Customer Service and Ticketing Chatbot: From Conception to Production

Gonçalo Lascasas Cereja Ribeiro Pereira
goncalo.ribeiro.pereira@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2019

Abstract

Chatbots, also called conversational agents, are software systems which can interact with humans using natural language, through text or voice interfaces. This dissertation presents a complete investigation regarding the current chatbot technologies. User messages are analysed based on supervised training instead of a rule-based approach, allowing conversations to be more fluid, and even the ability to improve over time.

It is possible to develop a chatbot using a platform in a matter of hours, although, each architecture brings limitations, which will be explained in state of the art. With the help of Link Consulting I could design, prototype and develop a customer support chatbot with ticketing capabilities for the Portuguese railway company, CP. After its implementation, CP Bot was evaluated both automatically and through user testing, gathering effectiveness, efficiency and satisfaction metrics. Some good practices along with a proposed chatbot life cycle are then presented.

Keywords: Chatbot, Natural Language, Conversation Design, Customer Service, Ticketing

1. Introduction

Ever since the rise of the internet, it has influenced the way humans interact with each other. It allows us to reach out a much greater number of people on a global scale. Starting two decades ago, social media has played an important role in changing the way people communicate[10]. Since 2015, the number of monthly active users of the four largest messaging apps surpassed the four largest social networks[6]. Users are rapidly turning to social media to request and receive customer service. However, a majority of these requests can’t be addressed timely or not even addressed at all.

Chatbots are software agents in which users can request data or services using natural language interfaces, text and voice. They are typically designed and developed for integration with mobile messaging applications, and represent a potential shift in how people interact with data and online services. Users want access to personalised information on demand, preferably 24/7, and although chatbots are not a new technology, the increase in popularity has strengthened due to the recent advancements in AI. Natural Language Processing is the AI sub-category that allows chatbots to "learn with experience" instead of matching user input with expected patterns. Chatbot adoption is also increasing with the launch of developer API by most popular messaging platforms such as Facebook Messenger, Kik, Slack, Skype, Line, and Telegram.

1.1. Objectives

Many building alternatives are at our disposal, but creating a complex chatbot is not easy, because each architecture has its own limitations. Implementing a customer support chatbot with ticketing capabilities introduces technical complexities that prevent the use of conventional architectures. To provide dynamic data, such as ticket availability, latency of responses should also be taken into account. Facebook Messenger and Slack have timeout limits of 20 and 3 seconds, respectively, and even platforms such as Dialogflow have similar policies, in this case 5 seconds.

My main objectives with this thesis are:

- Understand how well chatbots integrate with existing systems.
- Implement a goal-oriented chatbot that understands its users using NLP.
- Evaluate implemented chatbot through effectiveness, efficiency and satisfaction metrics.
Compare intent and entity detection rates for EN, pt-PT and pt-BR agents in Dialogflow.

Understand advantages and limitations of this technology.

Produce a list of guidelines and good practices for chatbot’s designers and developers.

2. Background

2.1. Types of chatbots

Chatbots can be divided into different types depending on their purpose and the way responses are generated, as demonstrated in Figure 1[8]. Closed domain chatbots are easier to develop because they have a somewhat limited space of possible inputs and outputs. Customer support or shopping assistants are typical examples, and their purpose is to help users get useful information or achieve other goals. Users can still ask anything, but the system isn’t required to handle all cases and the users don’t expect it to.

Open domain bots have existed since famous Eliza was introduced to participate in Loebner’s contest. They are mainly for entertainment purposes and focus on giving smart responses to maintain the conversation. The infinite number of topics and the fact that a certain amount of world knowledge is required to create reasonable responses makes this a hard problem. One way to assess them is by measuring average length of conversations, and obviously, if conversations are short means that the chatbot is not entertaining enough.

A retrieval-based chatbot uses a repository of predefined responses and some kind of parameter to pick the most appropriate response based on the message and context. The parameter could be simple, as in a rule-based expression match, or complex when using Machine Learning classifiers. These models don’t generate any new text, they just pick a response from all available templates.

Chatbots with generative-based responses are harder to create because they need to generate new responses from scratch. These models are typically based on Machine Translation techniques, but instead of translating from one language to another, they “translate” from an input to an output set[1].

2.2. Chatbot Architecture

Chatbot’s conversation is an endless cycle of receiving user’s utterances, then understand its meaning, and finally produce the most appropriate response. A chatbot is composed by a frontend and a backend connected by webhooks calls. Calling a webhook differs from calling an API by being possible to send data without being requested, improving system’s efficiency.

Frontend clients, or channels, are gateways for clients to access chatbots, and it is possible to reach bigger audiences by integrating with multiple frontends, usually IM platforms, but also possible through apps and websites. The backend of a chatbot is very similar to a typical Dialog System with natural language understanding, dialog management, natural language generation[5]. It needs to analyse and structure received messages, and identify user’s intent to generate the best response possible. To do so, it might require accessing databases or calling external APIs. Chatbot’s typical architecture is present in Figure 2. Although not included below, chatbots might have ASR and TTS modules to interact in spoken language.

2.2.1 Natural Language Understanding

NLU is responsible for extracting semantic information and meaning from natural language in order to create grammatical data structures that can be processed by the Dialogue Management unit in the next step[2]. NLU must be trained wisely and with a multitude of examples to correctly identify message’s intent and entities.

If an user asks “Can you please show me the trains tomorrow from Lisbon to Porto?”, the NLU should identify that the user’s intent is to get the schedule, but also identify the origin, destination and date entities that were mentioned. The context should also be updated in order to recall what is the current conversation topic, either directly with a NLU engine or locally in Dialog Management. NLP can be considered as a conjugation of NLU and NLG, and the three main concepts in NLU are explained in detail below:

- **Entities** represent concepts that are often specific to a domain as a way of mapping
phrases to approved phrases that catch their meaning.

- **Intents** represents a mapping between what a user says and what action should be taken.
- **Context** are strings that represent the current context of the user expression. This is useful for differentiating phrases which might be ambiguous and have different meaning depending on what was spoken previously.

### 2.2.2 Dialog Management

Also known as action planner, this module must maintain conversation context and plan the right sequence of service actions for incoming messages. Depending on user intention it must decide if it is necessary to retrieve additional information to generate a proper response, such as accessing an database or call an API. Finally, it either forwards a final response to NLG or the structured information required to generate it.

### 2.2.3 External API Calls

Integration with external services is useful when users request specific information or tries to invoke operations that cannot be processed locally. These include calls to databases stored in the cloud, typical APIs to retrieve information, and even payment gateways like Paypal. There is no limit for external calls but they should be performed quickly because the user is waiting and some platforms can even block replies after a certain period of time.

### 2.2.4 Natural Language Generation

NLG receives as input a structured representation of information required to generate a response to the user. This conveys information about who is speaking, the dialogue history, and the context[5]. For chatbots available in many channels, this module must take into account which richer message elements are available in each platform. This module should implement the desired personality and be consistent in its answers. Additionally, it can even implement multiple personalities to interact with different users.

### 3. Implementation

#### 3.1. Analysis

The goal of this project was to implement a customer support chatbot for CP clients, aiming to improve user experience when interacting with the company. Its target users are mostly Portuguese customers but it also have an English agent aimed at tourist passengers, which usually are the ones that take longer to buy tickets in stations. According to a study from Marketest, 95,3% uses Facebook, 74,2% uses WhatsApp, followed by 70,8% on FB Messenger and 67,9% on Instagram from a total of 809 respondents[9]. WhatsApp didn’t have an API for developers at the time, and as a result, Facebook Messenger was the primary channel chosen for this chatbot.

Its main functionality is providing online booking of trips with the motivation of reducing queues in stations, both machines and office desks, and at the same time help the customers instantly and wherever they are. This would be specially useful in festive occasions with bigger affluence of people, as well as in crowded stations which recurrently make clients miss their trains in waiting lines because current booking alternatives are not efficient enough. Selling CP tickets online is tricky, as some urban stations have physical gates which currently only open with a valid VIVA transportation card. That is the main reason why only long-distance tickets are for sale, in both CP website and mobile application, avoiding this problem.

To overcome this limitation, my proposal will simulate tickets for urban trains, that are delivered using a system of vouchers. When the payment succeeds a unique voucher associated with the purchase get generated, such as a QR Code, and the user can exchange it for a filled-up VIVA card in the stations. This voucher process was used by Metropolitano de Lisboa to sell transportation tickets in the Web Summit 2018 week, which allowed clients, including me, to exchange a 16 digit number for the ticket through voucher-only machines.

To increase chatbot’s value it will support some intents other than providing timetables with respective booking options. These include the ability to find nearby stations, show active alerts or promotions and allow users to subscribe new alerts of multiple service categories. The bot will also be trained to provide useful information that is present in their FAQ (Lost and Found, Luggage transport, and more), as well as alternative ways to contact CP, which may be useful for users with complex requests that are not automated.

The requirements selected to implement in this project were

- Provide schedules and ticket selling
- Show next arrivals/departures on CP stations
- Automatically respond to FAQ
- Show active alerts
- Send alerts based on user’s preferences
- Provide help for reduced mobility customers
3.2. Architecture
I decided to build a stateless architecture running on google cloud, to take advantage of the elasticity and distributed nature provided by modern cloud computing platforms. In other words, it is based on micro-services that don’t store client data and can easily scale up or down. Also, Dialogflow was the NLU platform chosen for intent detection and entity extraction, although, the user context is also stored in the database to avoid session expiration.

Figure 3 represents the current architecture behind CP Bot, and the sequence of numbers represent the order of steps that might be required to generate the select the response. Sending the message to Dialogflow can be skipped if the user clicks on buttons or quick reply. The same happens for calling External API whenever the necessary information to respond is present locally. Also, users can receive proactive messages depending on their subscription preferences.

3.2.1 Channels
Chatbots interact with users on channels, usually IM platforms but can also be accessible inside websites, mobile applications or even Twitter direct messages. There is no standard among these social networks for prioritisation, routing, or error handling, and different channels are popular in different regions [3]. Currently, CP bot is only available through Messenger but might be available through other channels if needed.

3.2.2 Dialog Manager
is responsible to handle every user message and even though using Microsoft Bot Framework would ease the process of connecting to multiple channels, would require and extra module in the architecture. The second option was to use Botkit Development Framework hosted by Google, because Dialogflow is the chosen NLP, ensuring reduced latency times and avoiding possible incompatibilities of using multiple service providers. In the end, I found that a Botkit based solution would require the server to be always running even without requests for long periods of time. To overcome this, the final version of the Dialog Manager became stateless application hosted in Google Cloud Functions which can scale up when needed but also saves resources by sleeping when there are no requests.

3.2.3 Natural Language Understanding
is a mandatory module for this chatbot because of all the advantages that were presented in previous sections. At the moment, Dialogflow and Wit.ai are the only NLU engines that support Portuguese (pt-PT) language. Dialogflow was my final selection mainly because it allows to create multilingual agents and to import/export training data which will be very useful if the NLU engine needs to be exchanged in the future. At first, I found that pt-PT support was quite limited, by not recognising misspellings, or ignoring parts of the input that are irrelevant to identify the intent. During 2019, they introduced a spell checking which works quite good for small syntax errors, both grammatical and misspelled entities.

3.2.4 External API Calls
The chatbot require some external integration to obtain real-time information. Through them I can provide schedules, live departures times, get directions to stations, and even connect to online payment service providers. CP API is used to obtain requested schedules, and may soon allow real ticket emission. Currently is sends user’s origin, destination, and date of travel to receive respective schedule. IP API is required to access live departure times of trains in stations. Paypal has more than 250 million users from all over the world[12], and can also accept payments from guest users which pay with credit cards. QuickChart was only integrated while writing this thesis, and came from a necessity to analyse data from testing users.

3.2.5 CP
CP representatives might want to notify their customers about some relevant topics. These notifi-
cations could be suppression or reinforcement of train services, promotions or other any other relevant alert regarding their trains services. Through an API endpoint they can integrate with the Notification Manager to send Portuguese and English versions of alerts to subscribing users.

3.2.6 Database

The database being used in this project is Cloud Datastore, Google's NoSQL database for web and mobile applications. It automatically handles sharding and replication, providing an highly available and durable database that scales automatically allowing to maintain high performance as traffic increases. It integrates via a RESTful interface and provides ACID transactions, SQL-like queries, indexes and more[4]. This database is responsible of maintaining which alerts are published, which fallback messages are accumulated for NLU training, and the following attributes for each user:

- Name
- Conversation contexts
- Favourite stations
- Topics subscribed
- Agent (Portuguese or English)
- Start and finish times for each task executed
- Fallback counter
- Last schedule searched

Additionally, it saves feedback submitted by users, which was shown as a button for those who performed in user testing. A new functionality of automated testings, still in development, will be based on collected conversation examples. Each functionality implemented will have a document with user messages and their respective answers, avoiding manually testing them.

3.2.7 Notification Manager

This module is an API endpoint responsible for sending proactive messages to users. When a CP representative adds a new alert, this is responsible to forward it to users depending on their subscriptions. This module can be also used to send follow-up messages to users that stopped interacting with the bot. To create a new alerts the request must include both Portuguese and English versions of to accommodate every user.

3.2.8 Natural Language Generation

This retrieval-based module is populated with templates of responses for all supported intents. Choosing the answer from a finite list it is possible to guarantee the messages won’t have grammatical mistakes and even allow to create a sense of personality. In case users interact with the chatbot through different channels, this module is responsible to generate the best reply according to each platform limitations. To illustrate this, a Slack user might receive the schedule of trains in a drop down menu while those interacting using Messenger receive them through quick replies.

3.3. Design and Prototype

I started by drawing low-fidelity expected conversation flows and then tried to convert them to a digital version, but without prior design experience, found it too much time consuming. These flows should be constantly updated to match new bot functionalities and updated response messages, although most of them remained in my notebook and served just as as guide to the development phase. To build the conversation flows the following elements were used:

- User written messages
- Chatbot messages
- Payload messages
- Flow elements
- Backend integration

Then, I decided to design the conversations using Chatfuel platform providing a functional chatbot in which beta users could interact. Implementing the conversation flows in this prototype was complex due to platform limitations, although I managed to do so with over 75 blocks for each language.

The main purpose of the prototype was to detect possible flaws before the actual development, and see how the NLU was performing. Implementing an automated user testing functionality was really difficult, leading to 45 additional blocks only for English users, and in the end it was not completely functional. I found it was too much functionality for the platform capabilities, and I had to find an alternative to evaluate it.

To convince people to use this prototype it needed to provide live schedules. Using user attributes to save entities, logic operators and API calls to a backend server in Glitch, I ended up with a dynamic prototype that provided real time departure times and schedule requests. Also, I started redirecting every user message, apart from button clicks, to Glitch server which was responsible to
call Dialogflow, and based on the detected intent would generate appropriate responses.

With all of this, I asked friends and family to test it and every time a message intent was not recognised by Dialogflow it was being saved in a spreadsheet for further training. Additionally, I had to see every conversation, manually, looking for errors and mishandled intents. In total, 53 users used originated more than 1500 unhandled messages accumulated.

This systems should be easy to use, both by beginners and advanced users. Those who are not used to interact with chatbots, can choose the most common functionalities from a menu, and then respond accordingly to the scripted conversation. To achieve their goals faster, users directly send a detailed request and skip some steps of the conversation. In some replies the bot should provide some buttons to guide the the user into possible conversation paths. These buttons represent pay-loads and can improve bot efficiency, by replacing written messages with a single tap. Figure 4 shows how a guided conversation compares to handling complex messages.

![Guided Conversation vs Complex Messages](image)

**Figure 4:** Examples of chatbot interactions

### 3.4. Development

While the prototype was gathering user interactions, all my effort went to build the final version of the bot using Cloud Functions. Each requirement, or functionality, consisted on multiple iterations of training possible user intents in NLU, building respective code which supports and followed by tests to ensure it works as expected.

#### 3.4.1 Dialogflow Training

Intents usually rely on entities to correctly understand the messages, for example, it would be irrelevant to know user wants a schedule if it doesn’t recognise the stations. With this in mind I started the following entities:

- Alerts: With a row for each alert topics and respective synonyms
- Stations: Over 450 rows of various CP stations
- FAQ-categories: Supported FAQ categories and some synonyms

Intents were then trained to look for these entities, although soon I realised that if the user started the phrase with something like ‘Can you give me’ or ‘I would like’ it would lead to the fallback intent. To solve these I created some helping entities which allow a single example intent phrase to cover multiple combinations of requests.

Similar, I also had to create a new entities with possible variations of articles, mainly to be used in before station names. Dialogflow provides built in date entities which are very useful although I didn’t want to multiply every training phrase by the different types of dates, and ended up with a single ‘DateVariation’ intent with all of them combined which I can handle differently in the code. In Portuguese agent this allowed to identify abbreviations of dates based on an additional date entity.

To support some basic small talk, I added the following intents as well:

- Cancel - The bot will forget the forget the conversation context
- Emojis - It will recognise over 1500 emojis and avoid a fallback response
- Feedback - Asks user feedback and then saves it in the database for analysis
- Help - Presents the menu and asks if the user need help in any of the supported topics
- Human Handoff - Gives CP contact alternatives and alerts the admin, me, of which user is asking for a live agent
- No - replies with ‘Sure, feel free to ask anything else related with CP services’
- Thanks - responds with ‘Always here to help!’

Dialogflow has a timeout for user’s context after several minutes, and when users interact using buttons dialog is not called. There are complex context scenarios, for example, when the bot asks what is user’s destination, and then he clicks to add a favorite station. At this moment Dialogflow must only recognise his station while dialog manager handles the context. in which the bot must ask the user which station he is referring, then add as favorite and save as destination. To accomplish this and other context-related handlers four new intents were added in DialogFlow: ‘Inputs-AlertType’, ‘Inputs-Station’, ‘Inputs-Date’ and ‘Inputs-Number’. 
3.4.2 Build and Test

To implement the CP Bot architecture the first step was connecting the backend to the messenger channel. This required creating a Facebook page and creating an App in Facebook Developers dashboard and request permission for pages.messaging. In a few days it got verified and I was able to receive webhook notifications in my cloud function. Also, they provide all the security measure to make sure the messages are authentic. With all this setup the backend started by receiving only text messages and button payloads, but in the end it was able to handle images, voice recordings, and user locations as well.

Integrating with Datastore was straightforward because it also belongs to Google Cloud Platform. Then, it got integrated with Dialogflow and I started developing one functionality at the time, testing it manually before moving to the next one. It was not possible to simulate function’s execution locally, which required deploying, and waiting 2 minutes to see it in action. At the moment of this writing, I found that major drawback was fixed when Google, in April 2019, released a new opened sourced Functions Framework for Node.js[11].

A new permission had to be requested in order to implement Notification Manager because, by default, chatbots in Messenger can only send proactive messages to users 24 hours after their last interaction. Apart from the expected functionalities it ended up giving Google Maps directions to near stations, allowing user to save favorite stations, show what personal information is being stored, why it is being kept, and the possibility to delete it.

In May, the agent was robust handling most conversations but I started noticing that most of my friends and family would repeat past requests to obtain information. To obtain new testers, I decided to test a ‘click-to-Messenger’ campaign as a mean to gather more users while I was developing the required user testing and questionnaire functionality. This ad was a screen recording inside CP Bot requesting a schedule and receiving it in a few seconds. Over 9 days, it reached more than 5 thousand people, 1,255 of them engaged with the post and 69 started conversations.

When CP Bot was several hours without activity, the following message would take around 7 seconds to respond, and even more if it was a schedule request. To reduce these cold start times, I found that the main factor was the number of dependencies, and ensuring multiple functions use the same version of dependencies also helped because they might be retrieved from a cache. An additional measure was to use lazy-variables, allowing parts of the code to only be loaded for specific requests. Also, after 859 deployments in ‘us-central1’ region, I found that ‘europe-west2’ was the one which provided better executions times. Table 1 compares the times of executions, for warm starts, before these improvement with the latest version, 340th deployment in ‘europe-west2’. The execution time reduced, in average, 44.71% and the real was calculated manually with a stopwatch, from the time I pressed the send button until the last expected response appeared on messenger.

4. Results

Ever since the first chatbot was developed, creators aim at measuring how well their chatbot is performing in order to improve it. There are different perspectives on how to evaluate the quality of chatbots. I will evaluate this proposal in line with Jongerious research, with metrics that can be gathered either automatically or through questionnaire-based methods[7]. He distinguishes questionnaire-based methods into:

- Expert Review, performed by chatbot experts, to capture bot’s performance based on Effectiveness, Efficiency and Satisfaction with which specified users achieve specified goals.
- User’s Opinion, a well-known approach to get insights about bot’s perceived performance, gathered by asking users for their opinion. The results of a user evaluation might not reflect the actual chatbot’s performance but instead reflect the perceived performance of the chatbot.

I tried to evaluate CP Bot in an automated and scalable way, based on Automated Metrics to measure its efficiency, and User Tests to obtain by perceived performance metrics, related to QoE. Taking into account CP Bot is a closed-domain chatbot, Table 2 presents the most relevant quality attributes which were selected to evaluate with users, divided by four categories: Functionality, Humanity, Affect and Accessibility. Ethics should also be included to see how user feel about protecting and respecting privacy when data protection is fully implemented. At the moment, GDPR functionalities allow users to delete their name, favorite stations or the last searched schedule, although does not allow the deletion of past conversations. An additional Ethics should be included in future user test when GDPR functionality is fully implemented.

4.1. Automatic Metrics

In this project I was able to gather metrics from three different sources, which I will describe below. Dialogflow has a smaller number of messages because clicks on buttons don’t require NLU intervention.
Table 1: Chatbot execution times

<table>
<thead>
<tr>
<th>Intent / Payload</th>
<th>Entities</th>
<th>us-central1</th>
<th>europe-west2</th>
<th>Real duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetStarted</td>
<td></td>
<td>1.16s</td>
<td>0.52s</td>
<td>3.04s</td>
</tr>
<tr>
<td>Menu</td>
<td></td>
<td>2.05s</td>
<td>0.58s</td>
<td>3.82s</td>
</tr>
<tr>
<td>Departures</td>
<td></td>
<td>1.82s</td>
<td>0.60s</td>
<td>2.25s</td>
</tr>
<tr>
<td>Inputs Station</td>
<td>station</td>
<td>2.90s</td>
<td>1.60s</td>
<td>3.92s</td>
</tr>
<tr>
<td>Near Stations</td>
<td>coordinates</td>
<td>1.58s</td>
<td>2.74s</td>
<td>0.61s</td>
</tr>
<tr>
<td>Send Location</td>
<td>origin,dest</td>
<td>2.02s</td>
<td>0.83s</td>
<td>3.40s</td>
</tr>
<tr>
<td>Schedule</td>
<td>origin,dest,</td>
<td>4.34s</td>
<td>3.34s</td>
<td>4.43s</td>
</tr>
<tr>
<td>View Stops</td>
<td></td>
<td>2.11s</td>
<td>0.92s</td>
<td>3.03s</td>
</tr>
<tr>
<td>Change Schedule</td>
<td>date</td>
<td>4.58s</td>
<td>2.26s</td>
<td>3.69s</td>
</tr>
<tr>
<td>Thanks</td>
<td></td>
<td>1.89s</td>
<td>0.59s</td>
<td>2.48s</td>
</tr>
<tr>
<td>Simulate Purchase</td>
<td></td>
<td>0.85s</td>
<td>0.22s</td>
<td>2.32s</td>
</tr>
<tr>
<td>Select Document</td>
<td></td>
<td>0.74s</td>
<td>0.22s</td>
<td>2.31s</td>
</tr>
<tr>
<td>Inputs Number</td>
<td>document_number</td>
<td>2.77s</td>
<td>0.59s</td>
<td>2.39s</td>
</tr>
<tr>
<td>Apply Discount</td>
<td></td>
<td>0.73s</td>
<td>0.21s</td>
<td>2.24s</td>
</tr>
<tr>
<td>Inputs Discount</td>
<td>discount_cat</td>
<td>0.83s</td>
<td>0.24s</td>
<td>2.35s</td>
</tr>
<tr>
<td>Pay with Paypal</td>
<td></td>
<td>2.39s</td>
<td>2.23s</td>
<td>4.88s</td>
</tr>
<tr>
<td>See Alerts</td>
<td></td>
<td>1.84s</td>
<td>0.83s</td>
<td>1.99s</td>
</tr>
<tr>
<td>Inputs Alert</td>
<td>alert_type</td>
<td>3.33s</td>
<td>2.36s</td>
<td>4.62s</td>
</tr>
<tr>
<td>Subscribe</td>
<td>alert_type</td>
<td>3.46s</td>
<td>2.73s</td>
<td>5.39s</td>
</tr>
<tr>
<td>Subsribe Alert</td>
<td>alert_type</td>
<td>1.57s</td>
<td>0.65s</td>
<td>2.88s</td>
</tr>
<tr>
<td>FAQs</td>
<td>laq_cat</td>
<td>1.42s</td>
<td>0.63s</td>
<td>3.15s</td>
</tr>
<tr>
<td>Average times</td>
<td></td>
<td>2.08s</td>
<td>1.15s</td>
<td>3.14s</td>
</tr>
</tbody>
</table>

Table 2: Evaluation Metrics

<table>
<thead>
<tr>
<th>Concept</th>
<th>Category</th>
<th>Quality Attribute</th>
<th>Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effectiveness</td>
<td>Functionality</td>
<td>Can detect user intent*</td>
<td>1-100%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Task achievement</td>
<td>1-100%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time elapsed per task</td>
<td># of seconds</td>
</tr>
<tr>
<td></td>
<td></td>
<td>User turns per task</td>
<td># of turns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maintain conversation context</td>
<td>1-5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ease of use</td>
<td>1-5</td>
</tr>
<tr>
<td>Satisfaction</td>
<td>Humanity</td>
<td>Represented a friendly personality</td>
<td>1-5</td>
</tr>
<tr>
<td></td>
<td>Affect</td>
<td>Turn tasks more fun or interesting</td>
<td>1-5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Willingness to use system again</td>
<td>1-5</td>
</tr>
<tr>
<td></td>
<td>Accessibility</td>
<td>Robust understanding messages</td>
<td>1-5</td>
</tr>
<tr>
<td></td>
<td>Ethics</td>
<td>Protect and respect user privacy</td>
<td>1-5 (not evaluated)</td>
</tr>
</tbody>
</table>

4.1.1 Functionality Tests

During all the development I had to manually test the most possible path of conversation I could, created new ones based on previous errors. Testing versions before deploying to production was not reliable. Although this feature was not finished, I started populating example conversations in database, by functionalities, which will be used as tests to verify every past implemented functionality is working as expected. The goal is to automatically update Table 1, with more and more rows over time, and a new columns for production versions.

4.1.2 Dialogflow Metrics

From 26th September I started keeping information about how frequently Dialogflow wouldn’t recognise user messages. In almost a month it detected 32 out of 829, and 3 with less the 50% confidence rate. Dialogflow’s dashboard includes some analytics, where it shows two types of data related to the agent and the conversations it’s been a part of. The following date range are provided: 1 day, 7 days, or 30 days. For each, an additional dotted line, light blue, indicates the data from the previous day or time period. Regarding CP Bot, I was able to obtain the following metrics:

- **NLU data**: All intents are ordered by the number of times it was used. The table included a total of 40 intents detected with the number of sessions it was used and the percentage of
those who exited the conversation in the specified intent. The conversational paths users have taken when interacting with the agents is summarised into a dynamic chart to analyse user conversation in a user friendly graphical representation. Over the past 30 days, Portuguese and English agents together, had a total of 113 sessions, and an average of 5.66 queries per session.

4.1.3 Dashbot Metrics

Their features are divided into three main groups:

- **Understand**: They collect more than 30 metrics to understand interacting users. They are constantly adding new, varying from Demographics, to sessions tracking and funnels of messages.

- **Real time**: Three useful functionalities, allow pausing the bot, responding manually to users, and see recent messages in an easy friendly platform.

- **Tool For Action**: The audience builder which allows targeting users based on multiple parameters. This powerful feature is not available in the free version, although, they allow broadcasting messages to all engaged users in the last 6 months, and creating Email, Webhook, and Slack alerts for specific types of messages.

4.2. User Testing

Instead of performing all user testing personally, which would have been very time-consuming, I thought about performing them directly through the chatbot in a scalabe way. This was possible by improving the Dialog Manager and Dialogflow modules to handle task-related intents, and tracking the functionality metrics automatically. Users that completed all the tasks were then asked to perform a questionnaire about their experience in a Likert scale, which 1 represents 'Strongly disagree', 3 for 'Neither agree or disagree' and 5 for 'Strongly agree'.

All these times will be shorter for real user intents, reducing the latency from task description message, time spent reading the task, and thinking about how to accomplish it. Based on the 21 users who responded to the questionnaire, every quality attribute measured obtained great satisfaction rates, but this result would be more valuable if those who got stuck also responded to it and analyse the reasons for task failures.

5. Conclusions

The output of this research is a fully functional chatbot trained to respond dozens of CP related intents using user’s natural language inside a Messenger conversation. It was developed to understand most common user intentions, and a menu is always available, where users can choose a functionality and CP Bot guides the conversation accordingly. CP Bot resulted from an initial analysis, producing a prototype used by 50 people to gather feedback and training examples. The final version became a stateless solution fully hosted in Google Cloud Platform and with more functionalities than initially planned.
Dashbot was integrated for message analytics since the beginning, and allowed to see useful metrics regarding past conversations and allow pausing the bot when a CP representative is available. The process of matching user intents to existing features or highlighting the need for new responses is a continuous cycle that moves the bot from a decision tree structure to a natural conversation. The most successful chatbots are constantly adapting and revising their conversation flows in response to their users.

5.1. Limitations
One of the biggest concerns about using chatbots is definitely its data privacy and security. When receiving personal information the user should know why it is being saved and preferably hidden before forwarding it for analytics or NLU provider. Comparing to chatbots inside website, those integrated with Instant Messaging channel have an additional integration and must be compliant with their platform policies. Only in 2019, Messenger was updated 8 times. Some important features are being eliminated to make the mobile app lightweight. These include the end of location quick reply and list templates which will have impact in providing near station information and showing train times. CP Bot interacted with a total of 207 users, but most of them are Portuguese, which resulted in much less training of the English conversational agent. Over time I would translate training phrases just to simulate interactions, although real testing with English speakers is required to take conclusions about English agent performance.

5.2. Future Work
All the proposed functionalities in CP Bot are working correctly, but in order to bring CP Bot into production some additional features should be implemented, which were divided into the following topics:

NLU Training
- Automatically generate NLU intent and entities training data in Dialogflow from spreadsheets, specially to update for FAQ
- Implementing sentiment analysis

Development
- Integrate with ticket emission service to reserve seat and associate ticket purchases with the user
- Finish GDPR functionalities, which includes forwarding data deletion requests to both Dialogflow and Dashbot
- Support MB Way payments to accommodate Portuguese citizens

Testing and Analytics
- Run automated tests based on example conversations to assure chatbot is responding correctly through all trained conversation paths.
- Implement A-B testing functionality, giving two groups different responses and comparing the metrics obtained for each.

References