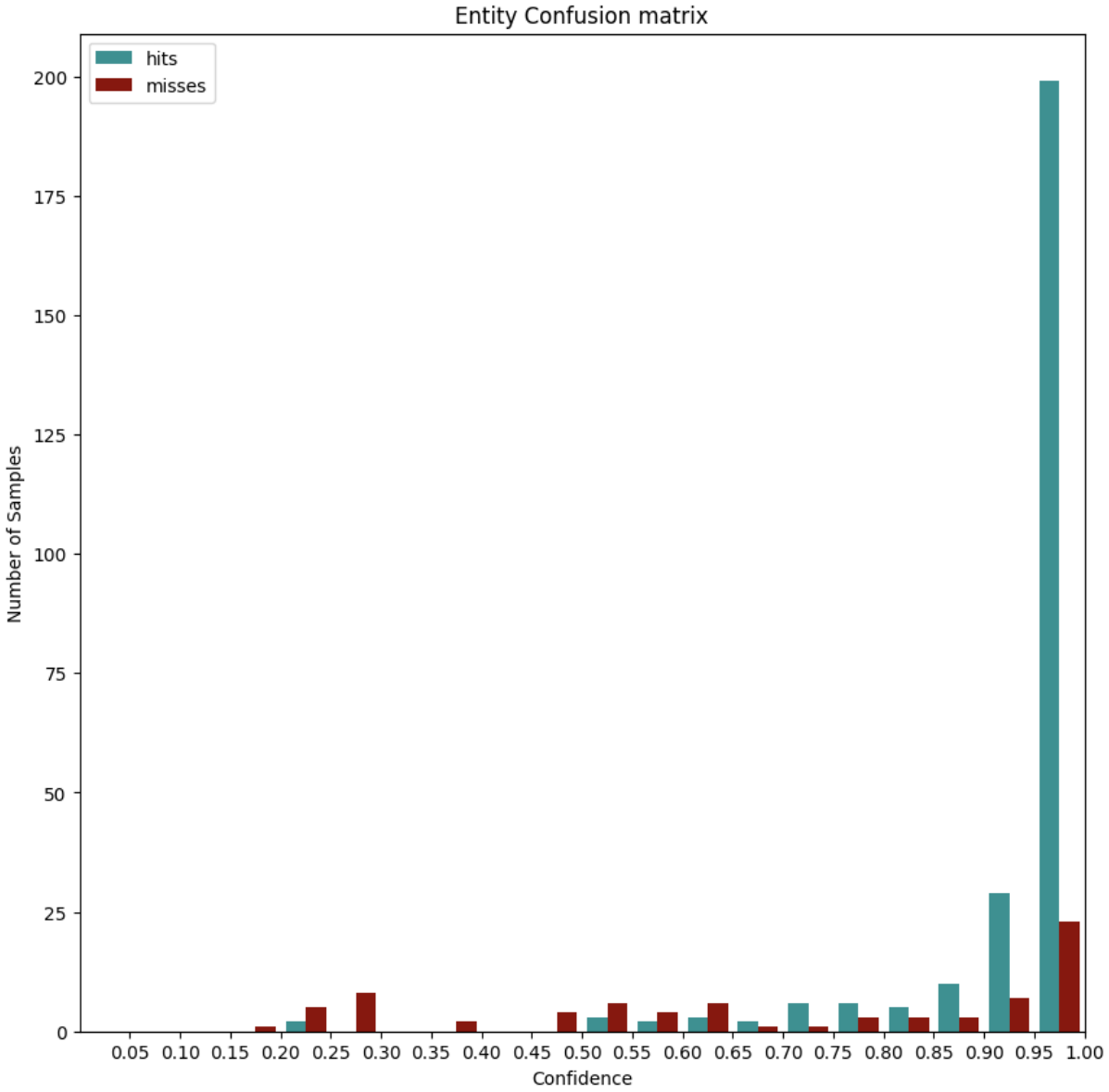


Figure 30. Confidence histogram for entity confusion matrix of convert config

In Figure 30, there is the confidence histogram of performing the entity extraction according to the convert configuration. According to the confidence histogram, the highest hit was reached in the 200th sample. It is also observed that there are no hits in some samples in the early stages.

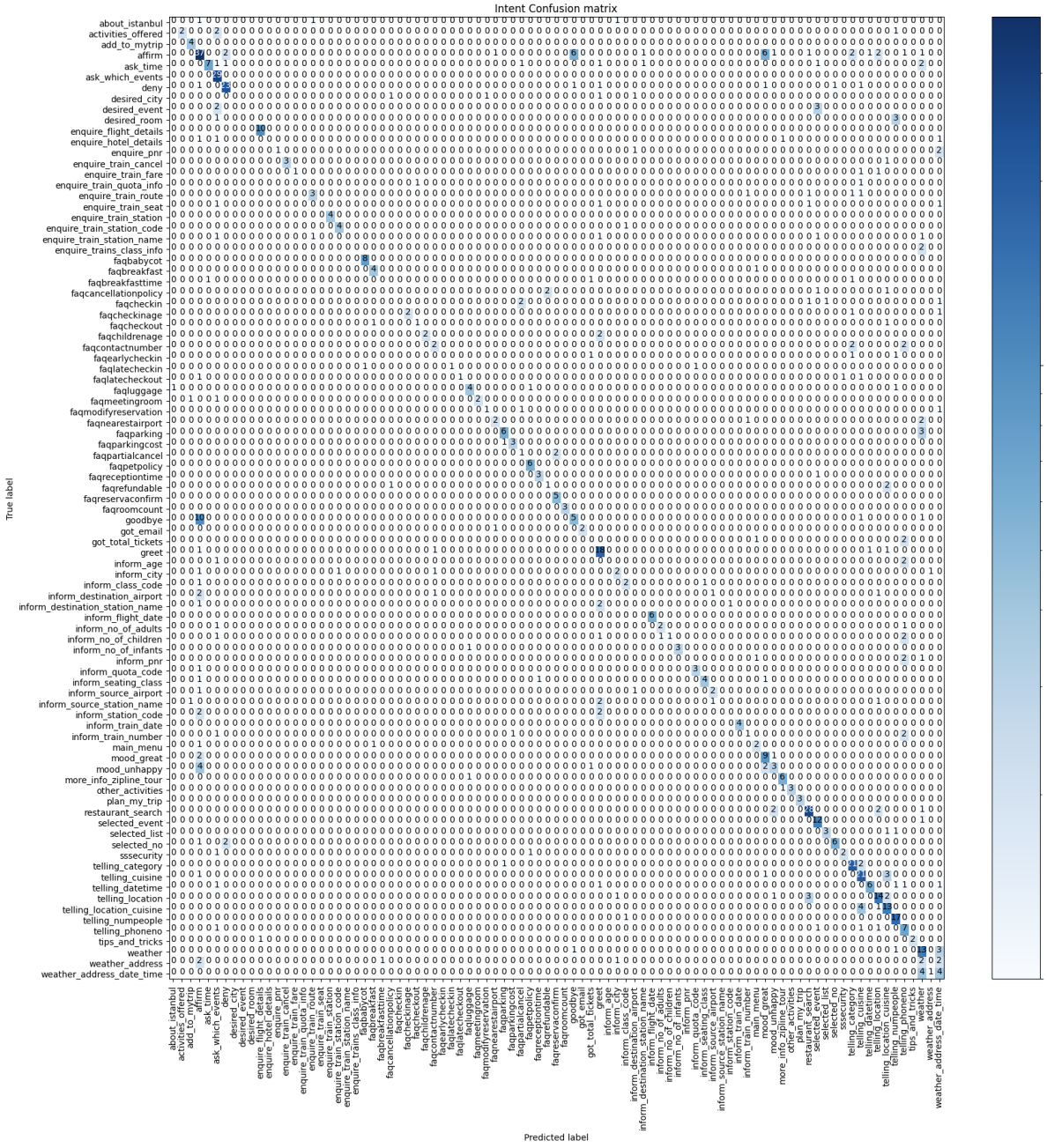


Figure 31. Intent confusion matrix of convert configuration

The confusion matrix obtained as a result of the intent classification is included in Figure 31. It is observed there are more mistakes in the intent matrix rather than the entity. Most mistakes were made by predicting goodbye intent as affirm intent.

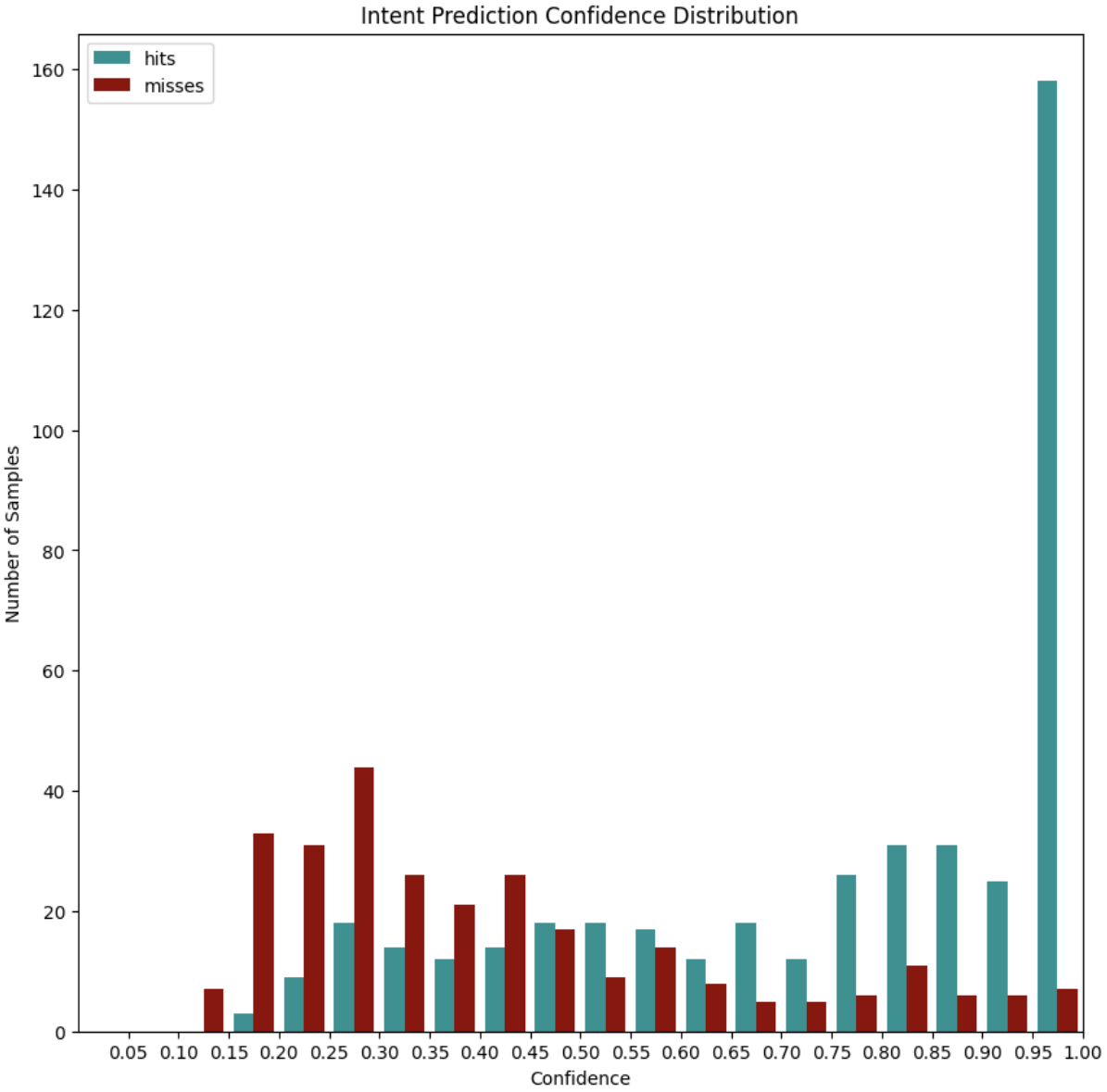


Figure 32. Confidence histogram for intent confusion matrix of convert config

Finally, we can examine the confidence histogram of intent classification in the convert configuration from figure 32. When we look at the histogram, in the samples at the beginning, the number of misses are more than hits. But later, hits made a big difference with misses in the samples that followed.

Config-heavy:

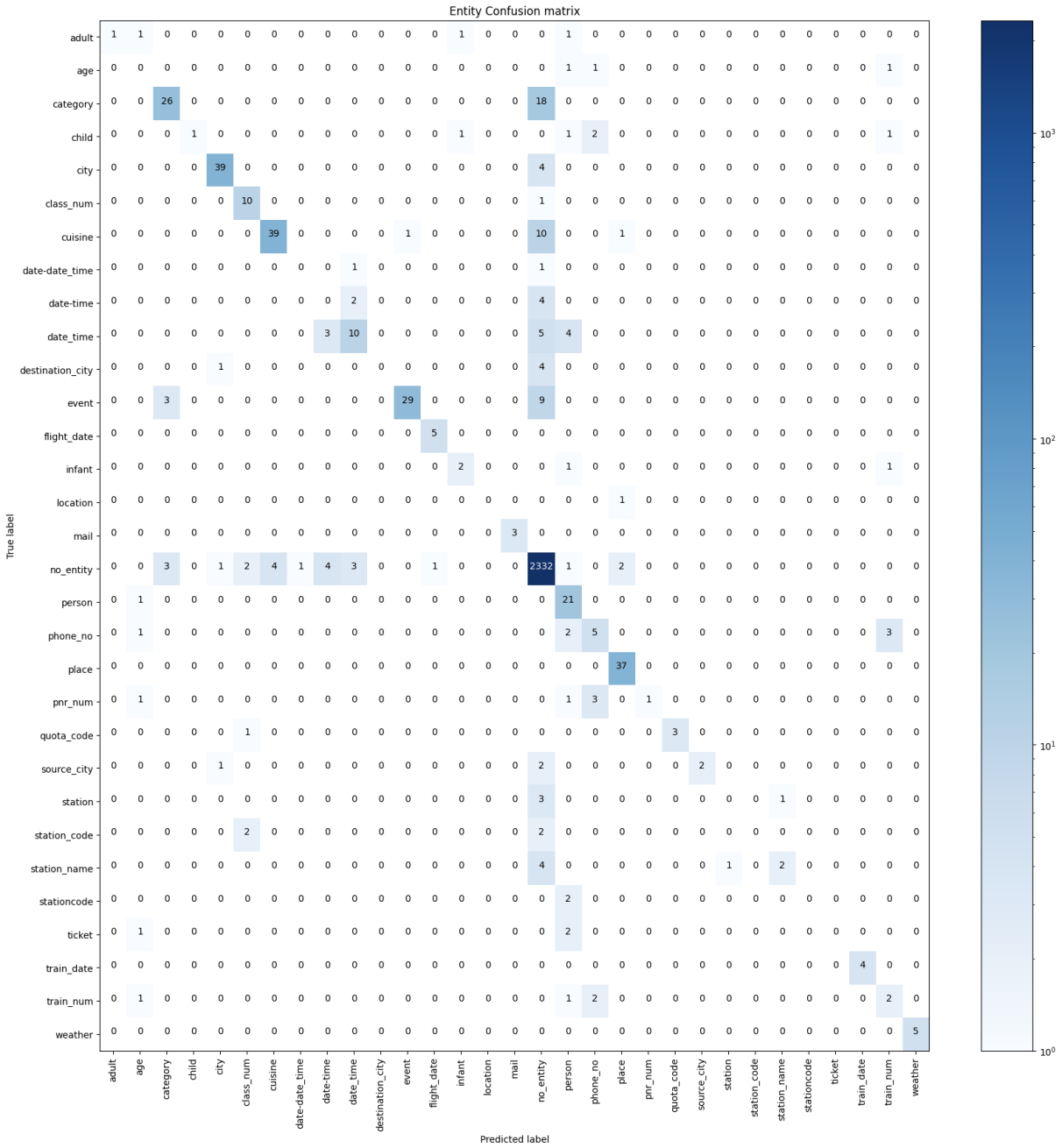


Figure 33. Entity confusion matrix of heavy configuration

Similarly, when we examine the entity confusion matrix of the heavy configuration, we see that errors have been made in the person entity. In addition, as we can see in Figure 33, the category entity is also a false estimate. Apart from these, most entities are predicted correctly.

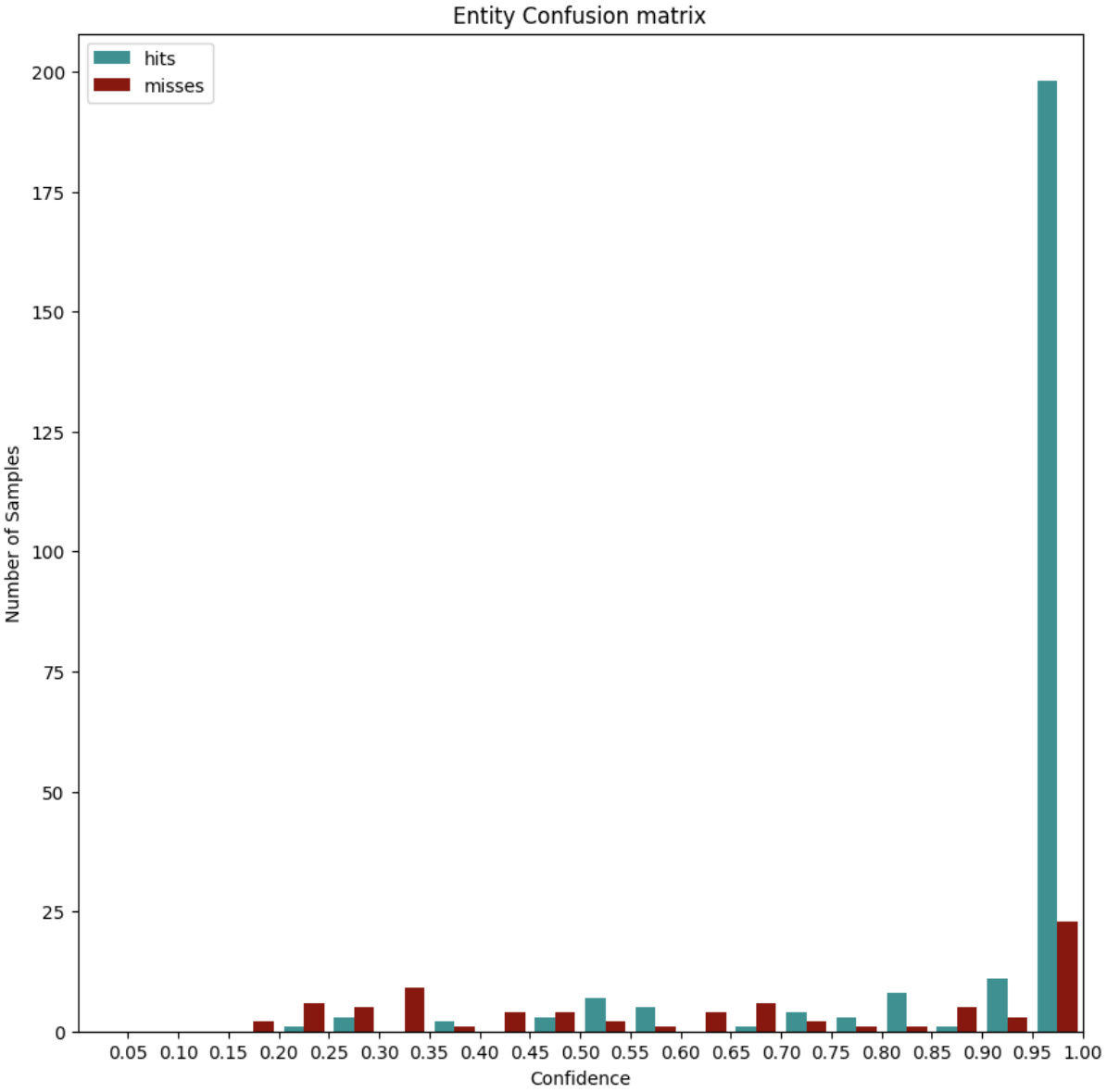


Figure 34. Confidence histogram for entity confusion matrix of heavy config

In the confidence histogram of the entities in Figure 34, it is observed that hits and misses values cannot be taken before a certain sample number is reached.mAs the number of samples increases, the hit and miss values remain low, but when it reaches 200, there is a serious increase in hits.

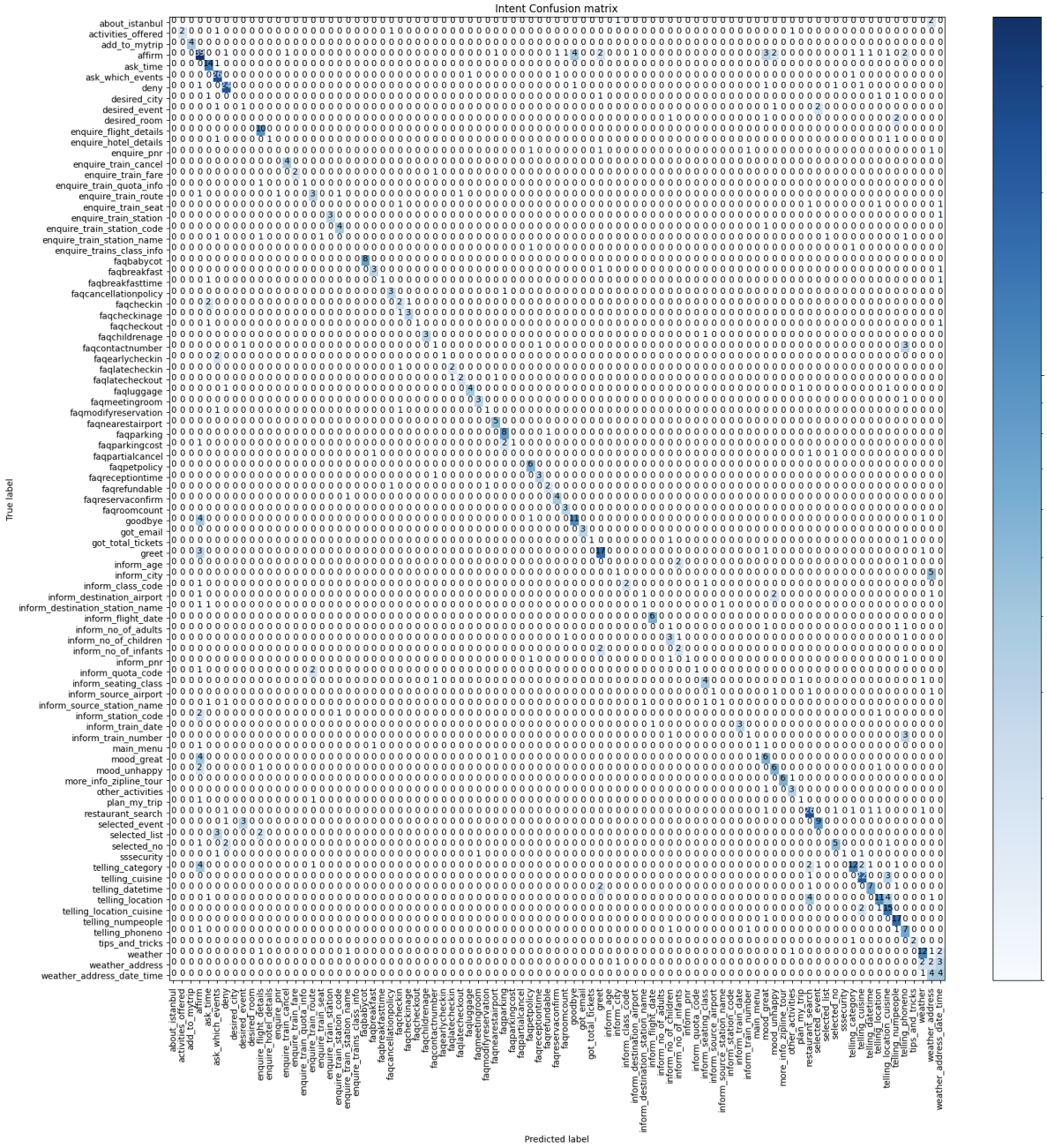


Figure 35. Intent confusion matrix of heavy configuration

When we look at the confusion matrix of the intent classification in Figure 35, we observe that unlike other configurations, it makes false predictions in intents such as restaurant_search, weather_address_date_time, as well as affirm. Apart from that, it can be said that it makes more wrong predictions as an intutive when we compare it with the results of other configurations.

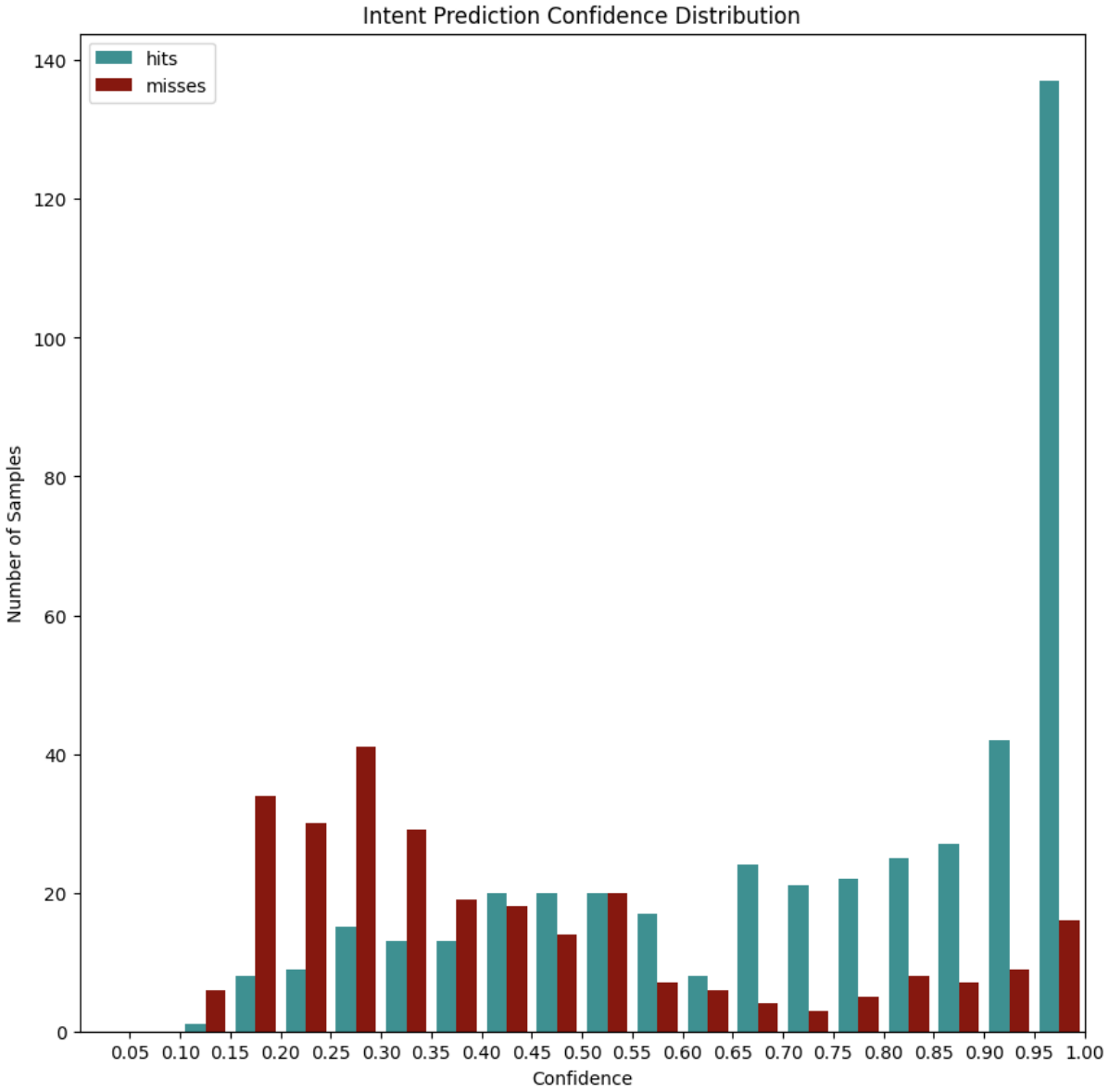


Figure 36. Confidence histogram for intent confusion matrix of heavy config

When we look at the hits and misses rates of the intents in Figure 36 through the histogram, it is observed that when the number of samples is less, the model makes more mistakes. At the same time, the confidence of hits started to increase after a point and reached its highest value.

6. RESULTS

As a result, we now have an application that works correctly. We are getting our expected answers from chatbot right. We can plan the trip with booking flight, hotel, and event as seen in Figure 37. According to this example, we booked an event. The chatbot also can make flight, hotel and restaurant reservations.

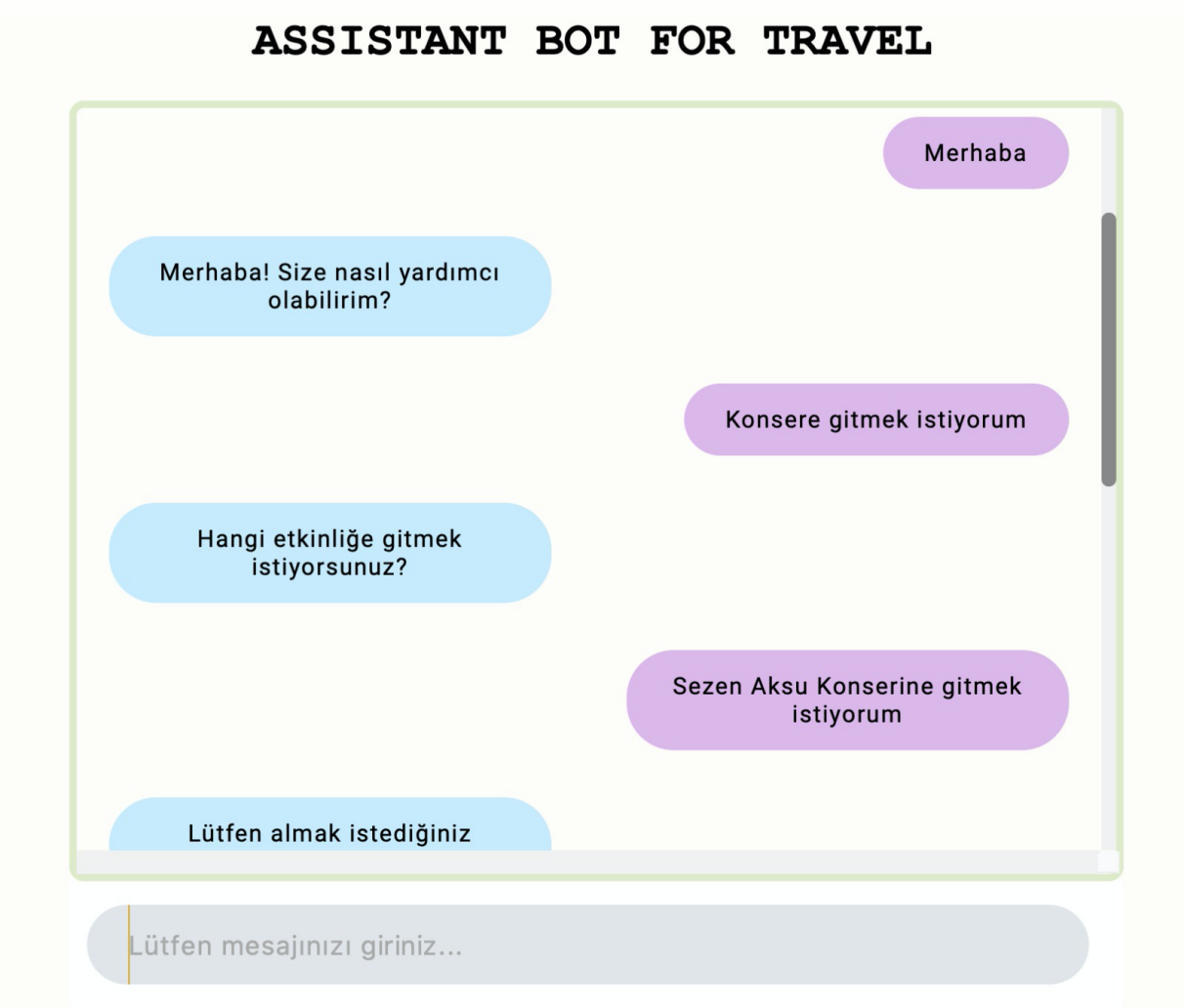


Figure 37. Sample conversation with chatbot

The chatbot understands that we are giving greetings and saying goodbye, as well as understanding and responding to whatever action we want to do. In order to do this, instead of generating lots of scenarios for every possibility, we created specific scenarios and added rules. The chatbot can fill the slots with the inputs it receives from the user during the

conversation, affecting the training in prediction and returning these slots in the response. More or fewer bookings can be made, regardless of the order of the conversation.

Figure 38. Sample conversation with chatbot part 2

In Figure 38 the chatbot receives the mail and ticket count. In Figure 39, the chatbot says goodbye and closes it.

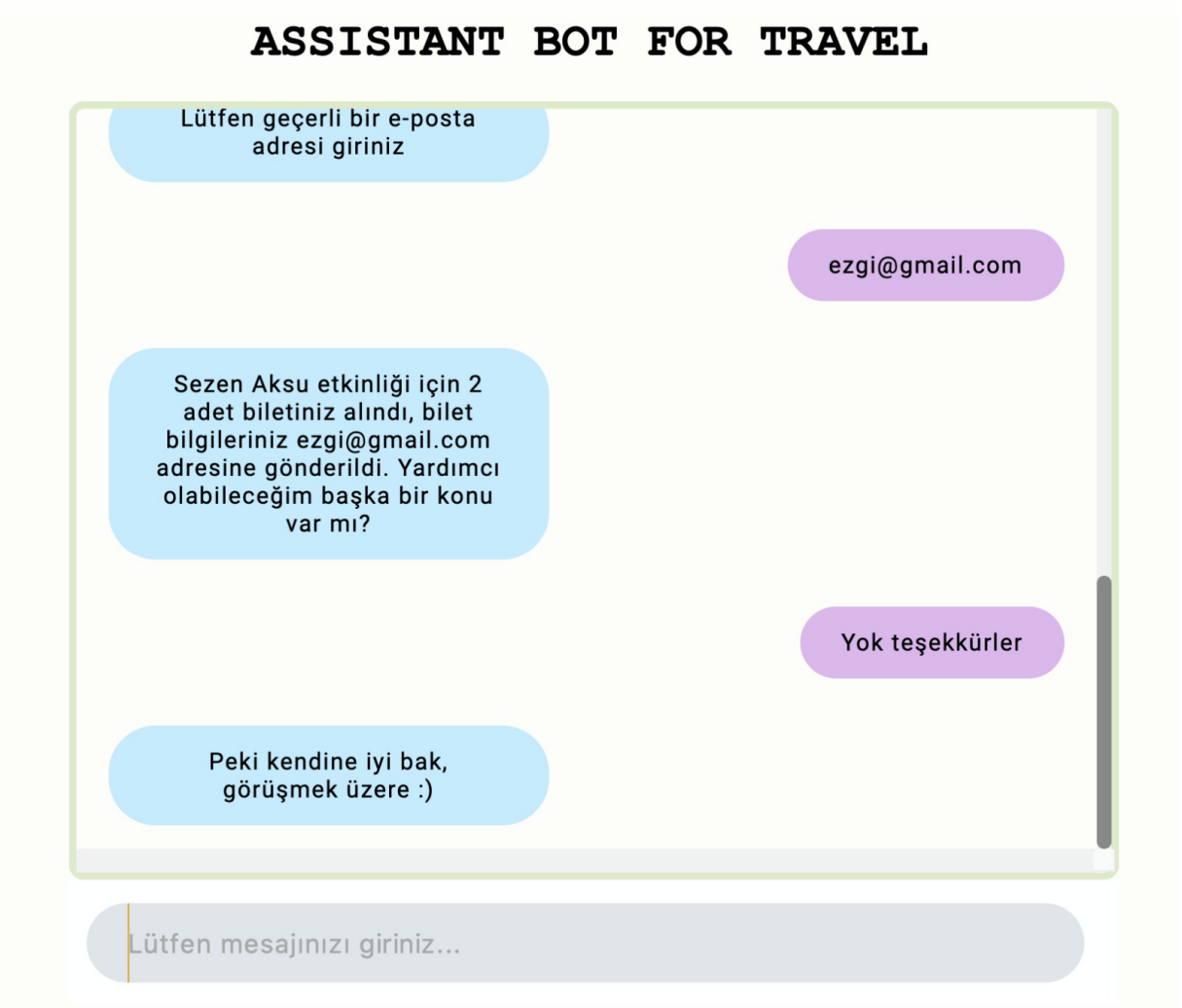


Figure 39. Sample conversation with chatbot part 3

7. CONCLUSION

We studied how to implement a chatbot which helps the user's travel. We created a rasa demo which can book an event, flight and hotel. To implement this project, we used Python, Rasa and Flask API as tools. After preparing the chatbot in Rasa, we enabled the user to talk to the bot through the user interface which we implemented using the Flask API.

7.1. Life-Long Learning

In order to do this project, we needed to learn Rasa. And for our further studies, we studied the different methods of Machine Learning. We examined some pre-trained ML models for NLU. For all these studies, we got help from our advisor and the resources on the internet.

7.2. Professional and Ethical Responsibilities of Engineers

We have always been careful not to do plagiarism. We preferred open source codes and took care to reference them.

7.3. Contemporary Issues

Our project topic was natural language understanding. NLU covers one of AI's narrower but particularly complex challenges. NLU is used by speech agents, including Alexa, Siri, and Google Assistant. NLU, "What is the weather forecast tomorrow?" It can digest the spoken text such as, and understand it a day later as a request for the forecast weather at the current location. With the development of smart assistants, it makes our daily life much easier. The ability to get things done without the need for manpower saves a lot of time.

7.4. Team Work

During our Senior Design Project, we worked together as two students. We always communicated with each other and tried to solve each problem that we faced. We always participated in the meetings we had with our advisor and we determined how we would proceed and what we would do until the next meeting together. After the meetings we held with our advisor, we did the tasks that needs to be done by sharing equally and fairly. When we had to do different tasks, we increased our knowledge and understanding of the subject by telling each other how we do the work we do. Thus, we both had detailed information about our project.

APPENDIX A

Your appendix goes here.

APPENDIX B

Your appendix goes here.

ACKNOWLEDGEMENTS

We cannot express enough gratitude to our Senior Design Project advisor Dr. Şeniz Demir for her continued support and encouragement. We offer our sincere appreciation for the learning opportunities provided by her.

REFERENCES

[1]A. Gupta, P. Zhang, G. Lalwani and M. Diab, "CASA-NLU: Context-Aware Self-Attentive Natural Language Understanding for Task-Oriented Chatbots", *Aclweb.org*, 2019. [Online]. Available: https://www.aclweb.org/anthology/D19-1127.pdf

[2]Z. Qiu, E. Cho, X. Ma and W. Campbell, "Graph-Based Semi-Supervised Learning for Natural Language Understanding", *Aclweb.org*, 2019. [Online]. Available: https://www.aclweb.org/anthology/D19-5318.pdf

[3]T. Bocklisch, J. Faulkner, N. Pawlowski and A. Nichol, "Rasa: Open Source Language Understanding and Dialogue Management", *Arxiv.org*, 2017. [Online]. Available: https://arxiv.org/pdf/1712.05181.pdf

[4]J. Shree, E. Liu, A. Gordon and J. Hobbs, "Deep Natural Language Understanding of News Text", *Proceedings of the First Workshop on Narrative Understanding*, pp. 19–27, 2019. Available: https://www.aclweb.org/anthology/W19-2403.pdf

[5]D. Braun, A. Mendez and F. Matthes, "Evaluating Natural Language Understanding Services for Conversational Question Answering Systems", *Proceedings of the SIGDIAL 2017 Conference*, pp. 174–185, 2017. Available: https://www.aclweb.org/anthology/W17-5522.pdf

[6]R. Bapat, P. Kucherbaev, A. Bozzon, "Effective Crowdsourced Generation of Training Data for Chatbots Natural Language Understanding", *Repository.tudelft.nl*, 2020. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid:93fdad6e-7d97-4441-b3ca-a1e9dde637d7

[7]J. Delvin, M. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *Aclweb.org*, 2020. [Online]. Available: https://www.aclweb.org/anthology/N19-1423.pdf

[8]Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov and Q. Le, "XLNet: Generalized Autoregressive Pretraining for Language Understanding", *Papers.nips.cc*, 2019. [Online]. Available: https://papers.nips.cc/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf [Accessed: 04-Dec- 2020]