University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

An Intent-Aware Chatbot for Cybersecurity Recommendation

Sebastian Sanchez
Zürich, Switzerland
Student ID: 17-732-710

Supervisor: Muriel Franco, Eder Scheid
Date of Submission: March 13, 2020

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
BinzmÃ¼hlestrasse 14, CH-8050 ZÃ¼rich, Switzerland
URL: http://www.csg.uzh.ch/

German Summary

Wir werden einen Chatbot mit Hilfe von Natural Language Understanding (NLU) erstellen, der es uns ermöglicht, Anfragen von Benutzern abzugleichen, die sich auf die Cybersicherheit konzentrieren, insbesondere im Bereich der Denial-of-Service-Angriffe. Der Benutzer wird eine Anfrage eingeben, die mit Hilfe eines Klassifikatiansalgorithmus durch maschinelles Lernen bearbeitet wird. Wir werden die eingegebene Anfrage mit einer Beispielanfrage vergleichen und die wichtigen Elemente daraus extrahieren, so dass der Benutzer sie zur Speisung eines Empfehlungssystems verwenden kann.

Wir werden 4 Elemente erstellen. Eine Benutzeroberfläche, die es uns ermöglicht, mit der Programmierschnittstelle (API) für maschinelles Lernen zu kommunizieren, die in Verbindung mit DialogFlow, einem NLU Framework, entwickelt wird. Darüber hinaus werden wir den Klassifikator trainieren und optimieren, so dass er Intents genannte Absichten einer Menge vordefinierter Intents zuordnet. Wir erstellen und verwenden als Grundlage eine Datenbank zur Identifizierung von Entitäten. Diese wird mit einem Proxy namens Fullfilment kommunizieren, der die Ergebnisse filtert; dies wäre unser drittes Element. Schliesslich müssen wir einen Evaluator entwerfen, der quantifizierbare Ergebnisse anhand einer Kostenfunktion liefert, mit Hilfe derer wir beurteilen wie nahe unsere Eingabe an der klassifizierten Ausgabe liegt. Wir führen auch eine Fallstudie zur optimale Nutzung durch, um zu überprüfen, in welchen Szenarien sich der CahtBot eignet.

Die Ergebnisse zeigen das es möglich ist einen Chatbot für dieses Anwendungsszenario zu entwickeln, der Elemente aus Abfragen in einer Konversationssituation extrahieren und auf der Grundlage dieser Informationen eine JSON-Datei generieren kann. Solche Informationen können in ein Empfehlungssystem integriert werden. Die Auswertung der Fallstudie ergab gute Ergebnisse, und die quantitative Analyse ergab ein durchschnittliches Konfidenzintervall für die Key-Intent-Klassifikatoren von 56%. Das bedeutet, dass bei einer zufälligen Eingabe des Benutzers die Wahrscheinlichkeit besteht, dass die Eingabe erfolgreich klassifiziert wird. Die Extraktion aus der Datenbank war erfolgreich, und der Web-Client wurde erfolgreich erstellt und eingesetzt.

Weitere Verbesserungen können erzielt werden durch direkte Verwendung der Eingaben des Benutzers im Klassifikationsmodell. Ausserdem kann die Klassifikation unter Verwendung verschiedener Techniken wie z.B. Long Short Term Memory(LSTM)-Pipelines anstelle von Support Vector Machine(SVM) und ersetzen Sie den Authentifizierungsmechanismus des Clients verfeinert werden. Eine Open-Source Alternative wie RASA kann auch evaluiert werden, um die NER mit Hilfe von Reinforcement Learning zu trainieren.

Abstract

Nowadays the cloud architecture is a growing trend that allow us to scale systems without having a physical architecture. This seems to be the future but it also presents several challenges for the current IT System Administrators since cyber attacks are on the growth and the need for performance and availability is also on the growth. In order to mitigate this problem we created a ChatBot using DialogFlow that gathers information from the System Administrator and maps the requirements using a pretrained Machine Learning model that use Natural Language Processing and Natural Language Understanding to identify and match possible requirements in a few layers of questions using the UNIX Man page collection as a database. The tool will also feature a client that will retrieve the data and process it using the described architecture to generate a data structure in JSON that other tools could use to generate a solution for the existing problem. Furthermore, we evaluated the solution using a loss function and a case study to check the impact and usability of the tool in real life.

Acknowledgments

First and Foremost to Muriel Franco my thesis supervisor who guided me throughout the process and provided guidelines and support to overcome issues. Also, I would like to thank the Communications System Group to allow take place in the research with the group and the support throughout the process and presentations. Eder Scheid who was always present in my milestones and who's recommendations helped me to improve in several areas of the research.

The Reviewers, Carolina Herrera and Ben White who helped me review my thesis and provided with recommendations to make it better.

Finally, Would like to thank Patrick Frank who's ideas helped me out throughout the process of research, the RASA community who helped me when I needed help with the code and my Mother Karola for the support given through the process and the rest of my Family who contributed.

Contents

Chapter 1

Introduction

Cybersecurity has proven to be a key player in the digital era and thus, an important matter to companies and governments that need to use or provide online services. The investments in cybersecurity have been arising along the last 10 years and is expected to exceed $124 billion USD in 2019 for contracting cloud-based security services [2]. However, there is no generic answer to the best way to defend against cyberattacks, since the protections offered can differ from several aspects (e.g., price, technology, performance, and supported cyberattacks). Thus, the process of decision for one protection solution or configurations is a hard task, mainly when there are no cybersecurity experts in a company. Based on that, recommender systems with simplified interfaces can help end-users to understand the most suitable security solution based on real demands. However, there is a lack of this kind of systems that might not be able only to identify the best solution for a network attack but also guide the end-user for possible configurations and tasks to be executed in order to prevent and mitigate a attack.

In order to take advantage of this system, a broad scope of possible solutions can be provided based on the existing state-of-the-art technologies such as Software-defined Networking (SDN), Network Functions Virtualization (NFV) and machine learning. Such technologies can be for example, a base for tools that aim to protect against different cyberattacks. Therefore, information from the underlying infrastructure, cyberattacks characteristics and end-users profiles can be used as input to help in a way that eases the task of planning and protection of a broad range of businesses. For that, different mechanisms can be explored to create a human-computer interface where information can be obtained directly from the end-user.

In such a direction, several studies have found ways to interact in a more natural way using chatbots [3] to map text to intents that might suggest the solution for a common or already experienced network problem. This is a guideline proposed as an accepted enhancer of the interaction between end-users and recommender systems to benefit different cybersecurity aspects. DialogFlow Chatbot, for example, can be used to map the needs of end-users using Natural Language Processing (NLP) to generate a structure that is capable of providing enough abstraction to be used by cybersecurity recommender systems like that introduced in [1].

1.1 Motivation

At the current state-of-the-art, there is a tendency to grow in the cloud-based services. As discussed in [4], the expected annual growth of the cloud computing services is about 27% annually starting from 2018. Thus the amount of the network operators needed would be defied from the market study tendency, which shows a growth of only 24% in computer science careers across the US [5]. This will undoubtedly culminate in an increasing tendency for network operators and sysadmins, which are educated to handle security issues such as cyberattacks, performance issues, or business availability. At the same time, there is a growing tendency of cyberattacks as described by Zahng[4], thus meaning that measures to guarantee security aspects must be maintained while taking into account the limited amount of people capable of maintaining the security of a system. Thus, there are opportunities to design solutions to help in the different aspects involved in the cybersecurity decision process.

Although such kind of solutions is still scarce, there are available tools that can be used to ease the process of eliciting the requirements from a system engineer and parse them into more natural constraints, such as Lex from Amazon or DialogFlow from Google. Such solutions use NLP to learn, identify, and process inputs from users to map into a rather simple set of actions. Thus, information can be mapped, for example, in order to ease the task of deciding the best way to defend against a cyberattack. In such a direction, we argue that there are opportunities to explore NLP as chatbots to provide simplified interfaces where operators can interact to define its requirements as intents, in a simplified way, to receive technical recommendations of how to prevent or mitigate cyberattacks, such as applying configurations or deploying protection services.

1.2 Description of Work

This thesis consists of the design and implementation of a chatbot capable of interacting with end-users using natural language and translates the end-users' intents to a data structure to be used as input for a protection recommender system. To reach this goal, as initial input, all information related to end-user's infrastructure, budget and cyberattack reports is considered. As output, the chatbot can provide insights regarding suitable solutions that satisfy the user's demands.

In order to achieve the thesis goals, technologies such as DialogFlow, in conjunction with a comprehensive man-page database [1], is used. Furthermore, this work provides a web-based interface for interaction between end-users and recommendation systems like [1]. The goal is to present a trained chatbot that is capable of suggesting a useful schema for a recommendation system based on entity analysis from the input of the end-user. It is important to mention that the chatbot proposed in this thesis will enable a client front-end to allow user interaction to ease the issue of defending and identifying cyberattacks.

1.3 Thesis Outline

The thesis will be described as an introductory step to background work and leading to a reference to related works, in order to understand how NLP works and how is this relevant for DialogFlow and Entity Analysis. The aim for this is to create enough background to understand the presented solution which will hold all the values recollected as following:

- **Background**: Here we will describe a few important areas about Natural Language Processing and Natural Language Understanding. The basis of how outliers work, what is a corpus and how each sentence gets vectorized and represented into a mathematical representation is described. Furthermore, we will present an intro into other NLU Chatbots that are relevant to the study and which one we decided to choose and why.
- **Related Work**: We will introduce the current work in the area such as RASA the main competitor of DialogFlow and how NLU is working in chatbots. The current state of cybersecurity solutions and the challenges presented in the Chatbot technologies.
- **Approach**: In this section we will talk about all the aspects regarding the implementation and the research that took place in order to create and define the intents and algorithms for the chatbot to take place. Areas ranging from Extraction of the information from the Manned DB [12] and reinsertion into Datastore are presented. Altogether with the Model and a description of how the entire solution was modeled for each component. Finally we end with a description of the intent mappings and training phrases for each set of the key intents in the DialogFlow Machine Learning algorithm.
- **Prototype and Implementation**: Moving on, this section will describe the solution that was designed to allow users to interact with a model and will alter on provide a basis for evaluation. This includes the used technologies and the process of creating the application on the front end side.
- **Evaluation and Discussion**: The evaluation and discussion will try to quantify the results from both the trained model and the client first by introducing a model that tests the classifier in a semi-automatic way so we can see the results in the way of graphs. Moreover, we also have a combined view of the solution with a case study that is presented top to bottom in an ideal case scenario. Finally, the discussion will take place that will combine the creation of the tool and the evaluation to see what are points that are worth improving or taking into account.
- **Conclusion and Future Work**: Here we will try to sum up the results from the research and give an idea of future areas where the chatbot could be improved or additional research could take place to improve the solution or solve other areas that remain inconclusive.

All these elements will be described using graphs and tables to expand the understanding of the content, furthermore the code and usage can be found in the Contents of CD and Installation Guidelines.

Chapter 2

Background

2.1 Natural Language Processing

In order to have an understanding of the current issues arising in the process of a chatbot several areas must be understood for the current work. For instance understanding the basics of natural language processing can give us a background on how the google algorithm of Dialogflow works.

The rational behind Natural Language Processing regards to a machine learning principle which relies on computing closeness between words and correlations with others to identify patterns and generate useful test sets. This area aims to divide language processing into three areas: bag of words, bag of concepts and bag of narratives. Each of this aspects aims to connect and find a pattern between words itself which is proven to be hard by Cambria [7]. On the other hand, combinations between bags of words and bag of concepts seem to infers better the context in which sentences can help to identify and predict the next sentence. Thus, NLP algorithms use a large set of data to combine word and context syntax to allow for an ease of linking between sentences. Finally, the bag of narratives aims to combine paragraphs in order to get connected contexts, this in principle is rather hard and usually done using statistical analysis to approach patterns among sets of sentences.

The best way to provide NLP algorithms that aim to extract not only word but also context and narrative are algorithms that simulate the human Psyche, thus deep-learning and Machine Learning algorithms that can aim to gather enough information to extract sentiments from the context and map sentences to actions and entities that allow to infers the context to provide a better word accuracy ratio as described by Cambria [7]. Using machine learning techniques to abstract and extract entities and context we can further the models and allow mapping and identification of elements that can be used and is used in transfer learning algorithms. This can be seen in DialogFlow as the tool's base maps entities, contexts and intents in order to identify sentences and break them down to identify elements and reply accordingly.

Based on the information extracted from Cambria [7] and the google white-paper [8] is clear that a large corpus of data or text can help to infer the context from in which it belongs and thus allowing a easier mapping of the narrative within the context allowing the extraction of entities to be perform in an accurate way. This could be combined with other methods of machine learning like reinforced learning to train a corpus of text and enrich it with interactions and allow an accurate mapping.

2.2 Chatbots

In order to understand how a chatbot work, it's important to mention that chatbots are in a constant increase and several chatbots pre-trained with big data-sets and undergoing active supervised learning are experiencing a significant growth. So is that we have the main technology companies investing a quantifiable amount of money into developing their products. This is easier understood by mapping each product to the company such as DialogFlow and Google, AmazonLex and Amazon or Luis and Microsoft. Regardless, this leaves behind the fact that some big competitors such as IBM with Watson are ahead but in practice the results seem to differ.

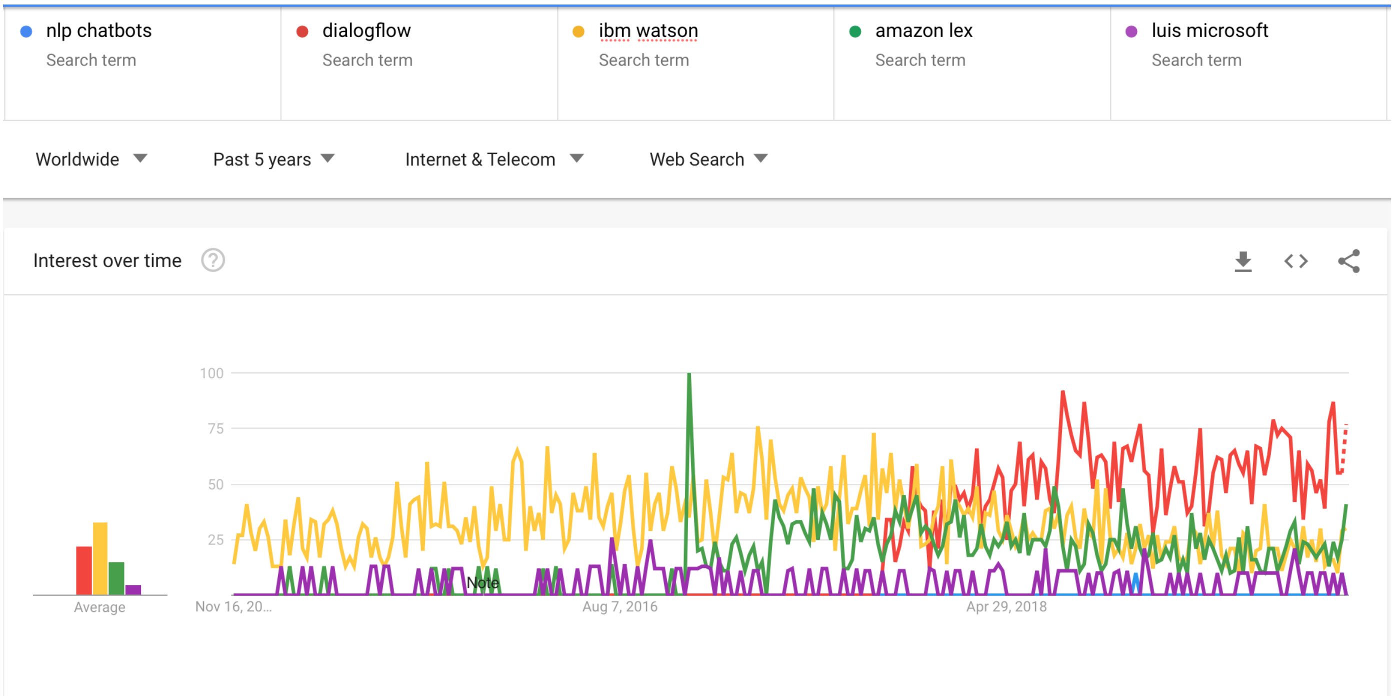

Figure 2.1: Comparison Of Google Trends for NLP Chatbots

This trend of growth among the different areas can be described in the figure 2.1 where we can see that there is an ongoing trend for DialogFlow over other chatbots despite the continuous big interest overtime in IBM's Watson.

- Non-programming chatbots
- Conversation-Oriented chatbots
- Platforms by tech giants' chatbots.

The breakdown of chatbots as described by Rahman[6] describes a schema were bots are offered in different layers which include the tree based chatbots limited to answer questions and simple structures without getting into detail of the content. Then on a different layer the Conversation-Oriented chatbots use data as a foundation of information and Natural Language processing for further inspections.

In the Conversation-Oriented chatbot layer we can find that most clients have issues with training layers of data. Thus the biggest players in this category are the ones who handle and create models based on big amounts of data combined with entity and sentiment analysis. Furthermore, this leads us to the Platforms by tech gigants as described above which include DialogFlow which features pretrained models in combination with Entity and sentiment analysis which leads to ease of implementation and great gather of responses as well as context analysis.

2.3 DialogFlow API

In order to understand how the chatbot and the mappings is done its important to understand how the DialogFlow API works and what are the elements a usual chatbot looks upon before being able to convert and identify sentences into understandable machine context. To start we must understand that each sentence to be taken by the chatbot must used in two contexts, the first one to create an environment and the second to map the elements to entities in order to provide content for the chatbot to work on. To achieve this, DialogFlow uses Intents and Entities as many other chatbots such as Rasa to extend the usability of the text received. After the chatbot receives the text it will try to get mapped by using the trained entities which are the words to be looked upon. For example in our case we would be looking for tool sets or problem definitions to map to Unix tooling. Thus, our entities would be all the tools from the Unix universe and the intent will try to communicate the issue within this context. Additional entities can also be added and the more entities the richer the model will be. Mapping back to the example previously defined, we can say that within one intent, many entities can be mapped. To get ourselves in context we can say that the work of a text is broken into this area, despite that this will be further described in the development section to explicitly highlight the work of splitting and obtaining information from a string using a base test set.

Another important area from the DialogFlow API is the REST backend provided by google which allows us to talk back and forth with the google API to obtain the answers and also this will allow us to have structured responses since the back-end can handle JSON files. Also, the API features fulfilment which can allow us to reply using web hooks within a server app or even just to store and process a new answer using this API. It's worth mentioning that all the text will pass through the google API server. Moreover, there are additional features that will be explained during the development section that include how sentiment and entity extraction can be done in conjunction of the chat bot to further enhance the capabilities of the system.

Finally, among the capabilities of the DialogFlow API we have composite entities and context which can aid the work of identifying subsections of entities and furthermore using the context to track a conversation within a set of responses using intents. This will be detailed in the development but is worth mentioning since most of the API is build upon this topics.

2.4 Natural Language Understanding (NLU)

Natural Language Understanding is an underlying layer of Natural Language processing but that in contrast tries to get close to the actual meaning of a corpus rather than the abstract representation of it. This means that we do not look after the mathematical representation of the corpus to find for example words that might relate or that are similar using a nearest neighbor search. In the case of Natural Language Understanding the corpus gets spliced and fit into categories where we can extract the main elements from, this is more or a classifying problem and the rationale behind it is to allow to compute the real meaning behind a corpus rather than just use or predict the upcoming text, to have a clear idea of this we have the following example:

- "The House is not Blue"
- "The House has Many Colors but Blue"
- "A House very colorful with the exception of Blue"

In this particular case we can see that in common human language all the three sentences mean the same thing, clearly the House is not Blue. Whatsoever, in natural language processing we would combine the words to create a corpus and based on this a mathematical representation that would yield three different results due to the word not present in the first sentence but not in any of the others and thus yielding an erroneous representation since the three sentences are very much similar. If we take this example into a NLU classifier we would look for particular elements within the sentence that would describe a core within the sentence for example we would define classifiers that would look for a subject such as a House or a Car and an adjective such as Blue, Green or Black. The classifier will ignore elements from common language and will only look for the key elements that we are looking for.

Moreover, NLU allows us to handle and control contexts using either LSTM(Long Short Term Memory) or SVM (Support Vector Machines) to pass the extracted information and perform something close to a memory of the extracted information into the next classifier, this is quite relevant for our research since we need to feature conversations and these conversations could lack elements that were provided before.

NLU as described by Zhong [28] has two key elements in short-long term memory and recurrent neural networks. Using peer elements obtained from a conversation can be done by reducing the common language phrases such a bi-LSTM model that takes only the matching argument in a neural network as quoted by Zhai [27]. The extraction of entities from intents using LSTM and recurrent neural networks is a key element of NLU and thus must be taken into account.

We will describe in detail which aspects from NLU we will be using to train our algorithm.

Chapter 3

Related Work

There are several related work that train to aim to recreate or at least up to certain extent achieve a similar solution as the one proposed. In order to understand one of the closest paper in terms of content and approach we can have a look at Jacob's [3] paper on Refining Network Intents for Self-Driving Networks which aims to use natural language to perform entity extraction and thus allow to create intents based on the extracted entities to perform certain operations. This is quite related to our work since it used DialogFlow as a basis as well and extended its usage with an intent mapper in order to allow users which have a median or low understanding of computer networking to use and create intents based on a relatively simple pipeline which allows to create entity extraction as a service.

The process described by Jacob uses google assistant to extract entities from the utterances and pre processes them into Nile. Nile is a intent translator that aims to try to gather entity information and map this to actual intents that can be performed by and android system. Once nile has completed the task this is later on passed on to the Intent Deployer who must deploy the service itself and finally deploy a network policy based on the previous intents. Throughout the process several important areas are described, for example the machine learning model which contains Nile as a decoder. Within this area, the encoder tries to identify the input entities which are then transferred through the Thought vector to the Decoder. The Decoder will aim to create an intent based on the extracted entities and output this to the intent creator.

The intent creator is out of the scope of out work, but is worth to be mentioned as it will gather the extracted entities into a new intent that will deploy policies. This can be done using the recommending system developed by Franco [1]. Related to this work we can find other interesting mappers for NLP such as Cocoon which uses first-order logic instead of machine learning for creating the intents into configurations. Moreover Cocoon lacks the capability of a operator refinement and learning the operator intent over time which leaves it behind.

In order to understand mapping of natural languages is good to have a look at Lumi, as described by Jacobs in Deploying Natural Language Intents with Lumi in [3]. In this paper we have a glimpse into information extraction, intent assembly and deployment using a tool called Lumi which is in charge of handling all the resources and routing to the right scenarios. In this area there are several other aspects that must be taken into account for our implementation such as the intention aware detection and the entity extraction that also takes place in this tool.

3.1 Rasa and NLU

An important tool to take into account before understanding why we used DialogFlow as our preferred chatbot to take into implementation instead of Rasa an opensource ChatBot that with an open source foundation. Since Rasa features NLU Natural Language Understanding which For Chatbots look into 2 elements and whose work is to look into entity extraction which allows to map further resources into either an intent description or a further entity extraction. This elements are proven by Braun [9] and the extraction of this element is also shown to be the most efficient with LUIS [9] as it shows that the corpus of the entry text usually influences on the capability of the ML algorithm to fetch and extract successfully the entity from the corpus. Despite this, NLU focuses on mapping the entities by default on chatbots but lacks the capability of mapping certain contexts and invalidate ambiguity within the context. For example, after the corpus is delivered then a further retrieval of another intent or a successful implementation must precede and thus query a common knowledge database in order to generate a message that could hold the main elements named:

- Text Planning
- Sentence Planning
- Linguistic Realizer

This elements will allow to map the original query of the user, combined with the entity extraction to a generated response that could be capable of delivering further questions to gather enough entities to satisfy the context or end the query and reply with the current information. Furthermore, neither RASA nor LUIS proved to handle the entity extraction in a very efficient manner which denotes that more work is needed in the area to correctly address the issue of entity extraction and thus allowing the chatbot to perform a complete analysis.

3.2 Chatbot Technology Challenges

In a different area we are also presented some interesting aspects of how chatbots are still limited not only by context but also by the corpus in which they work and show that there is work related to context identification but clearly addresses issues with the p-values required to accept the proposed solution by Rahman [10] in which three different aspects are presented that must be taken into account to understand limitations of the

Provider	Service	Pricing (USD)	Deployment
Cloudfare	Advanced DDos Attack Protection	Free Trial/ Quote	Cloud Based
Imperva	Incapsula	3.5k per Month, 5k Setup	Cloud Based
Arbor Networks	Arbor Cloud	on Request	Cloud/ On Premise
Verisign	DDos Protection Service	on Request	Cloud Based
Level 3	Communications & DDoS Mitigation	on Request	Cloud/ On Premise
Corero	SmartWall TDS	2.5k per month, 1k setup	Cloud/ On Premise
Flowmon	DDoS Defender	Free Trial/ Quote	Cloud Based
Akamai	Kona Site Defender	6k per month, 3k setup	Cloud Based

Table 3.1: Services And Pricing From Sula [11]

current state of NLU and chatbot development that might cap our project or set a logical boundary. Rahman describes that the current challenges which still hold now a days are related to how we extract information for each query.

They propose a theory which is used in most chatbots now a days, thus we use intents to map input to contexts and once this intent is recognized we can performs entity extraction to gather the important elements from the conversation leaving aside the facts that are not crucial for the implementation. In this way, the paper introduces B-Point Tree which outperforms the BST tree used in NLP Classifiers as supervised training which is the common technique used in chatbots. With this approach it is easier to infer the context and map the entities to the intents since you may have many intents to a single context.

3.3 Cybersecurity Solutions

Sula [11] gathered information about protection services for a subset of types of attack and generated recommendations based on this. This relates to our work as it might contain several pieces of information that help us to extend our service past the gathering of requirements and delivery of information. In his work Sula gathered a table containing a set of services and a relative reference price range which could be used to map possible entity extraction outcomes to possible solution tools. The table from Sula [11] gathered the following elements:

Here we can see a set of element that could be used to expand and use in our chatbot and can still can cause a beneficial impact on the system administrator experience. It is very important to mark that from this table not all elements are described in detail and not all solutions can be used to be mapped to all problems and thus there are underlying limitations.

Also, as it will be described later on many other services that also provide a comprehensive support system are available on the market and could be used to expand the usage of our chatbot.

Chapter 4

Approach

4.1 Extraction

During the process of building the chatbot several steps had to be taken into account in order to generate stable data that could be used to expand and integrate with our desired chatbot. For this we decided to extract the elements from the Man Pages from Linux in order to be able to generate a large amount of entities that could help us map each entity to a type of natural language element that will provide enough information to correlate the intent from natural language and the expected entities. To process this information we used a database that contained all the information regarding UNIX man-pages. The database we used for this is called Manned and was obtained from [12] in order to extract elements from this database. The original schema from the database can be seen in Figure 4.1 where it can be seen how a single index exists and for each element there are two partial sections which are the content or section for a Manpage and identifiers that can be queried to distinct if the element belongs or not to certain package. To expand this we can say that the database contains not only the recent elements but element from non official releases. In order to distinct and classify the information we must have classified and make some distinctions to be explained.

The database of Manned DB contains 18,926,291 Manpages extracted from several organizations, this database weighted around 21 Gigabytes of data on a PostGreSQL Database and had the schema presented on Figure 4.1. This recopilation of information includes information from a few elements, to name them:

- Packages
- Release Date
- Category
- Language
- Contains Section

- Architecture
- Section Arrangement

Based on this we filtered the important information that could be used to train our chatbot.

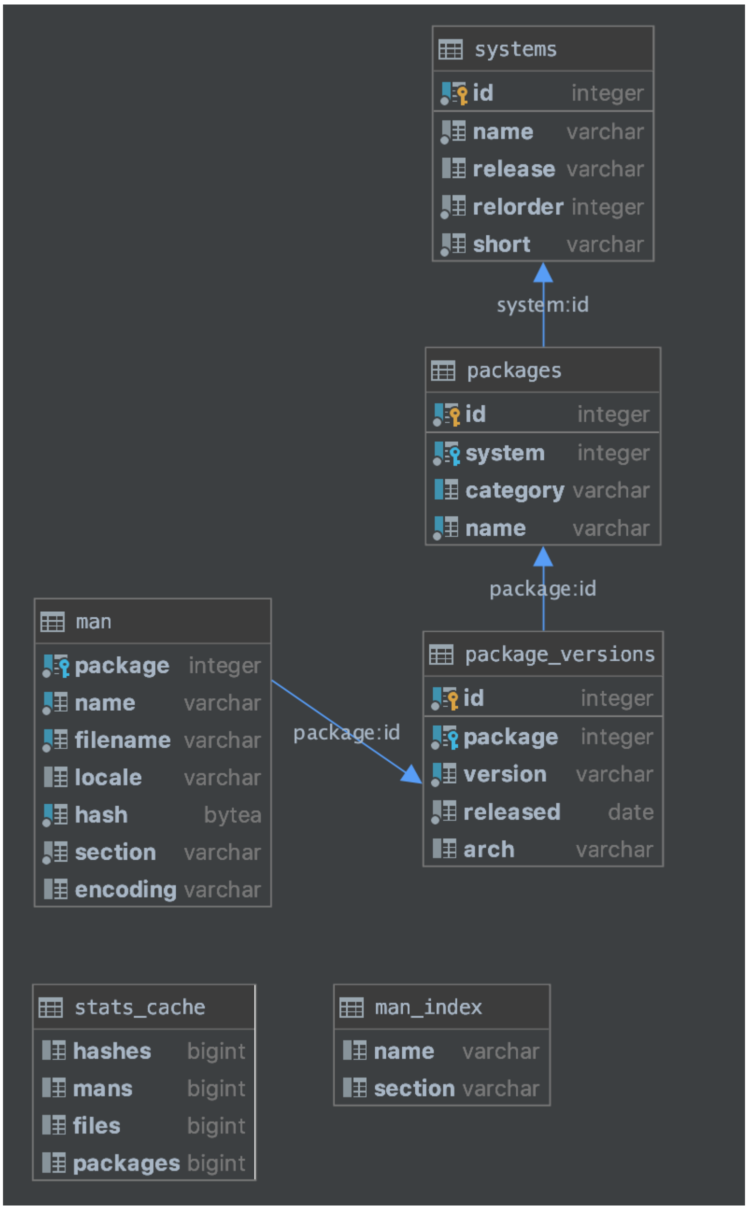

Figure 4.1: Database Schema From Manned DB

Package: To extract the right packages, the aim was to use the latest package from the current community edition. Meaning that for each Manpage it must belong to a certain package such as Development or Community and so on and so forth, expanding this each package must be versioned with a single number. In order to extract the latest information in the most stable way we used the highest version number from the community edition.

Release Date: In order to obtain the most relevant information and avoid indexing information that could not be used only the latest version of a manpage was used. In fact, older versions could also be used and might have expanded the accuracy of the machine learning training phase but due to time constraints was not used. This will be further expanded on the Further Works section.

Category: The categories can change from packages to packages and thus we decided to use the standard UNIX Manpage as described in the Ubuntu repo, other packages contain a different set of categories and thus we decided to use the standard leaving behind the POSIX and d,e,f sections to follow a more standardized approach.

Language: Despite the fact that DialogFlow can work with many languages and certainly most locales would've been beneficial for the project, to have a simple approach the proof of concept was developed aiming only for the English Language and thus, the other Manpages that were not in English were discarded.

Contained Sections: Is also important to mention that some packages do not fill the standard criteria for a Manpage section. Most Manpages follow a standard meaning that the must include:

- name
- short name
- synopsis
- description

If the manpage did not include this parameters or they were not in the standard way then they were also discarded as mining the data would've been a difficult task that was not part of the initial hypothesis. What so ever, if we included them the algorithm should've been able to filter them as well.

Architecture: Most servers run on a x86_64 architecture also known as the AMD64 Architecture, thus we decided that filtering all manpages that were not built for this set of instructions would ease the job of processing the entities to the right parameters. Some tools that work only in the x86 architecture were also included as they haven't been ported fully to a x64 architecture but are still relevant for usage. So is the case of tools such as i386 tools such as WineHQ.

4.2 Proposed Structure and Data Extraction

Furthermore, in order to be able to work with this information we had to filter all the previously shown information using a rather simple criteria. To show how this criteria was generated a simple snippet of code is presented.

```
SELECT s AS sys, p AS pkg, v AS ver, m AS man
        FROM man m
        JOIN package_versions v ON v.id = m.package
        JOIN packages p ON p.id = v.package
        JOIN systems s ON s.id = p.system
        !W
    ), f_english AS(
      SELECT * FROM unfiltered
      WHERE NOT EXISTS
        (SELECT 1 FROM unfiltered
        WHERE is_english_locale((man).locale))
        OR is_english_locale((man).locale)
    ), f_pkgver AS(
      SELECT * FROM f_english a
      WHERE NOT EXISTS
        (SELECT 1 FROM f_english b
        WHERE (a.ver).package = (b.ver).package
        AND (a.ver).released < (b.ver).released)
    ), f_stdloc AS(
      SELECT * FROM f_pkgver WHERE NOT EXISTS
      (SELECT 1 FROM f_pkgver
      WHERE is_standard_man_location((man).filename))
      OR is_standard_man_location((man).filename)
    ), f_secmatch AS(
      SELECT * FROM f_stdloc
      WHERE NOT EXISTS
      (SELECT 1 FROM f_stdloc
      WHERE (man).section = ?)
      OR (man).section = ?
    ), f_arch AS(
      SELECT * FROM f_secmatch
      WHERE NOT EXISTS
      (SELECT 1 FROM f_secmatch WHERE (sys).id = 1)
      OR (sys).id = 1
    ), f_sysrel AS(
      SELECT * FROM f_arch a
      WHERE NOT EXISTS
      (SELECT 1 FROM f_arch b
      WHERE (a.sys).name = (b.sys).name
      AND (a.sys).relorder < (b.sys).relorder)
    ), f_secorder AS(
      SELECT * FROM f_sysrel a
      WHERE NOT EXISTS
      (SELECT 1 FROM f_sysrel b
      WHERE (a.man).section > (b.man).section)
    ), f_pkgdate AS(
      SELECT * FROM f_secorder a
      WHERE NOT EXISTS
      (SELECT 1 FROM f_secorder b
```

```
      WHERE (a.ver).released < (b.ver).released)
    )
    SELECT (pkg).system, (pkg).category, (pkg).name AS package
    , (ver).version, (ver).released, (ver).id AS verid,
           (man).name, (man).section, (man).filename
           , (man).locale, encode((man).hash, 'hex') AS hash
     FROM f_pkgdate ORDER BY (man).hash LIMIT 1
```

Following the code presented above, the SQL query will extract the elements based on the criteria described in the previous subsections. This outcome was mapped within the same database before getting exported into the cloud. Figure 4.2 contains a diagram of the expanded database with the filtered elements into a new table.

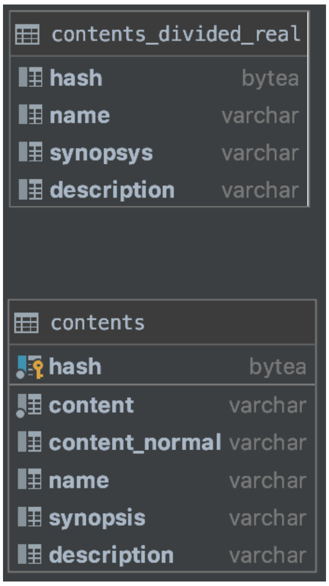

Figure 4.2: New Database Tables From Filtered Data

They all contain 100221 entries that could be mapped into entities, in order to do this the tables where exported into a NoSQL database that could interact with DialogFlow. In order to do this, we used Data Store by google which provides object storage within the cloud an allows dialog flow to extract and insert into entities as shown in figure 4.3. To have a better idea of how all the components work in Dialogflow a graph is provided to expand and explain how the interaction was made faster for a big amount of data as described above.

The NoSQL Database contains all elements sorted by keys which are then later on exported to the entity in DialogFlow using internal API calls. This was done by using a python script that took the relevant columns, parsed them and inserted them into the DialogFlow underlying layer. This script is provided in the Appendix and the logic works by processing chunks of information gathered from the database and converting them into data-frames that will be reinserted into the database while getting rid of the ROFF format [13].

The ROFF format is the standard for man pages but was extracted and removed using pre-processing headers within the python script. This allowed us to handle the information

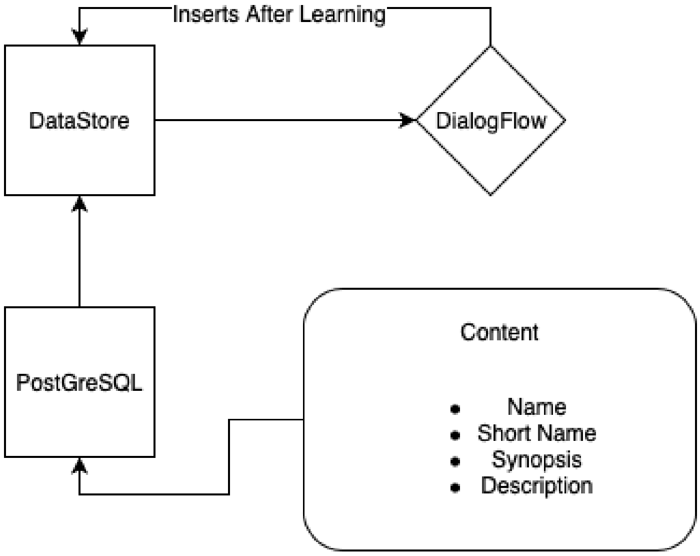

Figure 4.3: Data Extraction And Insertion Flow

in a rather easy way so DialogFlow can consume it and use it. Despite this, the algorithm should be capable of handling or discarding files that are not formatted using the ROFF standard and thus discard the entities that are not matching the right setting.

4.3 The DialogFlow Model

In order to understand how the model is developed we must have a relatively clear understanding of how is that the underlying NLU model works under the hood. Even though this was explained in the Background Section is important to keep into account concepts such as Entities, Intents, Context and Composite Entities all these elements refer to Domain Classifiers, Intent Classifiers and Named Entity Recognizers as described by Su [14]. In order to have clear picture of how each of this elements work together in the project is relevant to understand each element as separate. Thus, the underlying section of each one will be explain all together with a working description of the implementation of the intents in production.

4.4 Intent

The intents in DialogFlow are described also as Intent Classifiers (IC) within the NLU contexts and they recall the capability to identify certain mapped query based on a similar input query by filtering the elements existing in natural language and matching entities to the the ones from Entity Recognizers, this will be explained in detail in the Entity section. For starters there are two subsections of the Intent Classifiers as defined by Su [14] in which we can argue between Supervised Learning and Pretrained Embeddings.

On the first hand, we can describe Supervised Learning which is what RASA is using we have to understand that the model works by classifier a query based on a model that is trained already. It works by training word embeddings and counts how often distinct words can be used as an input for the classifier. This is denoted in the paper by Wu [15] where embeddings are trained from scratch using the n gram count algorithm. By the time the algorithm is trained we must try to use the input or original user query to match any of the relevant existing intents and thus identifying the original intent.

On the other hand, we have pretrained embeddings which is what DialogFlow uses to map or classify intents accordingly. In order to do this, each word is represented as an embedding and thus converted to a numeric vector. Therefore, we can find semantic and syntactic aspect of words in relation to other words that have a similar vector this is widely described in the word2vec paper from Mikolov [16] but the important findings regarding this paper rely within the fact that word embeddings can be used to relate words among each other using the following formula:

Bag of Words

$$Q = N \times D + D \times log_2(V). \tag{4.1}$$

Skip-gram model

$$Q = C \times (D + D \times log_2(V)), \tag{4.2}$$

As denoted by Mikolov where N is the previous words and V is the size of the vocabulary. Finally D is the corpus of the text. Also C is the calculated distance between the words. [16]

Taking this into account, we can say that the process of intent recognition used by DialogFlow reassembles the one used by RASA and calculates the average from the input of the user to calculate distances using fastText embeddings and the desired model in our case grid search.

Using the figure 4.4 we can reference that for our case using a pretrained embedding was the only option since we do not need to support multiple intents within a message and even though we do have specific terms for the domain the limitation in this case is the amount of training data available. Is possible that as the tool database grows and more intents are recorded and added as a future work a supervised embedding can be trained but this will be explained in the Further Works section. As of the models being used from Mikolov the CBOW referrers to mapping words to a word based on the context so based on the user input we can determine if the input relates to an intent. Furthermore, we also have the Skip-gram model that works the other way around, mapping from a word a context; this is quite relevant as this will be used to gather and extract the right context from the input but this will be explained in the context section. It is also worth mentioning that our chatbot contains a set of intents aimed towards DDoS attacks since the proof of concept is targeted towards this concept. Keeping this in mind will allow us to understand that the intent base can grow to match intents from the Tooling extracted in the first steps but at the current state just a handful of intents for tooling identification and analysis where left.

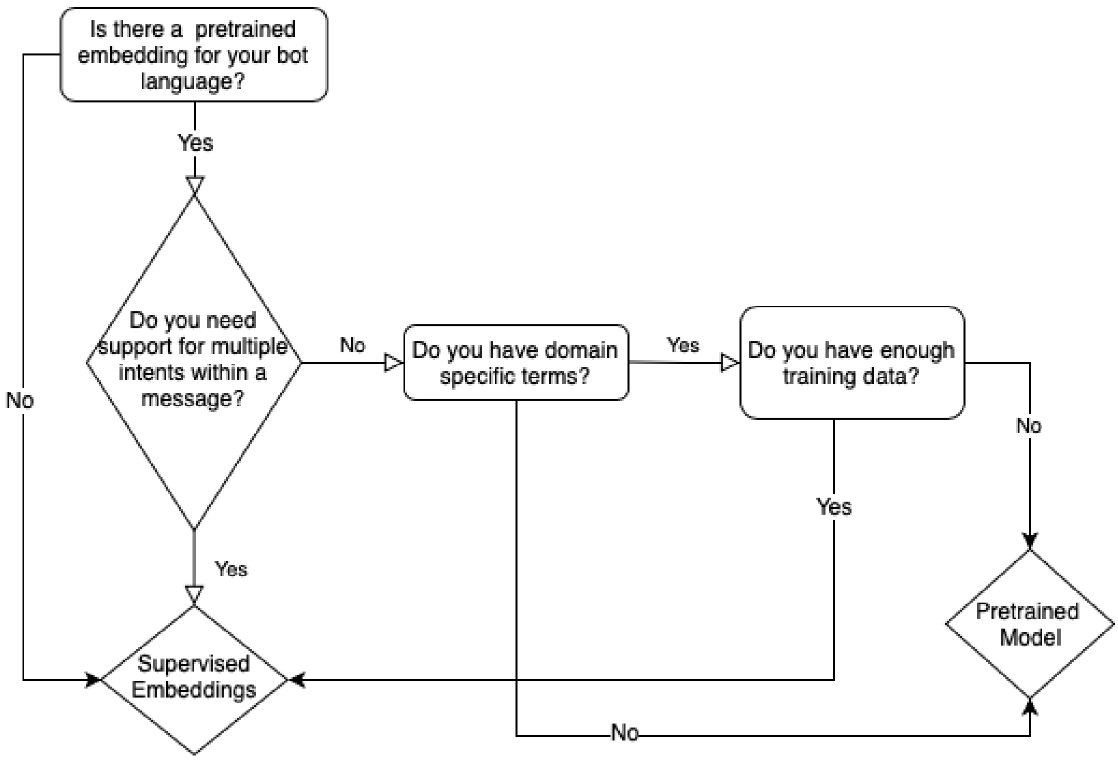

Figure 4.4: Diagram Of Selection For Pretrained Embedding vs Supervised Training

4.5 Entity

In each element of a sentence we must identify intents that will allow us to map the important elements within a sentence. Several examples can be provided such as "How is the weather today" or "Today we have good weather?" this two examples show how intents interact to provide the same set of information. For this case, the entities or important elements for further processing are Today as a time entity and weather as the requirement entity. Furthermore if we can identify this as a question then we would have successfully mapped the intent to the right set and thus would be allowed to expand this information to retrieve it from a third party service and provide an appropriate answer.

In the case of our chatbot we would need to extract relevant information that would allow us to generate a file that could be useful for a recomender system to give solutions to current problems. Thus, we used elements from the extracted UNIX Manpages to allow the ease of entity analysis and identification. To expand on this, the entities that exist or are added from the DialogFlow side are the ones obtained from the UNIX man pages and using the skip-gram algorithm on a pretrained model we aim to obtain embeddings that can map to the right intents. Also, this relies on the context that will be explained later on but on a first glimpse this is how the mapping between Entity and Intent happens in technical terms. Furthermore, we must expand on what an Entity is and how Entity Recognition works.

To understand entity extractors we must understand that there is not a single way to identify an entity but rather a set of choices. It is not known how Google handles the entity extraction but the scholar papers that recall entity extraction relate two main known ways which are:

- Pretrained entity extractors
- Rule-based entity extraction

For each one of this there are specific tools developed and run different algorithms to support the extraction. spaCy uses a already defined set of pretrained entities which are generally known, its most likely to be used in DialogFlow as well and is the default method of identification in tools such as RASA. It works by using a pretrained model with a list of entities mapped to results and uses the same for entity identification. For our case this is relatively simple as we provide a list of entities that are meant to be identified. Thus, we can say that our method of entity identification reassembles the one of a pretrained model with transfer learning on top of this.

Also, Duckling allow us to create a set of rules to identify an entity similar to a regex expression but generally this was proven to under-perform spaCy. If we aim to have a very comprehensive entity extraction and identification we can combine all the existing methods to provide the best solution.

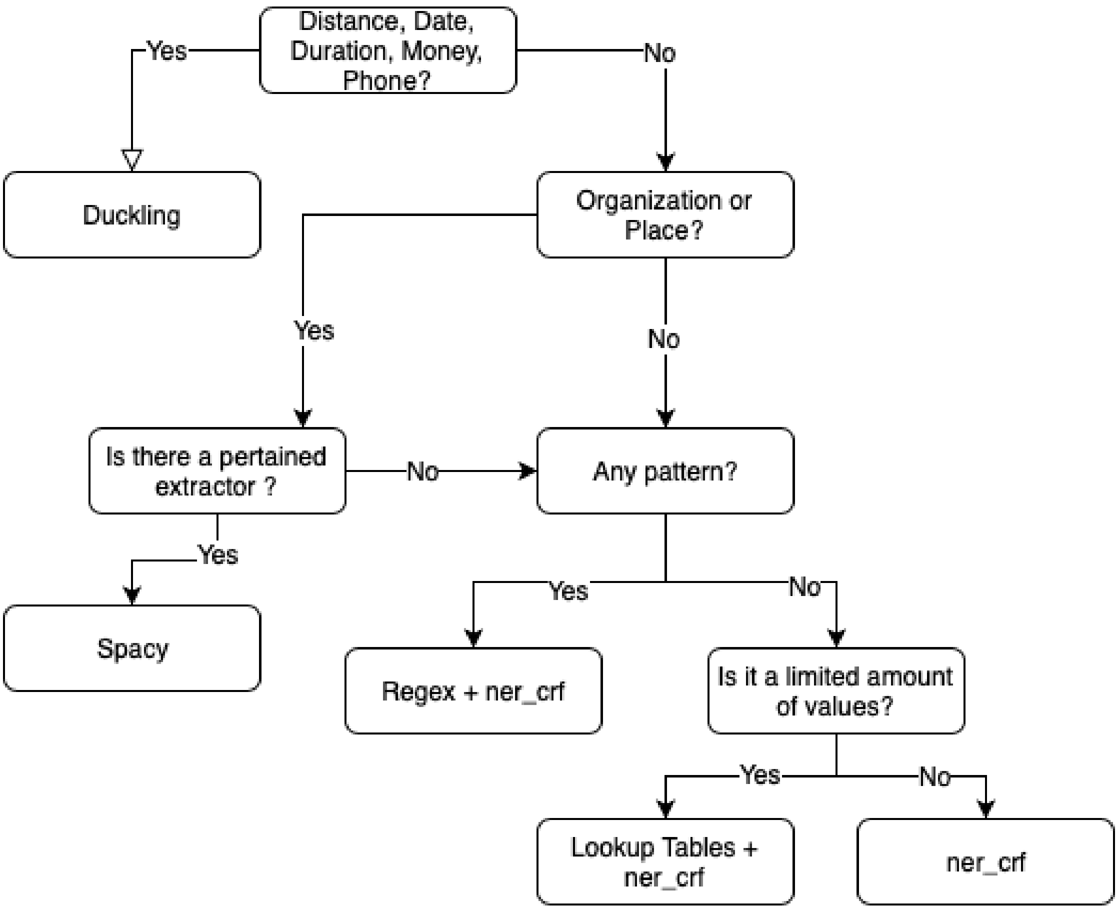

Figure 4.5: Diagram Of Ideal Entity Extraction Based On Fallback

The figure 4.5 is presented by RASA and allows us to have a better understanding of how the chatbot entity identification fallback service works and which tool must be used for each case. Despite this, at the moment several tools are still being experimented with and DialogFlow provides only a beta feature for entity extraction but does not detail the underlying process also Name Entity Recognition (NER) is still on a develop stage and there are interesting approaches such as the one from Strubell [17] which explains the underlying structure of looking through a set of rules and uses approximations on a pretrained model in order to get the best out of the model. Whatsoever, this shows that there is a trend to use pretrained models to solve the problem of entity identification as most entities can be identified from a preset entity and categorized model.

4.6 Context

The context of the conversation or Domain Classification relate to how an intent can pass extracted entities across other intents and use the extracted information to expand the conversation. This element is crucial in our chatbot since it allows us to use the previously obtained information to elicit or reiterate an intent in case there is no result that could be used to fulfill the requirements of the recommender systems. In order to understand how this works in DialogFlow we must get an understanding of how is that Domains get classified in upper layers and passed to other Domains or retained in a particular domain. Since two intents or more can live within one context and an intent might hold up for several contexts.

In order to understand how context manage and interact with intents it is important to clarify that the domain relies on information that must be passed among intents to hold a conversation. A good way to describe this lies within a normal context of a conversation where entities have to be extracted in order for the context to make sense. For example, if a George Washington was born in the year 1732 and then as a followup intent we ask who was he, it is important to pass the main entity which is George Washington to then next intent in order to be able to provide an answer using the current information. The time to live of a context is also relevant, thus if we want the information to persist over a span of intents then this must be defined too as an intent that contains too many entities could disrupt the results and the entities might overlap or the intent identification might fail.

Moreover, the elements that compose the identifier of a Domain Classifier are described by Jozefowicz [18] as classification in the neural networks domain and thus each element must go through a the network which yields a slow component. In practice our chatbot uses predefined domain identification, meaning that the intents map to which contexts but they context identification does not only rely on this but also on the IND and OOD utterance. To refine this, each intent maps which domain is received and to which domain it must be addressed in order to avoid having the input domain overpowering the output domain in case of a multiple domain definition across multiple intents. In order to calculate which IND(in-domain) or ODD(out-domain) takes precedence Young-Bum came up with the following [19].

To compute the utterance here represented by $\bar{h}$ we use a shared encoder and then the probability of the domain to be mapped must be defined using $\tilde{d}$ by mapping $\bar{h}$ to a 2-dimensional vector for each domain $\tilde{d}$ as described by [19]

$$z^{\tilde{d}} = \sigma(W^{\tilde{d}} \cdot \bar{h} + b^{\tilde{d}})$$
$$p(\tilde{d}|\bar{h}) \propto \begin{cases} \exp\,[z^{\tilde{d}}]_{IND}, & \text{if } \tilde{d} = d \\ \exp\,[z*^{\tilde{d}}]_{OOD}, & \text{otherwise} \end{cases}$$

where $[z^{\tilde{d}}]_{IND}$ and $[z^{\tilde{d}}]_{OOD}$ denote the values in the IND and OOD position of vector $z^{\tilde{d}}$.

Using this classifier we can exert and calculate the right utterance for each domain in a multi intent scenario taking into account the input and output context or domains. This behavior is represented both in RASA and thus in DialogFlow. In combination with both intents and entities this are the core elements of a query identification and information extraction for latter processing.

4.7 Composite Entities

As we previously defined in the Entity and Named Entity Recognition the entities are expanded from intents using different kinds of recognition through recurrent neural networks. In the case of composite entities which are entities expanded from the original entity. To refine this, an entity for example a T-Shirt or a Cardigan could possible and most likely have a color such as blue or white. Despite this, the refiner of the entity cannot exist if the entity itself is not present before. Following our previous example we cannot have a blue entity before defining to which entity it belongs. This allows us to classify easier the entities and gives us additional layers on the entity mapping thought this is only supported by DialogFlow and thus is one of the reasons that it was picked as the driver for the chatbot.

A sample of a composite entity is denoted in Figure 4.6 where we can see how an entity is expended by using another entity which will use a skip-gram algorithm in the underlay architecture. The idea under this is that we are capable of defining granularities for our chatbot in case a user does not only want to give a detail related to an attack but also wants to specify with which degree of complexity it happened. Take the following example as reference:
"The web server is receiving high loads that reach 80% of the total."
Hereby we can see that the entities to be extracted are mainly the high load and the server but as a composite entity the value of 80% is relevant for attack identification and thus is added to the composite entity.

4.8 Entity Definition for DDoS

Among the most relevant elements of the chatbot lie the entities which are meant to be mapped in the intents, thus they had to be extracted and inserted. For our case we

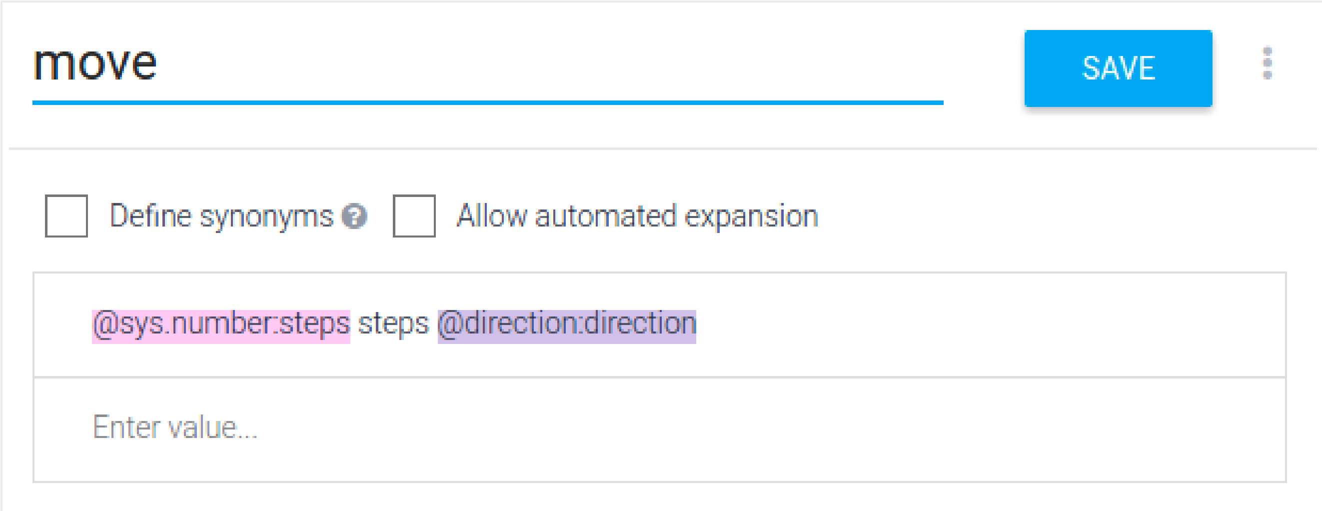

Figure 4.6: Composite Entity As Entered In The DialogFlow UI

researched into the most common entities to be extracted when a network administrator was under attack by a DDoS attack and thus we came up with the entity based table 4.1.

denial of service	denial of service, dos
High number of request	High number of requests, High load, accumulated load, high request number
UDP Flood	UDP Flood, UDP Packet Attack, UDP Packet, UDP
ICMP Flood	ICMP Flood, ICMP packets, single packet, ICMP echo, ICMP echo relay, ICMP replay
SYN Flood	SYN Flood, SYN packets, SYN attack, SYN requests
Ping of Death	Ping of Death, POD, PoD, Denial Ping
Smurf Attack	Smurf Attack, smurf ping, smurf request, smurf
Fragmented Packet	Fragmented Packet, fragmented request, fragmented information, partial packets, partial requests
Low Resources	Low Resources, low available resources, low amount of cpu, low ram, low disk
overflow spoof	overflow spoof, overflow spoof packet, overflow, network overflow
unresponsiveness	unresponsiveness, unresponsive, takes too much time, slow, froze, not responding, not responsive

Table 4.1: Denial Of Service Entity Identification

It is important to mention that the elements that were added for the definition of the DDoS table are coma separated values denoted by several elements on the second column which map to a single identifier on the first. All these entities can be expanded to match a large amount of identifiable elements that are present in an ideal DDoS attack and thus create a generalized solution. In our case we based the table on the work from Chen [20] which described some of the basic attack elements in DDoS related issues and used them for identifying vectors. Also, the elements can be expanded and used in combination with composite entities such as numeric values and distinct identifiers. For example, if a server is receiving a fragmented packets and the network admin wants to describe how many fragmented packets are being received this is a possibility that could help create a solution or even distinguish a DDoS attack from another kind of attack.

Although, this is only a partial part of the whole setting that can be found in the code of the chatbot which expands the entity extraction with other elements such as victims of the attack which can range from single units to multiple vectors are can be required in combination with the described base entities. Moreover, at least this two elements, victim and attack vector are required. Depending on the granularity of the chatbot more entities might be required and mapped to fulfill the needs of the recommending systems. This will be described as well since the model relies on the current status of recommender systems, for our proof of concept the work was based to fit a general model and in detail with MENTOR [1] and thus the elements that are present at the time are meant to match the ones from the recommender system.

Furthermore, DialogFlow features fuzzy matching and syntax analysis meaning that if a user inputs a result that is close enough to the intent the NER will match a good approximation to the entity. This can be further tweaked but for our chatbot we enabled the fuzzy chat to enable input and testing in cases of results that closely reassemble the original entity such as "ICMP and icmp" or "fragmnted packet" and "fragmented packet".

Moreover, it is also important to list some of the other entities that were used in the chatbot such as the types_problem which allow us to pick the right elements or attack vector whether we are seeing an attack or we are looking to information from an error. To expand how the Entities relate to each other we can use Figure 4.7 were we can see that there is a relationship among most entities that would allow us to extract the information in a cascade manner, thus if one element is obtained or present then the following elements would be desirable but this heavily relies on the needs from the recommender system. Also its worth mentioning that the intents that use this could map to one or more entities so this should not be the limiting in our case.

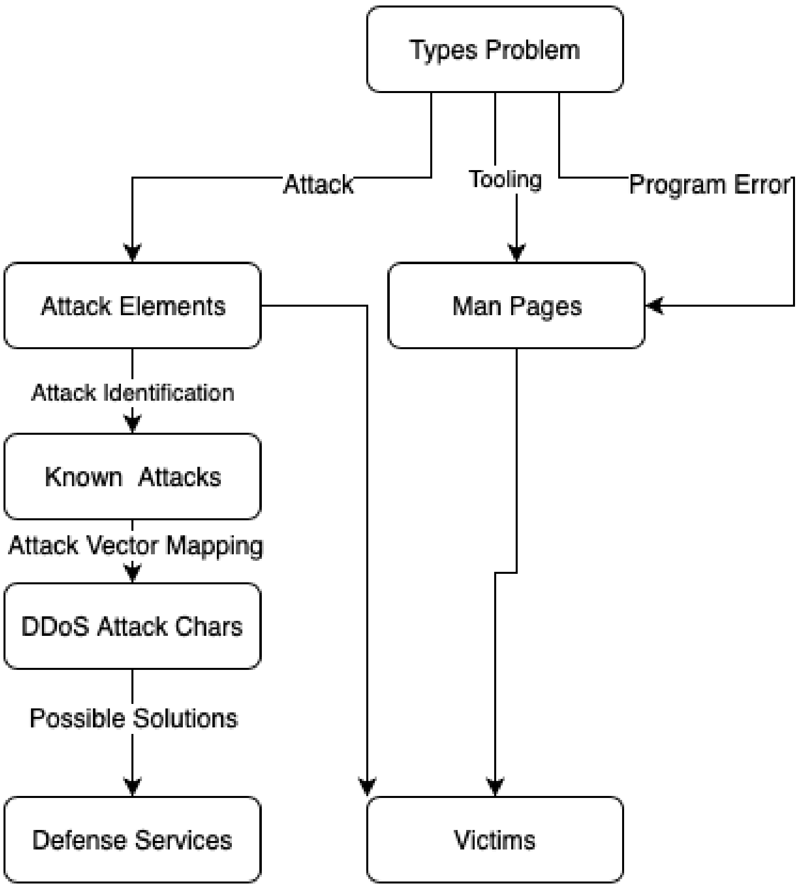

Figure 4.7: How The Existing Relations Work In The Entity Tables Within The DialogFlow API

Each element of the entities expands to a set or subset of similar entities which are matched by using fuzzy matching. Regardless, there are several ways in which we can expand the existing entities by using a similarity dictionary to increase the amount of similar entities to match the NER. Whatsoever, we did not proceed with this method in our case due to the precision needed for each intent and the non standardized way to calculate a vector based on its context. Using a skip-gram method could have given us a good amount of material but due to the hard process of computing nearest neighbors in the attack context, the results could have ended being to skewed to be used by the DialogFlow model. Also, some elements presented are not in use, for example in figure 4.7 the Defense Services are not being mapped but this will be later explained in the next section. Is important to say that most entities were optimized to aim for DDoS detection as this was the proof of concept of this work.

4.9 Creating the Intent Map and Training Phrases

The Intents are made to map all the user input into the right entities, for our original idea to take place we came up with several possible intents to map into tools to aid the user and set a low or high level of granularity depending on the expertise of the user. Whatsoever, we realized that mapping a large amount of entities would require a relatively large amount of intents and thus to provide a proof of concept this was no feasible. In the light of this, we decided to aim for specific use cases such as attacks and to get into details into the DDoS attack which is a very well known and detailed type of the attack within the cybersecurity area. It is worth mentioning that throughout the setup many cases where implemented and many intents could possibly overlap others, thus we do not only have intents aimed towards the DDoS scenario but also the default ones for tool assessment. Nonetheless, all these intents must be mapped into the right solution and thus we will only expand into the ones aimed for DDoS identification for ease of use and to have a deterministic point of view. Also, all the test cases will be performed taking this into account and the chatbot is optimized to work under these settings as a proof of concept. Therefore, even though this set of intents was created for the mentioned attack the basis would be the same as it will be described later on for all other types of attack.

An intent in DialogFlow is composed of several elements that allow us to increase the probability rate of a right intent classification. Within the Intent of DialogFlow API we must define several elements that will aid us to help or expand the granularity accordingly.

- Contexts
- Events
- Training Phrases
- Action and parameters
- Responses
- Fulfillment

All the elements that belong to the list above are part of the intent and a correct definition of each will allow us and expand the user experience and the accuracy rate depending on the right settings. We will describe them in detail and each element will shed light on why its dependent for the complete use of the entity, thus elements such as Contexts or Training Phrases should not be strange at this point but will be detailed so there is a high level of understanding. It's also important to mention that to set up the granularity of our Chatbot correctly several intents are needed, we wont get into details with this but it is important to mention that not all intents will define the same granularity and thus having multiple intents for multiple contexts is acceptable in the way that otherwise it would be hard to come up with different intents for the same granularity, thus intents map to different contexts depending on the granularity selected by each intent.

4.9.1 Contexts within Intents

Whenever we must define an intent this must belong to a context and the context will parse and pass all the elements among the intents so they can be further used. To understand this as explained in the domains we must pick to which element it must go into and out of, this also can help to set for how many intents we should keep the context. This is generally used to keep track of the right information across intents and specially useful if you have more than one parallel or similar intents. Something important to take into account is to prevent overwriting the information extracted by an entity in the context, since a context such as George Washington can be passed across the whats the age intent but it can also get overwritten by another NER such as Abraham Lincoln and thus pass the context of George Washington but overwriting some elements but not all. In order to prevent this our chatbot focuses on single contexts and unless is quite necessary has a relatively short time to live on the context across intents such as 1 or 2 intents as a maximum. In order to describe this, we can picture a set of intents that relate to each other through the extracted entities but communicate using contexts with a short time to live.

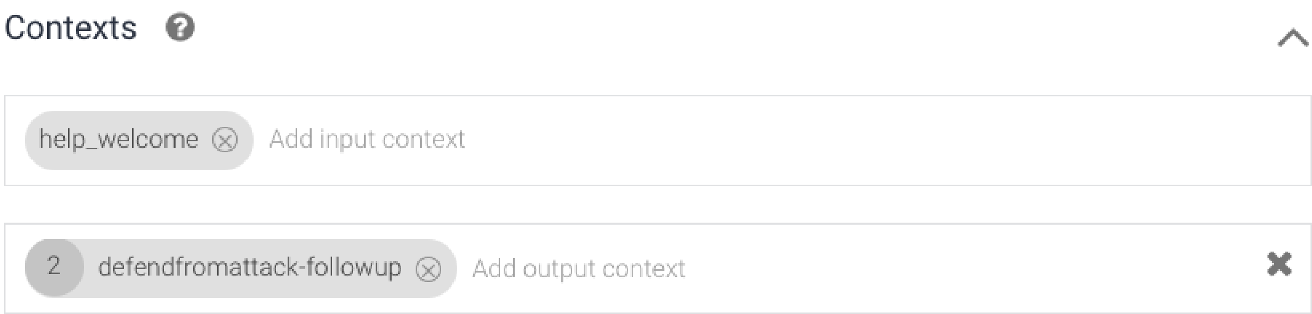

Figure 4.8: Demo Of How A Context Interface Looks Like And The In And Out Contexts From The Domain.

As shown in Figure 4.8 the number of time to live must be defined in advance and is possible to have multiple input contexts and also multiple output contexts. The benefit of this is that another intent is matching the classifier it will also pass the context is it matches the input context increasing the rate of accuracy of the intent classifier. Therefore, we set up a set of contexts within the intent dialogflow that relate to the following:

- Welcome Context
- Identify Issue (Attack, Tool, Error)
- Map Attack
- Gather Attack elements
- Iterate over attack elements
- Goodbye/ Parsing

This are the default contexts used to get the information through each set of intents, and for each one there is at least one or more intents matching either in the input or output.

4.9.2 Events

Events are meant to take place whenever there is a desired action without being mapped by an intent. For example, if we decide to extend the usability of our chatbot past the UI and we want to trigger an intent based on an event such as location and thus ask a question or query for an intent just in case of this event or trigger happening. There is room for improvement here, as its possible to use this feature to automate the process of solution finding based on standardized problems such as Fragmented Packet Count above certain percentage and thus launch intent to query for current web status from the user and based on this information create a solution that could aid the network administrator to ease the problem even before the user asks the chatbot for help.

The current status of the chatbot does not allow us to expand on this but will be further discussed in the Further Works section.

4.9.3 Training Phrases

Perhaps the most important part of the chatbot and a remarkable part of the research takes place in the Training Phrases for the intents in the chatbot, since without them is hard to come up with a feasible solution that would approximate to the right entities. Based on this training phrases and the NER classifier the intents get mapped and identify entities, thus providing the right set of queries is important and can ease the job of a Fallback event.

The underlying work of providing training phrases aims to have enough information in the intent classifiers in order to assess if the loss function has to correct and assert in a better way or if the result was accurate enough. For the intent training we must provide positive examples or examples that reassemble the original query or that are acceptable enough.

Since the process takes place in a higher layer of a pre-trained model there is no need for negative examples in this case, whatsoever it is important to take into account that fallback intents which are queries whenever there is no apparent result from an intent usually mapped from a context and those are trained using negative examples or the opposite of what the intent should be. The underlying concept creates another classifier for intents to expand the basic classifier with elopements that could be considered unusual. Despite this, all classifiers calculate the accuracy of an intent in order to match it to the best intent and if certain threshold is not reached then it will try with the fallback intents. Similarly, we have default intents that will match an expected response and can be automatically generated such as Yes, No, Later, Cancel, More, Next, Repeat and so on. This allow us to generate intents for simple responses within the conversation and through a series of tweaking we aimed to the best interaction from system administrator to reach a DDoS attack scenario which by itself is rather complex.

It is also important to mention that the training phrases must reassemble a close interpretation of the real query by the user but should not necessarily be equivalent since there must enough range of similarity for a proper classification, meaning that between training phrases there must be enough space for the real queries. Also, we could further process the successful queries obtained by the users and reinsert them into DialogFlow to latter use them as training phrases to aim for a constant improvement.

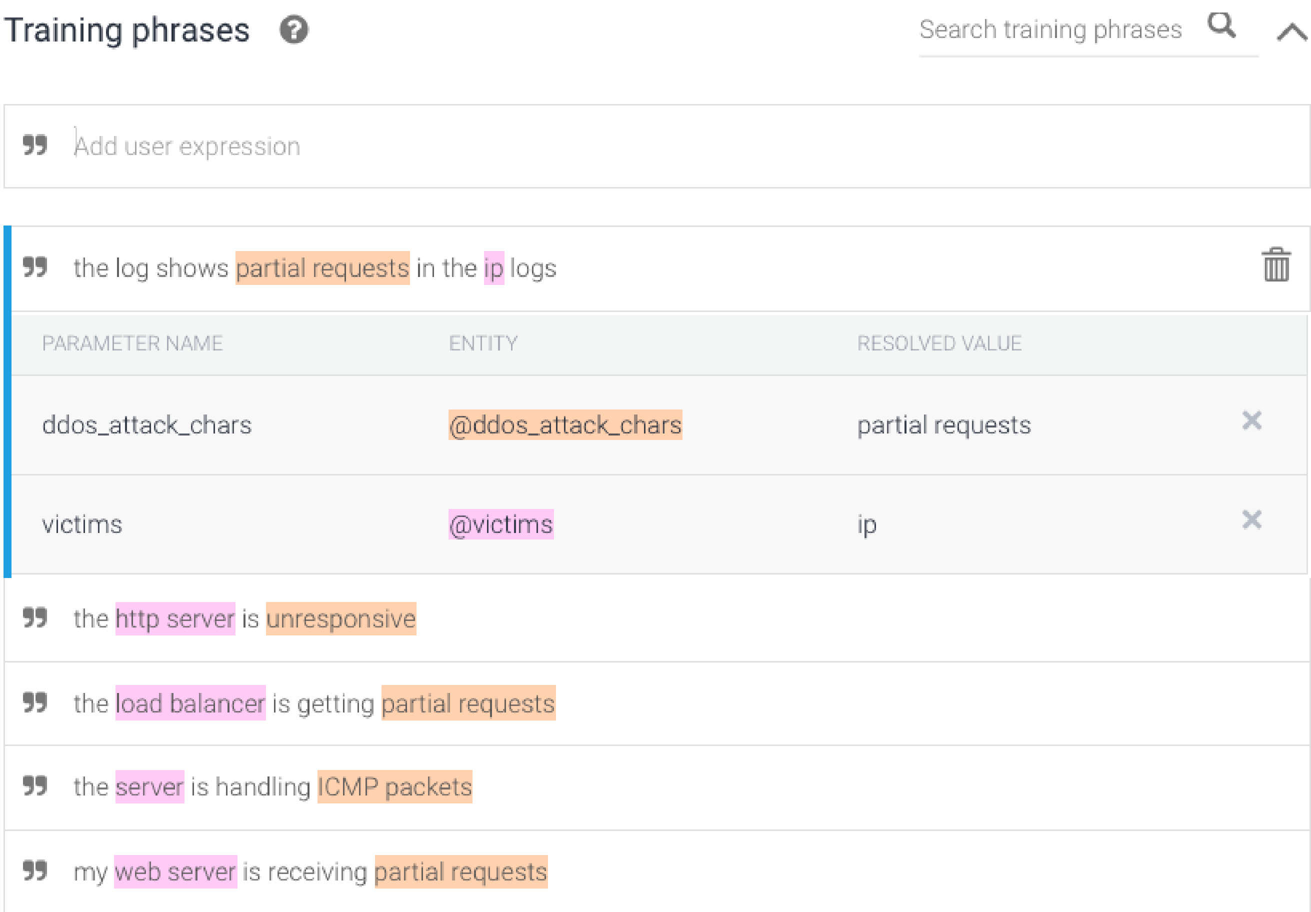

Figure 4.9: Example Of Training Phrases With Identified Entities Prior To Training And Resolved Values

As shown in Figure 4.9 we can see that the training queries must reassemble to the real query but should not be contained. Also, it is important to mark down the entities to help the NER to identify the entities within the context. Most of them are resolved automatically after a few training phrases are entered meaning that you could already have an approximation of how are the entities getting resolved by the previously entered entities.

Is also worth mentioning that the parameter value represents the bucket or bin in which the entity fits, thus it behaves rather like a category which is also parsed in the generated JSON file. The resolved value is the actual value which holds the value for the element, as mentioned before an intent can have several entities of the same type within a single query.

4.9.4 Action and Parameters

Another important part of the chatbot interaction is the actions and parameters to be performed. The action can allow us to send an identifier through the intent that will hit Fulfillment and allow us to move the response or gather information with this identifier. Also, this action can trigger certain activities by Google such as turn on lights or similar but must be expanded into how the work applies to each element. In our case we decided not to use any action within the chatbot since the UI only requires a simple conversation without too much content and does not feature location or any additional service that could be provided through the google API.

Then on the parameters side it is important to specify which entities we aim to gather and which ones should be present in order to have a successful intent, this particular area is very important since the outcome of the intent might not be successful and hence instead of having a fallback intent to redirect to the original intent we aim to use a prompt within the intent to specify a question that guarantees that the entity will be gathered and thus mark the intent as complete. Once again, once this happens the response is passed to the Fulfillment engine and then shown in the UI. Even though we must define the parameters so the chatbot can know what to look for, not all of them are required and thus the prompting will only happen if the entity is marked as required. Furthermore, we can also ask for a list of values meaning that you could describe a set of symptoms of attack vectors to be parsed and if one or more are missing then they could be prompted. Likewise, all the elements as a list will get parsed and tried to match with composite entities in case this is available.

Since the prompt is our intent within the intent we can use elements from the context to guarantee or rise the possibility of the desired entity to be gathered. Within the prompt we can use elements that were already extracted for the intent, for example we could extract the values server load but if the kind of attack was not explicitly mentioned then we could make a prompt similar to this:

"Please provide the symptoms that the server was experiencing during the load"

And thus passing the already extracted entities to ask for an accurate answer. In order to be able to bypass or score higher in the Turing test, the chatbot features a set of prompts meaning that it will not ask the same question twice to simulate the conversation with a human or simply to try to ask the same question in a different way so is more likely that the right entity will be obtained from the input of the user. Whatsoever, since we just require a handful of intents to be able to match MENTOR not to many prompts were defined but could possibly be expanded according to the needs of the model. Also, the parameters can be reused within the intent meaning that if at least one parameter was identified it could be used to gather other similar parameters. Meaning that for example if the web server is under ping of death requests, we could also ask if any other device is suffering from a ping of death request directly to the user if the NER identifies the entity successfully.

4.9.5 Responses

In order to make the chatbot dynamic and enable a good set of interactions providing more that one response is always desired, these responses can go from very simple responses using the extracted information from the original user queries or even more dynamic including custom content to expand the user experience. Usually the message flow travels across fulfillment which will be described later on. Under the hood, the message from the user gets extracted and then sent to the DialogFlow Engine which runs the previously described algorithms to obtain the intent classification, the entity name and maps to the right context or domain. After this steps, the message can be sent if decided to fulfillment which will preprocess the message using JavaScript ES6 code. This can be used to expand how a response gets mapped to the user and which elements are used in the response. For example, we could obtain the information about the DDoS attack and based on the type of attack lookup in the database for solutions that could match this and provide a real time response using fulfillment. In this case the recomender system would not be needed anymore and thus for our chatbot the Dynamic responses are not used but in practice we could provide code, images, audio and so on and so forth to enrich the response from the chatbot. If required we could also send locations or elements related to the Google API stack of applications.

The answers can also be sent through other already integrated services that do not belong to the Google Stack API such as whats-app or slack, giving our chatbot a wider range of opportunities to interact and expand the usability with teams or even pre programmed bots. Also, it's worth to mention that we can mark the end of a conversation within the response and this will flag the intent as the last one triggering a final response and our chatbot will use this flag to create an array that will be parsed into a JSON which can be fed to the MENTOR system to create an accurate solution. As a further work in this aspect we could possibly wait for the answer from MENTOR and using the output we could modify the answer and provide an answer in real time for the network administrator. Moreover, we can also submit multiple responses of an intent meaning that not only one channel can be used to answer for example if we manage to identify the issue we could possibly forward the solution using another channel such as slack or a call to an endpoint to implement the solution provided.

4.9.6 Fulfillment

The Fulfillment aspect of an intent is one of the most interesting areas of the chatbot since here we can manipulate the data as it reaches the server and thus we contain both the information from the API and the processed information from the machine learning algorithm. Thus the fulfilment allow us to extend the existing functions using customized code, most of it can be expanded into real functions using an external micro service. A good example of this could be shown in Figure 4.10 where we can see that through Fulfillment we can connect to the cloud functions provided by Firebase and also the database containing all the tooling information extracted from the Manpages which is stored in datastore as objects.

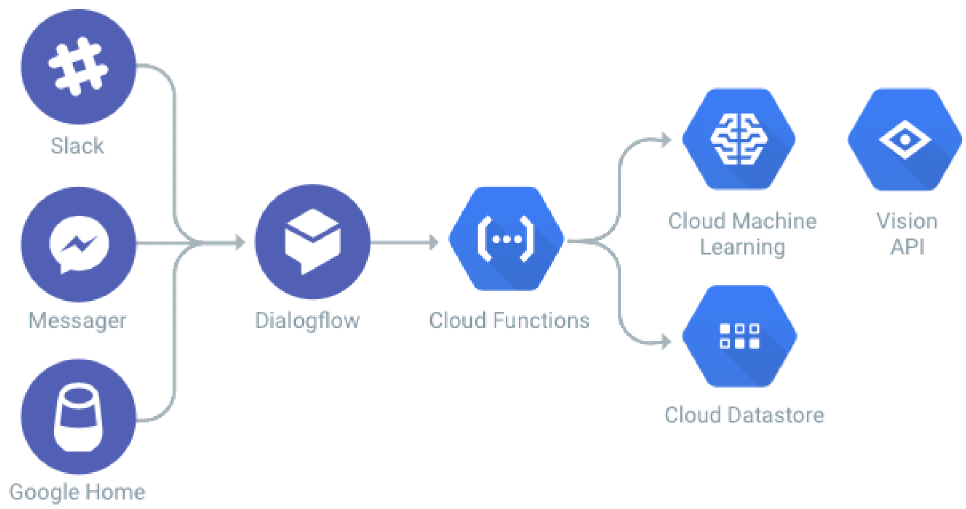

Figure 4.10: Map Of How DialogFlow Can Interact With Cloud Functions To Expand Functionality Of The Results

At the moment the chatbot uses the fulfillment in some important intents to fetch relevant information and to help to build the JSON tree and storing the built information external of the contexts and thus persisting the information. To expand this concept we can see that the elements obtained through a simple subset of requests is shown in Figure 4.11 where elements are filtered already such as the classification score or the amount of intents required to reach the stage. All these are filtered using the fulfilment and since the fulfilment integration uses data store is also possible to use elements obtained from there for further expansion of the response. Since the method control for fulfillment is enabled we could use other machine learning algorithm to further train our responses or to even generate a solution to have a solution independent of the recommending system.

4.9.7 Overall Implementation

All the elements have to be used in conjunction in order to generate a set of intents that can interact with other intents and fetch the information throughout the process. In order to achieve this we must define which entities are to be extracted in order to fulfill the goal

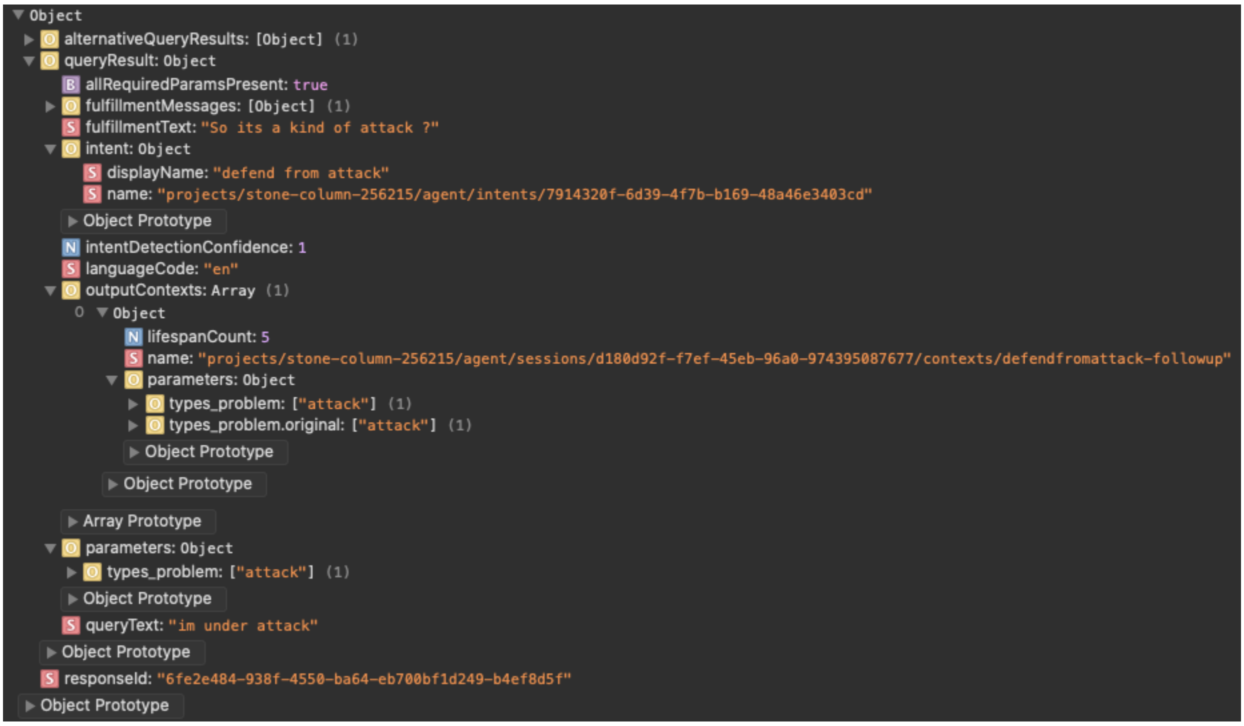

Figure 4.11: Elements That Are Contained After Processing An Intent In Fulfillment In DialogFlow, Extracted From The UI

of generating a JSON file that can be fed to MENTOR [1] to generate a solution for the system by using the external services that can guarantee defense against attacks. To see how is that they interact with each other the diagram from Figure 4.12 describes how is that the components interact with each other to fulfill either goal. Is worth to mention that despite the amount of intents these have to be created in a unique manner and thus are limited by the amount of intents we want to create. For our proof of concept we limit ourselves to the flowchart from Figure 4.12 but it is important to mention that we must go through this procedure since an important element of NLU is the fact that we want to understand what the language means rather than what it is per-se.

Each element correlates, for our aiming round we take into account the hello intent which will make sure that there is a human validator. Later on, we have the attack identifier which will aim to identify if the users suffers from an attack, a problem with an application or is looking for help with a tool. In our case we will only aim to fulfill the attack path. Once the attack is identified which could also be spread as a set of entities and symptoms to match to a right kind of attack but for now we will try to aim for DDoS attacks which have rather common symptoms. Once this is done, the user could input large sets of intents to match as long as there is a victim and a symptom or a set of symptoms which are also referred to as attack vectors. At some point all the details about the attack will be described and fetched to this intent, thus will pass on to the MENTOR [1] funnel. In this funnel we will aim to extract all the key elements to create a recommendation through mentor. To list the elements are:

- Type of Service: If the service requested is active or passive (Right now or to prevent later on)

- Type of Attack: Details about the Attack (Gathered by the attack vector intent)
- Region: Defense systems and deployments are available per region, thus this must be provided
- Deployment Time: How much time is available to implement the solution (Larger solutions are better but take more time)
- Leasing Period: The period for which the service must be contracted for at least.
- Budget: The budget for providing the service that will guarantee a defense.

With all these elements into account we can start to describe the funnel and which elements are mapped to which intents. This is a very important part of the chatbot as without it is impossible to generate results that have a good amount of content for the recommending system to work. Is also notable that if at some point we do not want to fit the recommendation from mentor we can type 'End here' or 'Finish here' which will terminate the event with the parameters handled up to the time. It is also important to say that from the gathered elements the intents can be triggered based on the input context thus the mapper or vector element attack identified can be queried from several points of the conversation just to guarantee that no data is lost due to missing the context of the conversation.

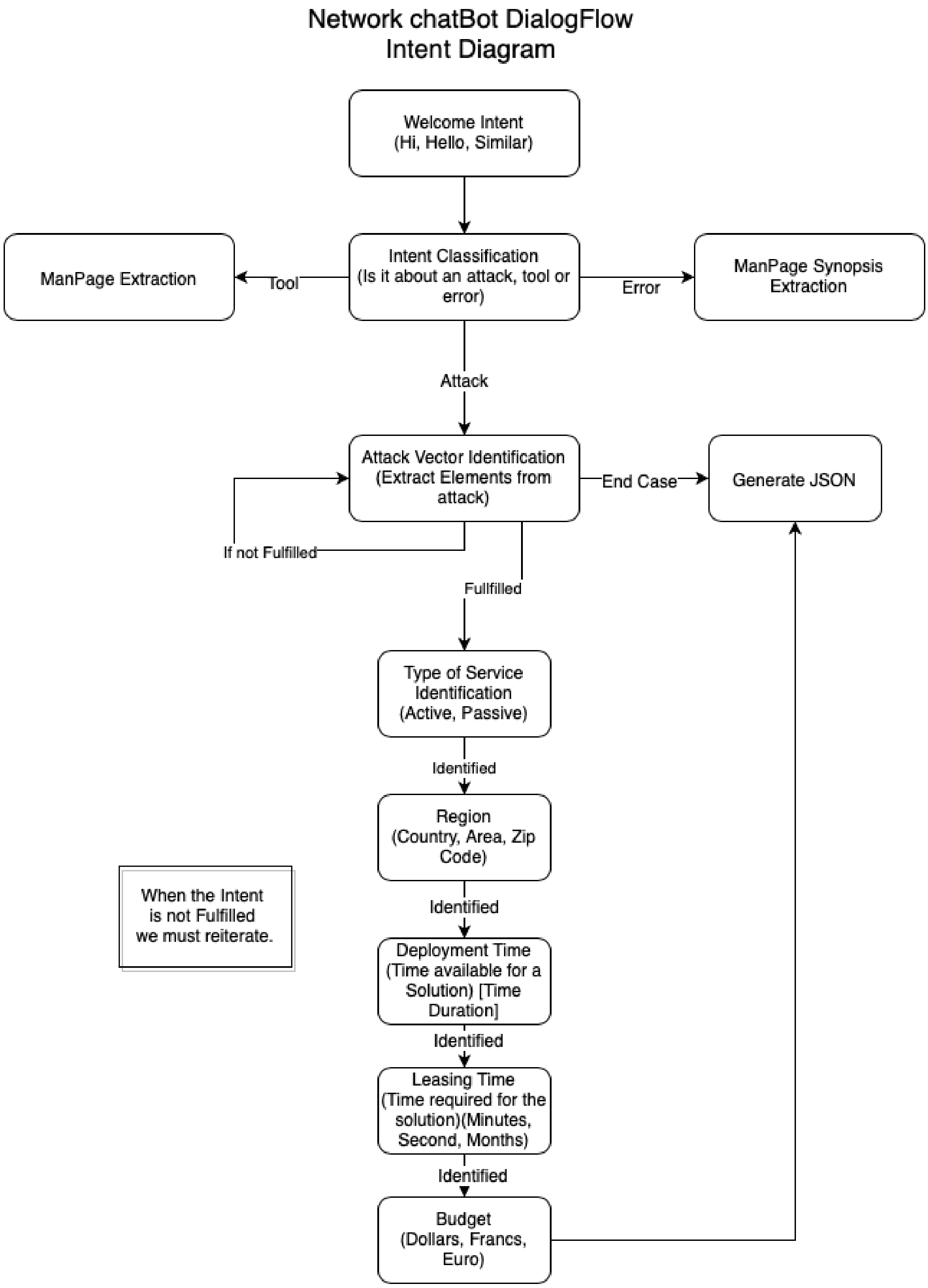

Figure 4.12: Intent Classifier Flow To Map Elements Through The DialogFlow API. All the Contexts are excluded from this graph as they are described in the context section.

Chapter 5

Prototype and Implementation

5.1 UI and Usage

The application was designed using Javascript and a React Framework in order to allow hot swap and component structures that can expand the usage even further. For This purpose we used a simple stack with a pipeline. The underlying idea relates to how to query the intents from the user and at the same time build a stable pipeline that will allow an easy development using tools that are available online. For this the backend of the react client uses Heroku for the the deployment stack, meaning that the set runs under an automatic deployment handled by Heroku on the client side of the application. Thus a Git Repository was created to map all the elements and trigger builds made from the repository. Thus we would have the heroku backend pulling from the git repository and launching the app in an updated manner since it uses the latest commits to automate the deployment.

The application itself does not feature many functions besides obtaining queries as input streams from the user and filtering or cleaning the information prior to be sent to the DialogFlow API. Once the information is received by the API then Fulfillment will gather all necessary elements and refine the information that must be returned to the client. Once the client receives the information then it is stored in the session Storage of the browser where we parse elements such as the identified entity and the entity bucket. It's important to mention that the bucket where the entities are stored could have different granularity and thus are likely to differ from implementation. In our case to handle a proof of concept we aimed to have a standard implementation aiming to a standard attack under this concept. Whatsoever, once this elements are saved in the client we can manipulate the answer to be shown to the user but the desired behavior will log the result and keep adding elements over a custom array.

One important aspect of the application is that in order to communicate with the API of DialogFlow it must use custom keys that are only available using the framework provided by google. In our case google had issues switching from the API v1 to v2 and thus we had to hard code the used keys and create a custom login screen that would allow us to interact with the server directly without a server side client. Thus there is an extra class that is outside of the scope of the initial functionality of the application that is used for this. Moreover, the functionality of the forms do not rely on an external framework besides react. This will allow us to have a interactive web form that can be used with almost any device, to show the interaction the following screenshots of usage are shown.

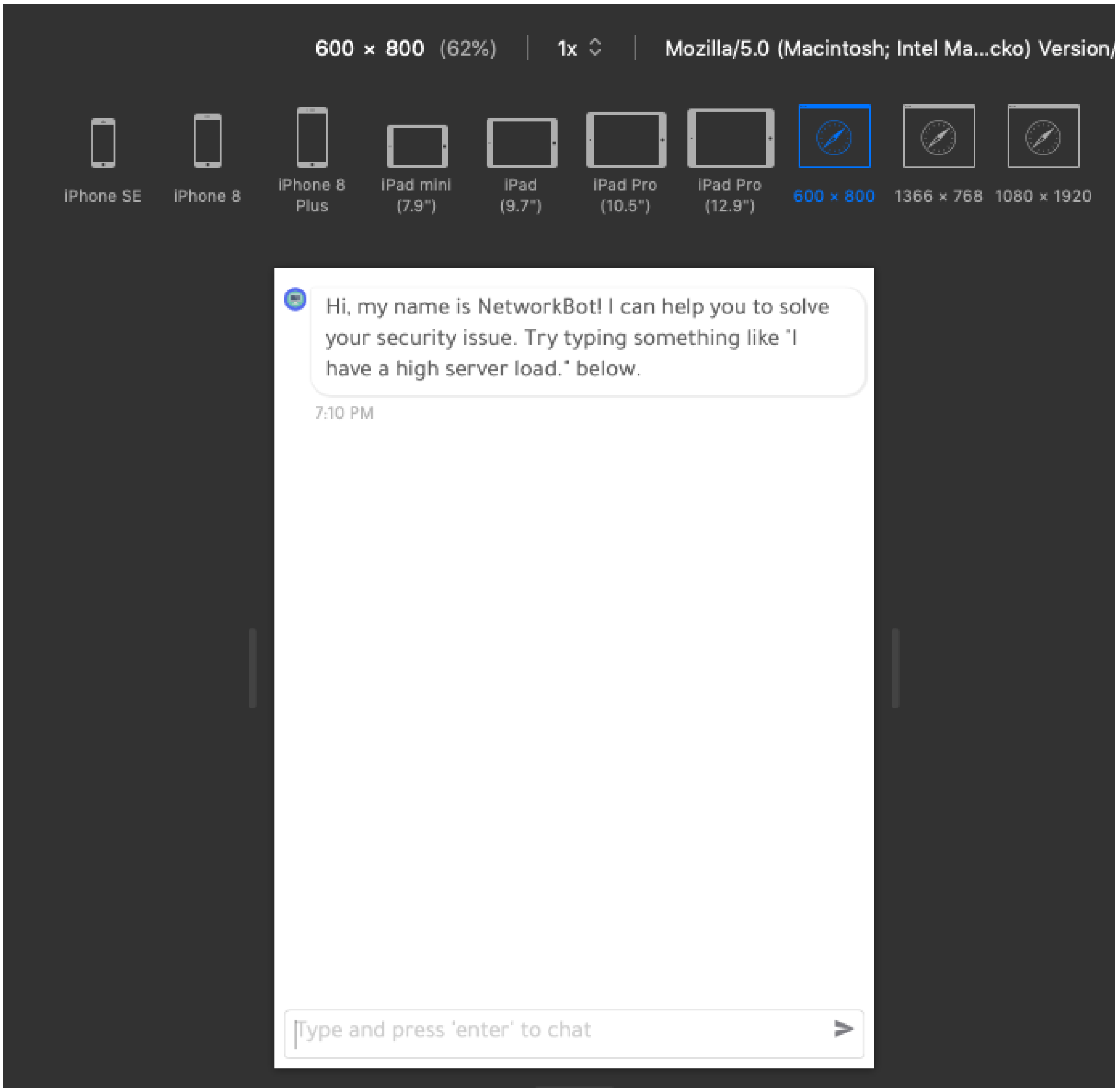

Figure 5.1: Web User Interface For The Network Chatbot

As seen in Figure 5.1 the react form scales according to the client need yielding a webclient that is reactive and has a relatively low page load. To expand this, the image denotes a simple UI loaded through a web server that is responsive and scales meaning that it can be used in any environment or device. For further testing the app has used in three environments Safari in a iPhone 11; Chrome on a Android Pocophone F1 and Safari and Chrome inside of a MacOS X 11.01. For all of these options there is also a proof of the responsiveness in Figure 5.2.

To expand on how the User Interface works, we can use Figure 5.2 to show that the input from the user is seen or denoted by blue highlighting and the responses from the chatbot

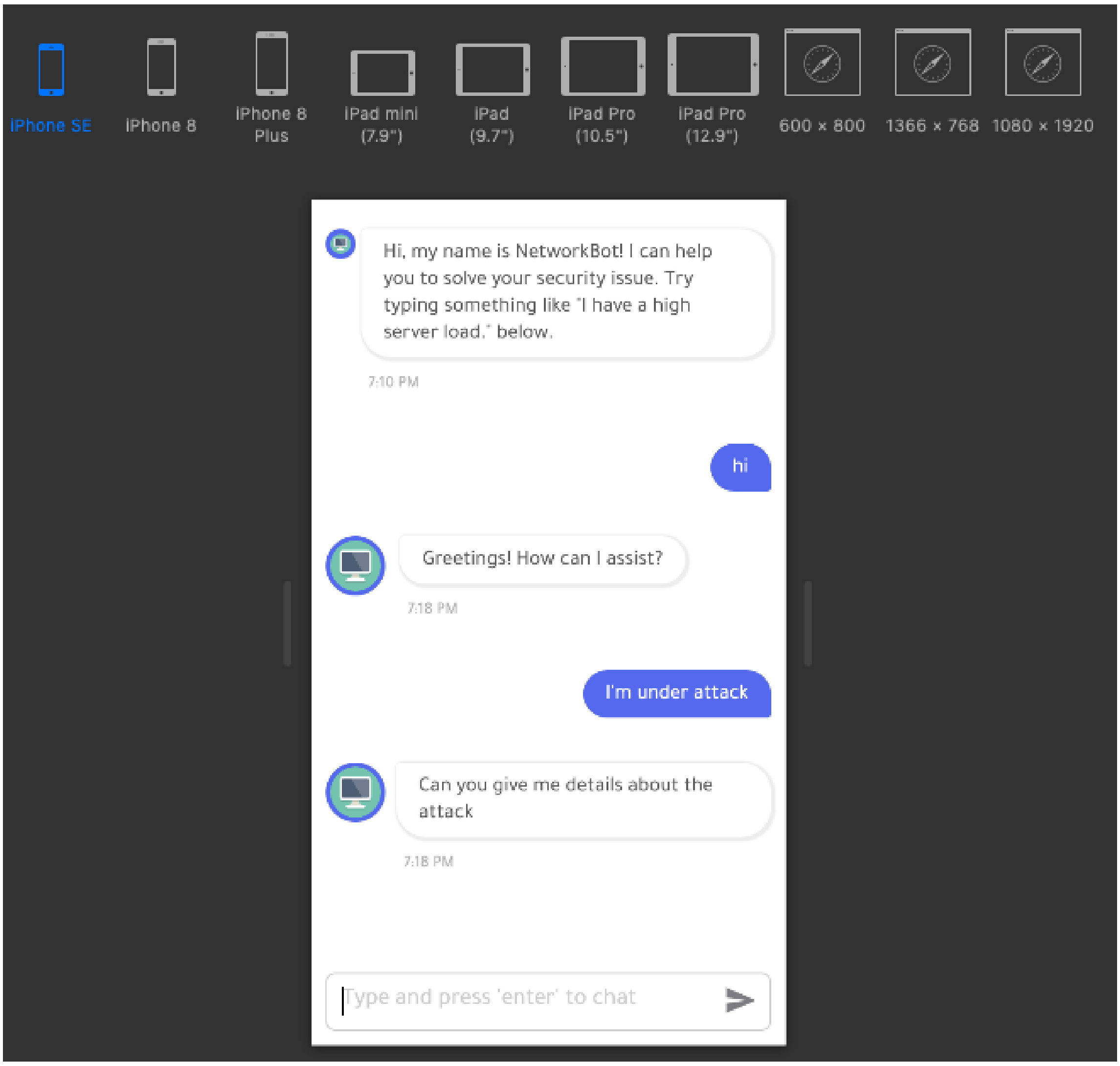

Figure 5.2: Responsive Web Interface As Seen From An Apple iPhone 8.

are in a white bubble. Each input from the user should map to a certain intent and each intent thus to the right chatbot. In our ideal implementation all the intents are mapped correctly, if a user inputs an incomplete session then this is logged by DialogFlow and shown so the intents can be improved to map all the necessary items for the Intent.

On the other hand, the tool was built to support further intent responses such as mappers from external resources that can be fetched from the tool like voice commands or images if supported by the DialogFlow integration. Also, it is worth to mention that the tools that are meant to interact with the system might support other external elements such as images or voice coming from the backend. For the proof of concept this was not supported but will be extended on the further works section. Nonetheless, the agent will also be capable of taking the elements from the input as long as they are text related and process them without using fulfilment. Other aspects such as the generation of the JSON file are also supported but rely on an additional code snippet which will also be described later on.

We can see in Figure 5.3 the full structure of the solution including the web client and

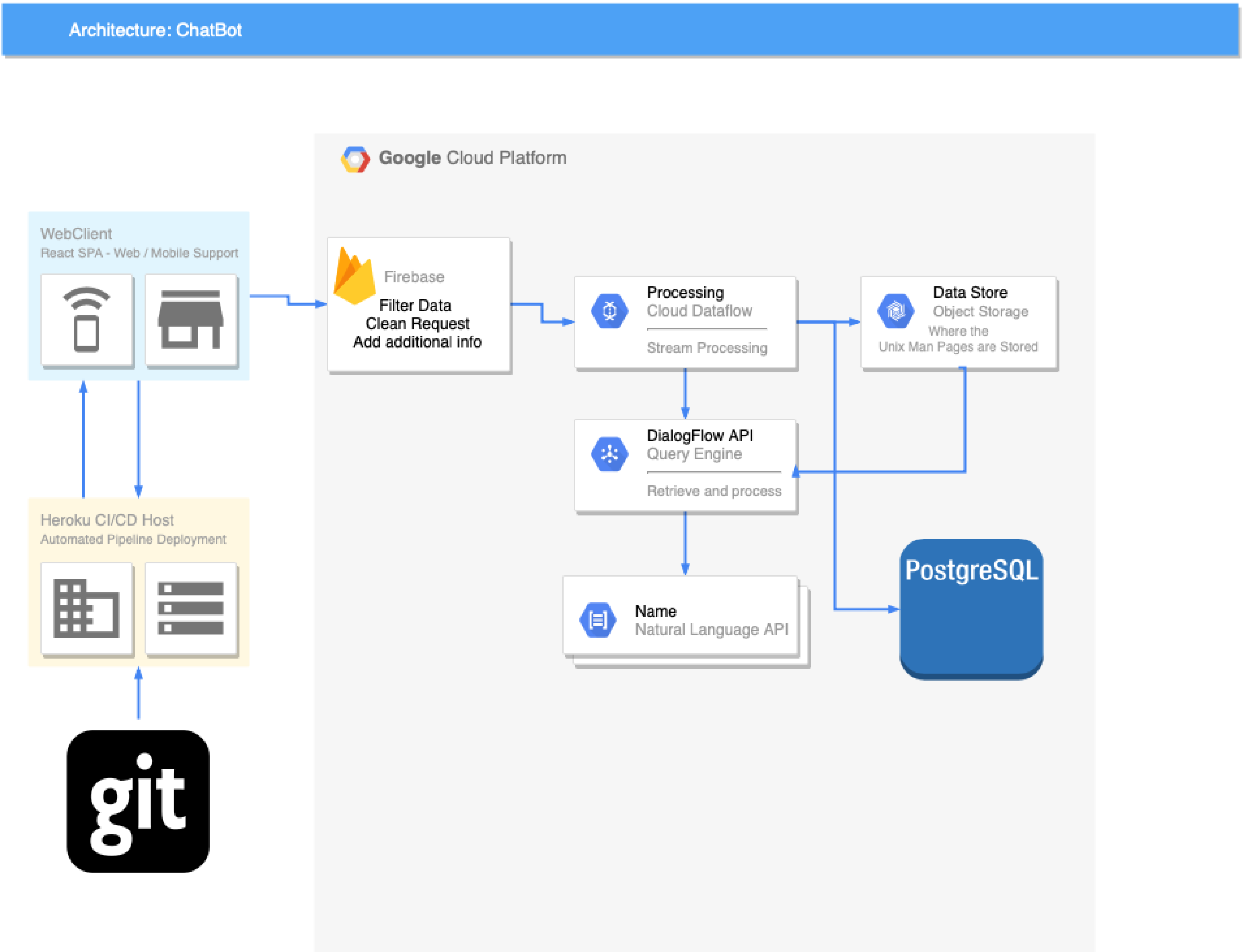

Figure 5.3: Architecture Showing All The Steps Involved In The Overall Architecture Of The Solution

how it interacts through the whole pipeline. As described in previous sections, the code is served from a git repository for the web client that gets automatically deployed to a Heroku Pod after passing a successful CI/CD pipeline and then once it is server to the user, it will generate authentication methods that will be served to Fulfilment which will handle the information received and will pass the filtered information to DialogFlow which will run the query through the classifier and try to match existing intents, this will be done using the Data Store database and the postgresql database. This is an abstract overview of how the implementation of the application looks like in production and how the components interact with each other.

Chapter 6

Evaluation

In order to define a schema that could hold up to the already defined elements we had to come up with methods that would allow us to quantify and expose valid ways of presenting data. To comply with this, two methods were preselected; for the evaluation we focused in using a set of predefined conversations that would map to a intent classifier and thus would allow us to quantify the reliability of our intent by using the Accuracy Index of each intent based on the query. This means, that we used the loss function from the model to predict how accurate was the result in comparison to our training phrases and would quantify this by giving a number ranging between 0 - 1, which stands for the percentage of reliability. Moreover, we also defined a second way to quantify our result by preselecting a set of Case Studies that would reassemble a use case of the tool. The idea behind a case study is to quantify the conversation and to give a use case so besides the numerical evaluation we can see how reliable or usable our model is. To introduce the results we will start by describing the quantitative analysis.

6.1 Quantitative Analysis

To have a clear idea of how the loss function works, we must understand the underlying concept in a neural network. As described by Zhao [21] the loss function is

$$\sum\nolimits^{batch}(sim(a,b), sim(a,b_1^-), ..., sim(a,b_k^-)) \tag{6.1}$$

Described as the loss function from the RASA NLU; where a are the bag of words, b are the labels from the training set, and negative b are extracted from the labels, (a,b) are positive entities of pairs that come from the pretrained set where (a,b) are the pairs. sim() is the similarity function that calculates the inner value using the cosine similarity and finally L denotes the loss function between the positive and negative pairs.

Thus this would allow us to compute a negative or opposite value for each intent as input in the classifier, meaning that the reminder of the loss function is the accuracy or relative accuracy of the intent classifier result. For each set of values we would have an estimation of the real accuracy level and using this information we could predict the error

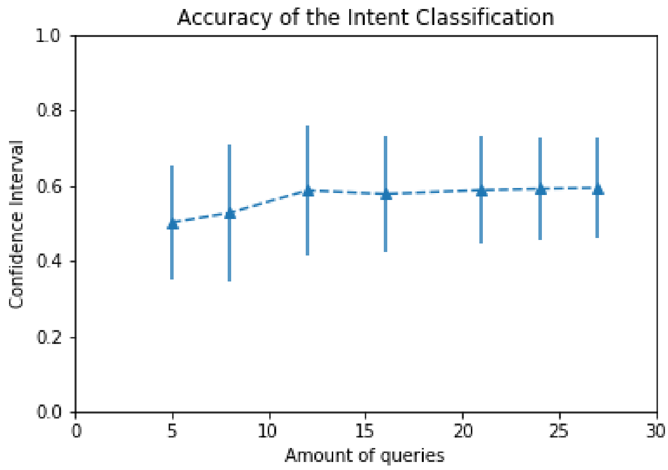

Figure 6.1: The Scale Of The Validation For Each User

rate or the relative error rate within the set by using a standard deviation across the group or test set. In order to generate a valid test set we predefined a conversation with the chatbot that could yield the needed results this can be seen in figure 6.2 where the data was split into several sets. In a first instance all the information coming was split into arrays of queries containing 5, 10, 15, 20, 25, 30 and 35 elements. This elements were selected sequentially and used in different sessions assuming that each conversation was an independent event. Therefore, for each set of conversations we could define a preset amount of data which in our case ranged from 0.5 to 0.6 meaning a median of 55% identification rate and classification, even though this might seem like a relatively low identification rate our bot was set to work and accept a yield above 40% meaning that most of our queries were successfully identified. Despite this, for each case of set of conversation there was an unidentified query or element that was not above the 40%. Is worth taking into account that the result does not mean that there is a cumulative error since the dataset is unique and thus the errors identified in each stage are transferred to the latter, for example if there was an unidentified intent within the first set, this intent will also appear in the second set since the sets are not unique. The identified error array is as follows:

Round Number	Number of Errors(Identification below 40%)	Rate of Error (%)	Sample Size
1	0	0	5
2	2	20	10
3	3	20	15
4	4	20	20
5	4	16	25
6	6	20	30
7	8	22	35

Table 6.1: Error In Intent Identification Per Round

Thus, with a round rate of 8/35 in the highest round we could say that the model is acceptable for a beta or alpha release. It's also important to take into account that some of the intents provided in Table 6.2 contain more than one victim or attack vector and this might lead to errors within the entity identification. Moreover, we can see in Figure 6.1 that the standard deviation is around 10% which yield an acceptable value if we refer to a p value of 0.1. Thus, we can say that despite the fact that intent classification is not perfectly achieved, it is still relatively accurate since most intents are mapped within the same context. Whatsoever, it is important to take into detail that the elements in the array are not significantly spread and can give an accurate answer to expand this furthermore we could use this set of intents as additional elements within the validation as training phrases and this would significantly upper our result and most likely all further queries that contain a similar syntax. The intents to benefit the most are the ones with a accuracy rate below 0.4 since they would retrain the transfer model and give us a more significant result in the next iterations. This will be described in the further work section in detail.

Following up, for each set of conversations we received several results that yield from 0 to 8 errors or unidentified intents which in the client side could have a big downstream as the elements that are not directly mapped to an intent just yield an error result. In figure 6.2 we can see that the elements that are mapped to each query set produce a high yield of errors but this is relative as they are contained in the previous set. In reality, this sets are meant to match a unique amount of elements not necessarily attached to a group and thus is understandable that the rate varies. Despite this, we must say that an unlikely amount of errors would be around 30% of the set meaning that more than 11 errors would be a noticeable amount of errors for the tool. Whatsoever, the tool is not featuring learning from the context which can potentially increase the amount of accuracy for intent identification.

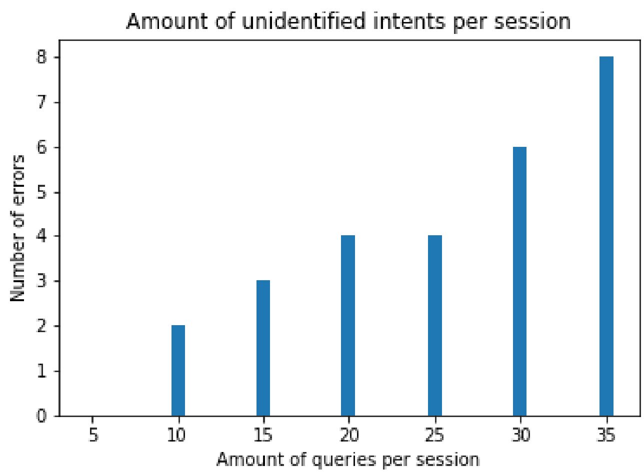

Figure 6.2: The Amount Of Errors Per Each Trial Session

To sum up, we can say that the tool can improve in the quantitative area by using the same queries as training phrases and enable the learning from querying to eventually have a close to perfect model. We do need a supervised training for this whatsoever but this is feasible since the amount of data gathered makes it possible. In order to have a more comprehensive model we must also understand the Case Studies which will give us an idea of the best case scenario for our tool and what to expect out of it, also further improvements can be done in this area.

Table 6.2: Queries Used For The Evaluation

Number	Example
1	'The load balancer graph shows a high increase in traffic'
2	'The webserver is handling a large amount of packets'
3	'The logs show that the router is receiving fragmented packets'
4	'Seems like there is a smurf atack going on against the memcache servers'
5	'my ingress controller is getting a lot of DNS responses'
6	'Im receiving DNS packets that the server did not request'
7	'i think im suffering a DNS amplification attack because the servers are getting many DNS answers'
8	'the server is getting packets that are too big'
9	'the load balancer is getting pings of death'
10	'my users are not connecting to the web server because of SYN dissosiation'
11	'my router is getting a huge amount of syn ack packets'
12	'the load balancer has low resources and a high amount of fragmented packets'
13	'my web server is getting UDP packets'
14	'the packets on the server are just UDP'
15	'the web server is getting an usual amount of 301'
16	'the nginx server gets a lot or redirects 301'
17	'the load balancer is prealocating too much memory'
18	'the web server is opening several connections at once'
19	'the load balancer is opening many connections'
20	'the load balancer is recieving a lot of small attacks'
21	'my server load is above average'
22	'i cannot access the web server and it is stuck due to high load'
23	'im getting NTP packets that i didnt request in the server entrypoint'
24	'Im reciving timestamps that were not requested in my server'
25	'the server log service shows an usual amount of udp requests'
26	'i have many open connections in the load balancer and a little bit of data transmited'
27	'I have too many connections right now and the web server is not responsive'
28	'im seeing a very high amount of requests per second in the load balancer'
29	'for some reason the web server is getting a lot of ICMP from other clients'
30	'the apache server has a very high amount of open concurrent connections'
31	'i see in the logs from the server that the packets are arriving with no information about the protocol'
32	'the database server is opening too many sessions'
33	'the load of the cluster is very high aparently due to many requests'
34	'the web server is delivering a lot of ssl certificates'
35	the web server is getting POST requests with a lot of data'

6.2 Case Studies

Previously, we defined a quantitative way to define how the model should behave and defined a preset of measures that would yield the desired result. In order to have a different approach to the same set of measurements we decided to come up with use cases that will expand and show how an ideal conversation would give the desired result and how a conversation could contain different elements yet providing a good result. To do this there are two possible routes for a conversation focused towards DDoS attacks that could be useful for the user and for MENTOR. The first set or Case Study will try to reassemble a conversation from top to bottom to generate a solution that could be fitted into mentor and provide a solution. The second case study will reassemble a close to reality conversation that is not necessarily made to fit MENTOR by rather any other kind of recommending system.

6.2.1 Case Study #1 - Requesting For A DDoS Protection

The first case study will try to generate a use-case scenario where a network administrator tries to solve a problem of a DDoS attack by providing a relatively limited amount of data and the chatbot would ask questions accordingly to obtain the most information in each case. Thus. in our first case study we will try to provide elements that can match MENTOR's [11] recommendation algorithm by gathering important elements such as Location, Type of Solution and Budget among other parameters that are needed to generate or classify an optimal solution for a recommending system. Therefore, we can see in Table 6.3 that the chat bot would query elements to the User and the user would need to answer this questions which certain level of detail. The intents also are described with the accuracy level for each case, thus we can see in the evaluation how close was the intent to the classifier's approximation. Also, is important to say that if an element is missing from the query set such as in the answer number 7 from table 6.3 we can see that no victim was provided and given a high accuracy rate the chatbot would proceed to ask a followup question to identify the missing entity within the context. At the end of this conversation a JSON file would be generated that would then be fed to MENTOR to generate a solution, the sample of the JSON file can be seen in Listing 6.1 where all the extracted elements are present.

The Generated JSON File can be used as an input of MENTOR but this will not be covered as the Proof of Concept does not require us to go past the generation of the file. As seen in the JSON file from listing 6.1 the structure reassembles a set of sub-array per each intent; thus allowing most recommending systems to parse based on the identified type and with the specific recommendation. In some particular cases such as currency or duration the sub-elements can also be described as such in arrays within the intent as composite entities.

Listing 6.1: Study Case 1 Generated JSON file


```
[
    [
        {
            "types_problem":"attack"
        },
        {
            "types_problem.original":"attack"
        }
    ],
    [
        {
            "ddos_attack_chars":"Fragmented Packet"
        },
        {
            "ddos_attack_chars.original":"partial requests"
        },
        {
            "victims":"load balancer"
        },
        {
            "victims.original":"load balancer"
        }
    ],
    [
        {
            "ddos_attack_chars":"Ping of Death"
        },
        {
            "ddos_attack_chars.original":"long pings"
        },
        {
            "victims":"web server"
        },
        {
            "victims.original":"web server"
        }
    ],
    [
        {
            "type_of_service":"proactive"
        },
        {
            "type_of_service.original":"passive"
        }
    ],
    [
        {
            "geo-country.original":"Switzerland"
        },
        {
            "geo-country":"Switzerland"
        }
    ],
    [
        {
            "duration.original":"1 day"
        },
        {
            "duration":{
                "unit":"day",
                "amount":1
            }
        }
    ],
    [
        {
            "duration.original":"3 months"
        }
    ],
    [
        {
            "unit-currency.original":"3000 swiss francs"
        }
    ],
]
```

Table 6.3: Case Study 1. The conversation between a user and the chatbot is described as it travels fro the agent from top to bottom. The user queries were entered in a sequential order and the results of the approximations are real and gathered from the evaluator of the conversation.

ChatBot	User
Hi, my name is NetworkBot! I can help you to solve your security issue. Try typing something like "I have a high server load." below.	
(1)	Hi
Hi! How are you doing?	
(0.803)	i think im under attack
Can you give me details about the attack	
(0.764)	the load balancer is getting fragmented packets
So we know that load balancer is having Fragmented Packet right, anything else to add?	
(0.578)	im getting a lot of pings of death
Can you tell me who was the target?	
(1)	the web server
the web server suffers a and Ping of Death, could you provide any other info?	
(0.781)	no, thats it for now
Do you need a solution for now or that will work overtime (passive or active)	
(0.809)	an active solution
where is the target located?	
(1)	in Switzerland
How much time do we have for a solution?	
(0.732)	within the next hour would be best
For how long should I make the contract?	
(0.486)	3 months is enough
What is your budget?	
(1)	4000 swiss francs
If that is it please type 'that's it for now' and we will prepare a solution	
(1)	that is it for now
Ok, sending to MENTOR to generate a solution.	

6.2.2 Case Study #2 - Generic Recommending System And DDoS Attack

Furthermore, in this case study we aim to provide a generalized solution to the identification of issues, thus we can allow the attack identification intent to gather extra information on each step so the generated JSON file could be fed to another classifier that could predict the solution based on a set of queries. This is done by repeating the process or gathering as much information possible from the attack identifier intent.

The idea behind the second case study is to find which is the easiest method to gather information from the user without having to preset limitations on normal constraints in contrast to case study 1 where the elements where extracted to fit MENTOR [11] with the benefits that would allow to gather more information and thus retrain the machine learning model to allow a constant expansion on the information obtained. Moreover, the model only obtains information extracted from intents; as an extension a heading or key can be added to detect the intent from which it comes from with the idea that it can be extracted from any tool by just reading the header of each element, similar to how frontend systems parse responses from backend systems.

Its important to say that despite the fact that the conversation was shorter, the JSON file can grow to several lines as the identification of the intent grows. The only way this can be limited is if the file is not mapping the right intents. As a current limitation if there is a disruption of the conversation and there is not fallback intent then the system restarts to the beginning.

Table 6.4: Case Study 2 Chat Demo

ChatBot	User
Hi, my name is NetworkBot! I can help you to solve your security issue. Try typing something like "I have a high server load." below.	
(1)	Hi
Hi! How are you doing?	
(1)	im suffering an attack
Can you give me details about the attack	
(0.790)	the web server is only getting partial requests
the web server suffers Fragmented Packet, could you provide any other info?anything else to add?	
(0.77)	also the database server is receiving a high number of request a lot of pings of death
the database server suffers High number of request, could you provide any other info?	
(0.704)	also, i have low resources
and what is the target?	
(0.714)	the main server
So we know that also is having Low Resources in the main server right, anything else to add?	
(0.656)	end here
Ok, Generating JSON File with the obtained values. Mentor won't be able to use them thought.	

As observed in table 6.4 each element maps to the input from the query on the right side. Below each response form the chatbot there is a number which denotes the confidence level of the intent classifier; meaning that in general the generic intent identifier has a higher success rate than the case study 1 which is tailored to fit a recommending system. This is expected since the chatbot was tweaked to fit the intent classifiers of common DDoS attacks rather than other sets of questions such as location or currency.

6.2.3 Analysis from evaluation

We can sum up the case studies by denoting that most of the intents have a relatively high amount of positives which denote a good level of accuracy for both models. It us important to contrast both in order to understand how a tailored solution can be made using the same model and just modifying the set of intents to be queried. In the case of the study 1 we aimed for a more explicit and templates set of answers which usually will reduce the amount of errors through the process. Meanwhile, the second case study presents a more general an usable use case but sacrifices the areas where data could be erroneously interpreted.

In order to get the best of both worlds in an ideal representation or production case we should aim to use both the generic model but tailored with modifications to fit the recommending system to avoid the chatbot generating errors or miss-interpreting the input. Needless to say that the chatbot classifier could also gather an input with a confidence interval higher than 40% and thus saving to the JSON file an element which

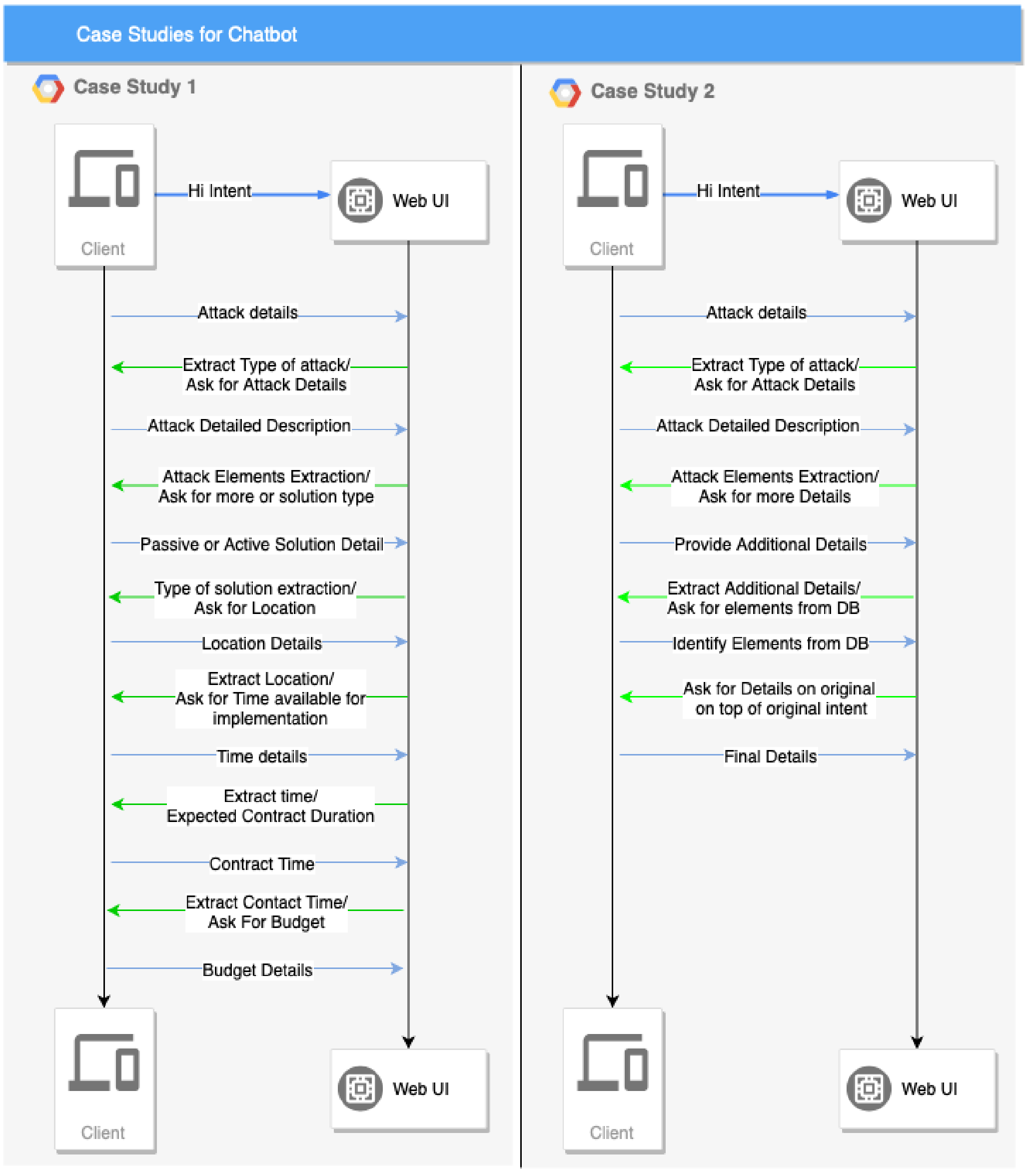

Figure 6.3: Case Studies Comparison Graph, Case study 1 From Top To Bottom On The Left Side, Case Study 2, Right On The Right Side.

might not be an accurate representation of the intent. For our first case study we used a bigger set of intents which is prone to more errors from one side but also easier to debug and to add fallback methods as the granularity of the solution increases in a controlled environment. On the other side, it is hard to predict the granularity based just on the input, the only way to guarantee this is by increasing the amount of training phrases to include high and low granulates or to decide the granularity by questions such as in the first case study.

Listing 6.2: Case Study 2, Generic Requirement Gathering

```
[
    [
        {
            "types_problem":"attack"
        },
        {
            "types_problem.original":"attack"
        }
    ],
    [
        {
            "victims":"load balancer"
        },
        {
            "victims.original":"load balancer"
        },
        {
            "ddos_attack_chars":"SYN Flood"
        },
        {
            "ddos_attack_chars.original":"SYN packets"
        }
    ],
    [
        {
            "ddos_attack_chars":"Low Resources"
        },
        {
            "ddos_attack_chars.original":"low resources"
        },
        {
            "victims":"web server"
        },
        {
            "victims.original":"web server"
        }
    ],
    [
        {
            "victims.original":"database"
        },
        {
            "ddos_attack_chars":"many NTP results"
        },
        {
            "ddos_attack_chars.original":"many NTP results"
        },
        {
            "victims":"database server"
        }
    ],
    [
        {
            "victims.original":"database"
        },
        {
            "ddos_attack_chars":"many NTP results"
        },
        {
            "ddos_attack_chars.original":"many NTP results"
        },
        {
            "victims":"database server"
        }
    ]
]
```

6.3 Discussion

Among very important areas discussed throughout the previous chapters, is relevant to tackle areas that could be improved and that are not necessarily defined as there is room for improvement or merely choice. This is the case starting from the beginning with areas such as NLP which is show to be beneficial compared to traditional methods of looking for a particular word in a context from a provided bag of words [22] despite this, we can not say that this would solve the problem of interpretation within the context; Moreover as shown by Duran [22] the are several limitations between traditional methods and NLP classifiers such as expressions proper to the language that could affect the way in which words can be recognized. For example a computer can be male or female in french and also could mean a different thing depending on the context, therefore traditional options are not always the answer.

Moving forward to another aspect which is the tool set for the Machine Learning algorithms to be trained into is a topic worth being discussed. As described in Chapter 3 the best choice for our chatbot was Google Dialog Flow in contrast to other tools such as Microsoft's LUIS or IBM's Watson, despite this there are interesting areas were we could presume that our assumption was not completely valid. Despite the fact that we can use DialogFlow composite entities, in practice this entities could have been used better in conjunction with a preset intent classifier. After gathering enough data the best way to continue the development of the tool would be to use a Supervised Embedding classifier instead of using a pre-trained model with transfer learning. This is detailed by Gardner [23] and in comparison the supervised model was rerun under a Recurrent Neural Network resulting in more accurate classifiers using less information.This is set to test and trial but is strictly related to the fact that this is not possible in DialogFlow and thus would have to be used in RASA which features an open implementation. This is also further expanded by the classifier and could be targeted using interactive learning from the RASA core which expands datasets using an algorithm similar to CBOW from Mikelov [16]. Also, related to the NER extraction we are limited to DialogFlow's pretrained extractor, for our base case this is fine as having a custom entity recognition component would be out of scope but is worth taking into account, whatsoever this will be expanded on the further work section. Is worth mentioning that if we wanted to keep the implementation open source there was the chance to implement composite entities using RASA from Sklearn and tokenizer pipeline, albeit this was out of scope as the implementation would have taken more time that the given for this project. Other options that could also be named is Wit.Ai which provides an API for automated intent recognition with the underlying technology of RASA but tweaked using pretrained models, similar to DialogFlow

Another important aspect that is worth analysing is the API being used from DialogFlow, which is at the moment the version number two but is prone to changes. This happened during the development of the the chatbot, were Google announced a change from the API V1 to V2 and the first version was being phased out. Despite this being a not necessarily disruptive behavior in order to maintain and have a long lasting solution we must opt to use a service that is not maintained/updated by third parties. This opens the lead to a few other cases for example when is the best time to swap technologies and for how long can a chatbot run in legacy mode; all these issues are related to having the server running at a external thus using an opensource alternative such as RASA might yield a better result if we aim to have a long lasting self maintained application. Likewise, if we want to have a reliable system which should be up to date periodically the preferred option should be DialogFlow. During the process of enabling authentication for V2, several issues rose such as the need for a server side application that could handle the authentication to avoid client side authentication directly with the DialogFlow API. To prevent this, we developed a client that only runs in the server side of the application when the application is served generating the tokens, whatsoever this implementation is not ideal as the authentication tokens had to be hard coded into the code and the login mechanism generates a token and a refresh token for each login. This is not an ideal situation and also prevents using security metrics such as requests per login.

Other aspect that is quite relevant for the development and further expansion of the chatbot is the granularity of the use. This is very important since the development of the chatbot goes strictly related to the usage and the target user of a chatbot. Lets take for example an Airplane company and its users who will most likely have simple questions related to flights and customer support, when a question becomes to complex a human will have to intervene. This is because of the granularity of the problem, different use cases have different levels of granularity. To put it in perspective for our chatbot we can take for example a simple issue with a very low granularity such as "my computer is slow" vs an issue with a quite high granularity such as "my web server is running nginx and the logs show that the handler has connections that are kept alive over 5000 ms thus depleting the resources of the node", given this two examples we can see that the difference in granularity plays a key role and despite the fact that we could ask the user to input the level of granularity to be used this could be more prejudicial than beneficial. Overtime, this issue can be overcome using a supervised learning model that takes this into account and with enough information is capable of creating a subset of intent classifiers. Whether this solution will be used in practice by more expert users is prone to evaluation since most advanced users expressed the lack of need to use external tools to solve common day issues as described by Botta [24].

During the Approach we stumbled upon many interesting aspects that can be found relevant was the usage of the extracted data. For our proof of concept the extracted data was not fully used as we decided to focus only in the DDoS section and most of the extracted information that lies within is subject to reprocessing. Within the chatbot code the functions for tool assessment and recommendation are available using the information obtained from the database, whatsoever for the DDoS attack intents this information is barely used as there is not much information about DDoS attacks within the UNIX manpage tool-set and thus if we would like to expand our chat bot to map recommendations related to unix manpage this is feasible by doing some changes to the existing model or by just extending the current model.

Among the development of the intents and the entity map its clear that there are methods that could provide a better result in terms of performance or at least result in a higher confidence interval, now the trade off would be to have additional information from a single source and then train expecting a higher confidence interval than with the established model. Now, this could have to results since we would be expanding both the intent and the entities using NLP rather than NLU so we could possibly have a higher amount of fake positives than we should have. Meaning that we would have a higher confidence interval but the result from this classifier would match to an intent that does not relate to the user need. Therefore, we are limited in this way by the amount of intents at least that can be created because this most be supervised instead of a traditional unsupervised learning, this is interesting because new methods of how to map embeddings in NLU are becoming a trend such as using LSTM as defined by Young [25] by combining a multiplier of inferences to have long term short memory instead of passing contexts through a pipeline. Is possible that this will become the default in the coming years but at the moment opensource chatbots that have a pretrained pipeline such as RASA use SVM as a defacto since RASA does not create the word embeddings.

Also, an interesting aspect regarding the process of extraction is related on the storage of the information. Since the base was a postgresql database we decided to stick to this one, but it seems like having everything in a NoSQL database such as DataStore is way better as we have no need of relationships and the insertions are rather rare in comparison to the readings. This was described in the paper by Padhy [26] where we can see that in some cases NoSQL Databases have a better performance when there are not too many columns and thus in our particular case once the index is built there is no need for a relational database. Moreover, the extraction of the data used the name, short name, synopsis and description. These elements are not standardized and thus not all of them can be used for example the description, is up to the usage and the project to implement this or not but in our case the description was no necessary. Following up on this, the structure definition was very close to our goal and the model seem to reassemble very close to common problems in the space of identifying key factors in the area of DDoS attacks, furthermore changes would be needed to adapt this to other use cases.

The overall section that features fulfilment describes a brief implementation of the system controller, sadly this is not a complete picture of how the control of the information flows. While using cloud functions might sound very appealing we are also bound to the vector and thus in order to avoid using this function we would have to jump to another function service provider such as AWS Lambda. Despite the fact that there are open source alternatives such as open faas, the implementation still remains a milestone to solve. Moreover, related to this is the fact that we trained our model only using positive training phrases that reassemble the expected intent, whatsoever we did not feed negative examples of what is not what the intent look like, this is because of a time constraint and to certain extent is not needed unless there are two intents which are quite similar and we need to dissociate one from the other.

The architecture is also prone to improvement and discussion as many of the elements provided in the architecture are highly bound to google cloud platform, for example the client authentication. If we wanted to escape vendor lock then the best way would be to avoid using google data store and dialogflow and replace them with RASA and a database such as MongoDB plus a open ML pipeline that can feature options such as automated training and management of data. This is again relevant to the project and the implementation. Once again, this is also related to the content that can be fed into the algorithms, if possible we would like to use a big amount of data but if this is not feasible then a pretrained model would be the best approximation.

The User interface works in a quite simple manner and would say that is quite successful, moving the authentication to a server side and reuse tokens is feasible but would raise questions regarding security. Even though this is a trade off, we can rework the login algorithm to perform the login operation faster. As for the responses, DialogFlow can feature a set of additional options that can be included and expanded on the model on the fly. Meaning that we could support voice to text or custom intents within the model to perform certain actions based on the input, for example we could open a browser with google to look for certain term after the bot would extract certain elements from the conversation that map to obvious solution, now we do not know if this would be beneficial for the user or a distraction, thus is prone to usage depending on the application. In our case this does not make sense as we want to gather as much information as possible from the user.

6.3.1 Threats to Validity

There are many areas where we have to limit the scope of the research and take into account that not all the changes were supposed to aim for a single objective which is having a functional working client that can solve issues related to DDoS attacks. In a general case this is not a problem as this solution can be expanded into other interesting areas such as tooling and problem resolution within the operating system. What so ever we must take into account some aspects that we assumed under ideal scenarios but would not scale in other scenarios.

Lets take for example the intent classifier which is currently looking for two key elements which is a symptom within the the attack that could be between a very simple issue with resources or a very complex problem that has to do with the frames within the packets. For our solution we assumed a median case scenario where the problem is properly described, in order to get more detailed information for the user a more detail model would need to be implemented by increasing the amount of intents. Even though this is not impossible, certainly would take much more time and will not have a big impact in the generic behavior of the chatbot.

Another limitation in our project is the intent classifiers themselves as for each we have to provide a set of training phrases that have to be specific for each kind of usage. If we try to take into account a different set of training models instead of a supervised trained model but rather a recurrent neural network that features reinforcement learning we could save time and let the algorithm retrain on each iteration. What so ever, we are limited in our case because DialogFlow is the owner of the code and to train in this particular way we would need to migrate the setup to an open source solution such as RASA or create custom datasets that could reassemble the training process using Google's natural language notebooks.

Additionally, the model takes into account predefined entities. Even though, we extracted over 300 entities form the manned DB a secondary manual inspection was also needed and this could raise issues if we decide to automate the process of reinsertion of the data. Worth mentioning that if we can reach an average confidence interval of 50% then we could potentially say that above 55% we can reinsert the data into the database and learn from this. Now this is a limitation of the dataset we picked. Alongside, the context can include a limited amount of information and the number of intents to which the context can be passed to is also limited and this is predefined by DialogFlow which specifies that for each intent we can only pass the element to the next 5 intents. This is a limitation that lies within LSTM (Long Short Term Memory) and the way the recurrent network spreads and retrains.

Finally, we have to assume that the model works using positive examples, as we did not include any negative phrases to avoid collisions in the model, meaning that is possible that there is not a major improvement using negative phrases and if there is we would need to include these negative phrases manually as they are only available in the model trainer but not in the UI that the user has access to. This continues since most of our limitations lie within the DialogFlow capabilities particularly in the NER extractor and the training of the intent classifier, in order to avoid this we would need to migrate the model to RASA.

Chapter 7

Future Work and Conclusion

7.1 Future Work

As future improvements to our chatbot we can take into account that most models are meant to deliver a particular function, in our case we aim to extract information and provide a representation of this information in an abstract way. In our current status there are many areas that can be improved to achieve a better model and to improve the confidence interval within the context. For example, Currently we use the confidence interval to retrieve the current information on the loss function, but we use negative phrases for this even though the model was trained without negative examples. We should focus in expanding our usage to include negative training phrases to increase the accuracy of the model. This is directly proportional to the amount of accuracy we want to reach, the more information we can provide to the model the better it will get.

Another interesting aspect in which we could improve the functionality of the bot is the way it interacts with the Client, at the moment we are only using the controller to pass data and do flow control. It could also handle elements that are integrated within the DialogFlow implementation itself such as providing localization, speech to text features and parsing of other information such as images and localized data. This is all integrated within the DialogFlow API and is simple to setup and use, we could potentially increase the usability of the chatbot by providing google searches or even images directly for identification and solutions.

Also, the amount of information that we are using for the NER is almost negligible compared to the total information that lies within the data store of DialogFlow. For this we could use the already existent parser written in python to extract elements such as the Synopsis and create another category of Entities based on this. In this way we could have entity identification for a broader set of tooling. Also, having a larger data set will reduce the probability of missing the identification of an entity and since the data set that we currently have is related to computer science or engineering the information is relevant and labeled.

Finally, we could improve the measurements in the evaluation section by lowering the confidence interval to 35% or even lower, this is because at the moment we evaluated with a model above 40% and within the identified data many intents were correctly identified meaning that we had false negatives reducing the real metrics of the model. We would need to test and trial a right number that would reflect the real accuracy of the model but this would also mean a better way to create intent phrases in our model. All these improvements can benefit the most by using an open source model that would allow us to tweak each element at it's core such as RASA NLU which can be beneficial in our case since it features all the current capabilities of DialogFlow but the pretrained model, which in case we manage to gather enough information would not be needed anymore and thus we could modify the source code and the intent classifier to use cutting edge algorithms that would most likely yield better results.

7.2 Conclusion

In our work we decided to take available information from different sources such as man pages and white papers to generate a database of content that could be fed into a Machine Learning algorithm that would allow us to parse queries from a user and predict if they match a certain intent predefined. In order to achieve this, we used a classifier that featured a loss function that would relate to a user query. The results show that this is a feasible task but requires a lot of information in order to obtain accurate results. Also, we defined a structure that could be relevant for recommender systems such as MENTOR [1] to perform the data extraction and have a generic abstraction of information from a certain user.

We can have several areas into which this can be expanded and are subject to evaluation. Our results point out that the best method to achieve this once enough information is available is to train a supervised classifier and enhance it with trending techniques such as reinforcement learning. This is a relevant area and has much potential, the client is capable of handling all requests due the architecture and furthermore we can integrate new features into the client in a relatively easy manner due to the early integration in the Google technology stack.

Finally, elements that are worth taking into account are the manual creation of intents that has to be mapped into the DialogFlowAlgorithm to retrain and that during our evaluation phase we see that despite conclusive results were obtained we would need to research more into how to optimize the loss function and reduce the standard deviation across the evaluation. Moreover, using independent intents for each round was shown to be positive by inferring the results of the evaluation. The Case Study was successful but should be expanded with other recommending systems in order to get the most out of it. This also will show other areas in which our project could be expanded to besides cyber security chatbots.

Bibliography

[1] M. Franco, B. Rodrigues, B. Stiller: MENTOR: The Design and Evaluation of a Protection Services Recommender System; 15th International Conference on Network and Service Management (CNSM 2019), Halifax, Canada, October 21-25, pp. 1-7.

[2] Gartner: Forecast: Information Security, Worldwide, 2016-2022, 2Q18 Update; August 2018, [On-line] https://www.gartner.com/en/newsroom/press-releases/2018-08-15-gartner-forecasts- worldwide-information-security-spending-to-exceed-124-billion-in-2019

[3] A. Jacobs, R. Pfitscher, R. Ferreira, L. Granville: Refining network intents for self-driving net- works; SIGCOMM Computer Communication Review, Vol. 48, No. 5, January 2019, pp. 55-63.

[4] Frost & Sullivan, Global Cloud Infrastructure as a Service Market Outlook, Forecast to 2023. March 2019; ID: 5761855.

[5] Bureau of Labor Statistics, U.S. Department of Labor, Occupational Outlook Handbook, Software Developers, on the Internet at https://www.bls.gov/ooh/computer-and-information-technology/software-developers.html (visited October 07, 2019).

[6] Rahman, Al and Mamun, Abdullah & Islam, Alma. (2017). Programming challenges of chatbot: Current and future prospective. 75-78. 10.1109/R10-HTC.2017.8288910.

[7] E. Cambria and B. White, "Jumping NLP Curves: A Review of Natural Language Processing Research [Review Article]," in IEEE Computational Intelligence Magazine, vol. 9, no. 2, pp. 48-57, May 2014. doi: 10.1109/MCI.2014.2307227

[8] Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., ... & Collins, M. (2016). Globally normalized transition-based neural networks. arXiv preprint arXiv:1603.06042.

[9] Braun, D., Hernandez-Mendez, A., Matthes, F., & Langen, M. (2017, August). Evaluating natural language understanding services for conversational question answering systems. In Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue (pp. 174-185).

[10] Rahman, A. M., Mamun, A. A., & Islam, A. (2017). Programming challenges of chatbot: Current and future prospective. 2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC). doi:10.1109/r10-htc.2017.8288910

[11] E. Sula.(2019, August).ProtecDDoS: A Recommender System for Distributed Denial-of-Service Protection Services.

[12] About Manned.org. (n.d.). Retrieved October 29, 2019, from https://manned.org/info/about.

[13] D. M. Ritchie, The Evolution of the UNIX Time-sharing System (AT&T Bell Laboratories Technical Journal, Vol. 63, No. 8, October 1984)

[14] C. Su, R. Gupta, S. Ananthakrishnan and S. Matsoukas, "A Re-Ranker Scheme For Integrating Large Scale NLU Models," 2018 IEEE Spoken Language Technology Workshop (SLT), Athens, Greece, 2018, pp. 670-676. doi: 10.1109/SLT.2018.8639519, URL: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8639519&isnumber=8639030`

[15] Wu, L., Fisch, A., Chopra, S., Adams, K., Bordes, A., Weston, J. 2017. StarSpace: Embed All The Things!. arXiv e-prints arXiv:1709.03856.

[16] Mikolov, T., Chen, K., Corrado, G., Dean, J. 2013. Efficient Estimation of Word Representations in Vector Space. arXiv e-prints arXiv:1301.3781.

[17] Strubell, E., Verga, P., Belanger, D., McCallum, A. 2017. Fast and Accurate Entity Recognition with Iterated Dilated Convolutions. arXiv e-prints arXiv:1702.02098.

[18] Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., Wu, Y. 2016. Exploring the Limits of Language Modeling. arXiv e-prints arXiv:1602.02410.

[19] Kim, Y.-B., Kim, D., Kumar, A., Sarikaya, R. 2018. Efficient Large-Scale Domain Classification with Personalized Attention. arXiv e-prints arXiv:1804.08065.

[20] Chen, J., Tang, X., Cheng, J., Wang, F., Xu, R. 2019. DDoS Attack Detection Method Based on Network Abnormal Behavior in Big Data Environment. arXiv e-prints arXiv:1903.11844.

[21] Zhao, H., Gallo, O., Frosio, I., Kautz, J. 2015. Loss Functions for Neural Networks for Image Processing. arXiv e-prints arXiv:1511.08861.

[22] DURAN, N., HALL, C., MCCARTHY, P., & MCNAMARA, D. (2010). The linguistic correlates of conversational deception: Comparing natural language processing technologies. Applied Psycholinguistics, 31(3), 439-462. doi:10.1017/S0142716410000068

[23] Gardner, M., and 8 colleagues 2018. AllenNLP: A Deep Semantic Natural Language Processing Platform. arXiv e-prints arXiv:1803.07640.

[24] Botta, D., Werlinger, R., et all. 2007. Towards understanding IT security professionals and their tools. In Proceedings of the 3rd symposium on Usable privacy and security (SOUPS 07). Association for Computing Machinery, New York, NY, USA, 100-111. DOI:https://doi.org/10.1145/1280680.1280693

[25] Young, T., Hazarika, D., Poria, S., Cambria, E. 2017. Recent Trends in Deep Learning Based Natural Language Processing. arXiv e-prints arXiv:1708.02709.

[26] Padhy, R. P., Patra, M. R., & Satapathy, S. C. (2011). RDBMS to NoSQL: reviewing some next-generation non-relational database's. International Journal of Advanced Engineering Science and Technologies, 11(1), 15-30.

[27] Green, G. M. (2012). Pragmatics and natural language understanding. Routledge.

[28] Zhong, V., Xiong, C., Socher, R. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. arXiv e-prints arXiv:1709.00103.

Abbreviations

NLU	Natural Language Understanding
NLP	Natural Language Processing
ML	Machine Learning
AI	Artificial Intelligence
CI	Continous Integration
CD	Continous Delivery
DB	Database
LSTM	Long Short Term Memory
SVM	Support Vector Machine
DDoS	Distributed Denial of Service
DoS	Denial of Service
API	Access Point Interface
V1	Version 1
V2	Version 2
NER	Named Entity Reference
PoD	Ping of Death
NFV	Network Functions Virtualization
SDN	Software Defined Network

List of Figures

List of Tables

Appendix A

Installation Guidelines

As a start point we must setup the chatbot Dialogs and Intents using DialogFlow, we must have an account for this and using the web client of dialogflow import the chatbot from the zip file provided as chatbot.zip. Once the chatbot is imported we need to setup the web server.

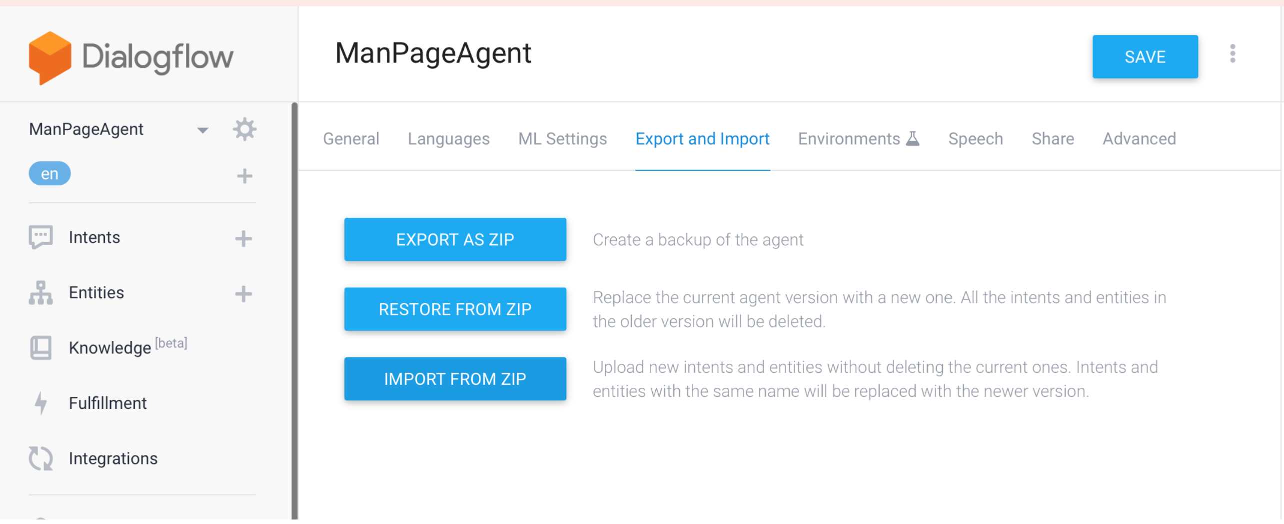

Figure A.1: DialogFlow import chatbot screen

In order to have a functional replica of the chatbot we must start a web server containing the client. The client is a react app and thus we can start it by setting up the authorization within the application.

The right parameters for the project must be setup on the client side. Also, we must configure the authentication tokens for the owner of the project given by google as a public and private key described in figure A.4. Moreover, once web client is working we can test the web client to see if we have a connection, by default the logged is enabled and errors should show at any time.

If there is a working connection with the client we can pass to run the evaluator code in python. To do this we have to install a jupyter interpreter locally and then run the jupyter notebook as a whole with all the blocks.

```
class App extends React.Component {

    constructor(props) {
        super(props);
        const extractedEntities = [];
        const accessToken = '';
        const id = uuid();
        const projectId = 'stone-column-256215';
        const serviceAccountEmail = 'dialogflow-mypfjy@stone-column-256215.iam.gserviceaccount.com';
```

Figure A.2: Settings relevant to the DialogFlow Project

```
async getCredentials(){
    const claimSet = new ClaimSet(["https://www.googleapis.com/auth/cloud-platform"], 'dialogflow-mypfjy@stone-column-256215.iam.gserviceaccount.com');
    const accessToken = await GetAccessToken(claimSet,'-----BEGIN PRIVATE KEY-----\n' +
        'MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQCOEpaPs9EwUue2\n' +
        'Po4afjC74mszyo6qN56NxKMQ23+2LCxkkirDLHbYLf1OVN5vGRsJmiy9uurm2dM2\n' +
        'o6BDThIdryLJv3rZMq5FDlFqrTz1GRBCmDj8tQiegoc9qdlKR3lX3271eMsF4pkx\n' +
        'q6gipli2+PyJgWmh9jDa3QWru2ZSIiBQ8ob7+QH+Gazqp2TQkdgMl5sD5gdlcX/7\n' +
        'JSCVkugNKnG5ueN6YfjVTYB4mQzv708QeDDyxxIjgzr1mrFwtQHUN/pDsoq8UcRS\n' +
        'wD1OcKx+8bJT5AoHpYKemGBcEvS8+y45cd4j2xUJfGlkpmWZY4GLTzb+igHx5Gs8\n' +
        'fscGv2xRAgMBAAECggEABA4yR4xXvJQlQp3DAy0WeYqDSmFuFJxNVZknGXhWiy30\n' +
        'H3PPri4S/2VCrY19Ivk5vRquS8Ut5uSRY6YkkT9Fg5zn9o/VcyU/9b8sokmrK4jp\n' +
        'XqlHOSl12W56607xoa3sIA7D9VRbIUIgYttWS5Tpg+GUHd24lPO43cqA2q/ZAh+l\n' +
        '3MKvvEQdwk2mnvYPK5mNu/ZLvKdsYzw1dflRyvXjqbeWmwtOZHnYckafySxl/x8h\n' +
        '/WMmhRoecYXwRBYFHSaMDM/2tlhwJQnFbX7uiwHBeXRqaumf276q9q+M2KNK+cCG\n' +
        '9D+QlVwYmW5U+jGPlQGCbg6ITBSQALYW5d2IPPE3gQKBgQDGAVuFdWHRK7DK+0W2\n' +
        'RSCuEqnm8IFXMzVzy9bHcyyVKC7zbeSamrLPPs4IFoL48VrBLAsG7xGBjU6bgXg3\n' +
        'QrhfsYU54C6w+gjHiC2z1XJECWsWN/MjUNf4hHlTufUJOBjbmdlT4lsNni5xpALq\n' +
        'F0vM6J7GsX5Iia/myS8hN5eR8QKBgQC3r1blNKRBfl2r+B1pwlHiO+ILY4enwR7F\n' +
        'PZoffCTlTW2ToCzOBKI1CCvIMMwgCK4C/TXKFjuHQw4Yl0xtOX5vqNZT1Fss3EM+\n' +
        'pRi07ch1xcPu44WCqkcy8+flpC9Imcla1J7BWH/dD/iORuB6GK0lwwYN5SHhq4UA\n' +
        'KcRt8kMgYQKBgCwPK0VULFL2DN3t8cIwOahiYMoRpEY6D9/XcNBq0XXTPjdOsbqx\n' +
        'R66UBBaKo9QvKImpej6tvdB9GAvEUHLDdvFcOA/70Yfx6zgzFjBZA5ceHWURSBoI\n' +
        'qDYVjEXUMDE/+VAMOO7QhIBQPXduDnP2gRh0frqp0LblZG3PQJx8mU6RAoGBAJhb\n' +
        'CDFPS5rDTMDyKLtrIW4MC380y/U5+uLRKMBKwXOsmGHlbqRnNX/VSjNfPkCn0mXK\n' +
        'Ts5TSIb1+IAvJoZGNPg6y6N2tTVtcR1J5UdVH3uIUO8ZqsjpoerCYiuY1ltag6f6\n' +
        'Sb/HPy9GKv0gDttXEm8K5z+pOUQsd0HeYv5q/oChAoGBAKFOHwIcuEezQkZnze0y\n' +
        'bpwGhCXaMJ6aDAMxXq/MpXvQ+3zDZoJll14Fl8y+tUJjmDcr0KBXyWucXYOuQ6wx\n' +
        'JTiqY0bQlnLEJkQGMkJhcbWFX6Ek3x8R68drOo9tUC87zV5WZVCYaaLNcObffyxS\n' +
        'Nr2CPZVR3Uxv+xnj4LeHStlX\n' +
        '-----END PRIVATE KEY-----\n');
    this.state.accessToken = accessToken;
```

Figure A.3: Authentication Credentials

We must go to the evaluator.ipynb and then in the headers we will find the project and a token ID which can be generated using google auth for a one time usage or using google credential systems with a new user for the project. Once this two parameters are fed, we can pass to run all the blocks at once, the results should be similar as the ones obtained in the evaluation section.

```
projectName = 'stone-column-256215'
authToken = 'ya29.Il-7B3993ZkkQnzql-pjiPQbdqQYfFIPR0TOWAHzbNMRIFwkz2ckk2aXwMBqpgcYXSzi0jDBOoT4Pmp7UZBs3rUOokfQSv4B8iX9g3ep1lYexhNjjpAIYhjlhqJ7Xn6q
```

Figure A.4: Credentials for the evaluator.

The authentication token must match the one from the user of the DialogFlow chatbot and in that case should yield the results and generate a graph towards the end of the execution.

The extraction of the man pages is also possible using DataGrip and SQL queries, but in order to have this setup you must download the complete database schema from manned DB in the url `http://dl.manned.org/dumps/` and then use the database sql queries to match the extraction into a new database. In our case we created a PostgreSQL Database in Google Cloud Platform, but this can be replicated in any PostgreSQL database.

Appendix B

Contents of the CD

- extractor.zip (Source Code of the Database extractor Jupyter Notebooks)
- evaluator.zip (Source Code of the evaluator notebook)
- chatbot.zip (Source Code of the chatbot)
- webclient.zip(Source Code of the webclient in Javascript)
- MasterThesisSource(Latex Code including graphs of thesis)
- MasterThesis.pdf (The printout of the thesis)
- FinalPresentation.pdf (The final presentation in PDF format)