DESIGN AND IMPLEMENTATION OF A CHATBOT AS A TOOL TO ASSIST A HELPDESK TEAM

Diogo Pinto Ribeiro[1], Ana Anjo[2] and Pedro Rangel Henriques[1]
[1]*Centro Algoritmi & Dep. de Informática, Universidade do Minho, Braga - Portugal*
[2]*VILT Group, Portugal*

ABSTRACT

The existence of internal helpdesk teams is a common occurrence in companies nowadays, especially when considering the IT sector. These teams are an expensive resource and are only able to serve a limited number of users at a given moment, which evidences the importance of helpdesk teams operating as efficiently as possible. A common occurrence in the daily operations of these teams consists in the existence of a set of repeated tasks that could be automated through the usage of a chatbot capable of acting on behalf of helpdesk team members. By allowing a chatbot to perform some of these repeated actions, helpdesk teams are able to focus on other tasks, thus allowing to increase their productivity. Additionally, the usage of chatbots to assist a helpdesk team creates a highly available tool, capable of giving answers in a short time frame. In this paper, the design and implementation of such a tool is presented, including concepts and approaches related to chatbot development. As a result, a fully functional chatbot named Triton was produced, capable of helping employees of a consulting company with helpdesk-related problems and questions.

KEYWORDS

Chatbot, Helpdesk, Knowledge Bases, Conversational Agent, Rasa

1. INTRODUCTION

Nowadays, most companies that offer products or services have helpdesk teams available for different purposes. The existence of these teams is expensive and cannot guarantee that all users are served immediately or in a short time frame. One of the most common tasks of a helpdesk team consists in solving repetitive problems or replying to questions that have already been answered somewhere in the past. This takes up valuable staff time and could be solved with the existence of a chatbot. Other scenarios include cases where users need to access documentation pages, wasting time that helpdesk teams could be dedicating to other tasks. The process of searching for documentation could be done by a chatbot in an efficient and simple way. The existence of a chatbot allows users to get replies with a low response time compared to a human. Alongside low response times, a chatbot allows users to be more productive by automating certain tasks. Another advantage of a chatbot is its relatively low maintenance cost compared to a helpdesk team with the same level of availability and the fact that it allows the helpdesk team to focus on more important tasks.

The main motivation behind the development of a chatbot to assist the helpdesk team of a consulting company lies in the possibility of having an agent capable of integrating the helpdesk team and providing support to all employees, thereby freeing the helpdesk team to work on other tasks. Besides freeing the helpdesk team, another motivation for the implementation of a chatbot involves the possibility of employees getting faster and more straightforward answers to a set of common problems and questions.

The main goal of the implemented solution consists of having a fully functional chatbot capable of interacting with users by answering their questions with useful information regarding aspects such as company procedures or troubleshooting. To do so, it is expected for the chatbot to access the internal infrastructure of the company, including various knowledge bases in order to retrieve information. Some of the most relevant functionalities of the chatbot include the capability of searching information in wikis, getting links to open helpdesk tickets, or accessing troubleshooting pages.

In this paper, the implemented solution is presented, aiming to give a better understanding about the research and development process of a chatbot in an industry-focused setting.

The remainder of this paper is organized as follows. Concepts regarding chatbot development and related technologies are presented in Section 2. In Section 3, the architecture of the implemented solution is proposed. Section 4 explains the implementation details of Triton. It also discusses various problems faced during the development stage, along with their respective solutions. The Triton prototype so far implemented is presented in Section 5. Lastly, Section 6 ends this paper and discusses the state of the project, as well as future work.

2. STATE OF THE ART

A chatbot can be defined as an interface that allows users to interact with a software application using natural language. The main objective of a chatbot is to simulate a human conversation, and it can range from a simple Question & Answering System to a complex digital assistant. Chatbots may also be referred to as Bots, Conversational Agents, Chat Agents, or Conversational Interfaces (Shevat, 2017).

In this paper, text-based conversational interfaces are the main focus. This type of interface started to appear in the 1960s and 1970s in the form of Question Answering systems. The first systems to appear had a limited syntactic structure and rejected questions that were out of their scope, but decades later, with advances in research on topics such as AI and NLP, the capabilities of these systems evolved into tools that are used on a daily basis by all kinds of industries and millions of users (McTear, 2020). Chatbots are an example of text-based conversational interfaces and can be found on websites or applications such as WhatsApp or Slack.

According to Google Trends and as of May 2022, the term chatbot has a score (ranging from 0 to 100) of 78, having peaked in April of 2020. This is an excellent indicator of the popularity of chatbots alongside users. Looking at Figure 1, one can infer that chatbots are quite popular at the moment.

Figure 1. Google Trends result for the chatbot topic

According to research published on CNBC, by 2022, chatbots could help reduce business costs by more than $8 billion per year, being that the banking and healthcare industries are the ones that can benefit the most due to the high volumes of human interaction inherent to these businesses (Gilchrist, 2017). Even back in 2017, when this research was published and chatbots were still gaining some traction, the prediction of their impact on the industry was significant, which allows one to better understand the possibilities and magnitude of this kind of tool, and therefore, why it is interesting to be used to assist a helpdesk team.

Another interesting aspect regarding chatbots lies on the possibility of integrating them into existing messaging applications like Facebook Messenger or WhatsApp. Many of these messaging applications have millions of users worldwide, and in the case of WhatsApp, around 2 billion monthly active users according to Statista (2022). With these kinds of figures, an opportunity is presented for companies to reach their customers, using applications already installed on most of the user's mobile devices, instead of developing dedicated applications for chatbots. This would require additional development and deployment time, and there is no guarantee that users will be installing a dedicated application just to interact with a bot. Various companies have already taken advantage of this, having chatbots available on platforms such as Facebook Messenger. Some examples include Volkswagen and Domino's.

To build chatbots there are different tools available, being that some of the most known include Google Dialogflow (Google, 2022), Microsoft Bot Framework (Microsoft, 2022), Wit.ai (Wit.ai, 2022), IBM Watson (IBM, 2022), and Rasa (Rasa, 2022). Each tool has its strengths, but in this paper, the focus will be on Rasa given that this tool was chosen due to its open-source nature and the fact that it can easily be deployed in the company's infrastructure.

To evaluate the performance of a chatbot, different metrics may be collected. These can be grouped into different categories called perspectives such as the user experience perspective, the information retrieval perspective, the linguistic perspective, the technology perspective, and the business perspective (Peras, 2018). Different metrics were researched and are further explored in Section 5 as part of the evaluation process.

3. TRITON – ARCHITECTURE

The architecture of Triton is essentially composed by four components. The first of these components is the Conversational Agent. The Conversational Agent is the core of the proposed solution since it will contain all of the logic necessary to maintain conversations and execute actions. The Conversational Agent will need to communicate with at least three other components: an Input/Output Channel component, a component that allows interaction with the knowledge bases, and a component consisting in all Virtual Machines of the company's infrastructure. Ideally, there will be only one instance of the Conversational Agent for two reasons: the scale of the chatbot to be developed does not justify multiple instances of the Conversational Agent, and the limited resources available to build the bot only allow to build a centralized approach. Having a single instance of the Conversational Agent has some benefits like the lower cost to host the solution, and the simplification of the development process since concurrency problems will not be an issue. On the other hand, the main drawback of this approach consists in it being limited regarding the number of users that will be able to have conversations simultaneously.

As mentioned previously, the Input/Output Channel component is directly linked to the Conversational Agent and is used as the primary gateway between the user and the bot itself. Despite the Input/Output Channel not being directly part of the bot, it is the primary contact point for the user, and therefore has a huge importance regarding the perception of the bot by the users. With that in mind, the main focus will be to integrate the Conversational Agent with Google Chat given the fact that it has a user-friendly interface and allows for integration with bots through the available API.

The knowledge bases are another essential component in Triton's architecture. These knowledge bases contain information about various topics such as troubleshooting information or wiki pages.

The first of these knowledge bases is Zeus. Zeus is a platform that serves as a human resource management platform, project management platform, and knowledge base. Zeus allows employees to log in and register in their personal timesheet how many hours they have been working, register travel expenses, schedule vacations, signal if they are working from home, access information about different ongoing projects, and access information about topics such as troubleshooting, configurations, onboarding of new employees, or human resources questions. To access information about employees, Zeus interacts with an employee database hosting all employee data such as available holidays or working hours. The second knowledge base to be presented is Athena. Athena is built using Redmine and serves as a repository of different kinds of information regarding topics ranging from development guidelines to specific information about projects. The third and last knowledge base to be presented is Cognos. Cognos is built using a tool called Confluence which is developed by Atlassian and serves as a knowledge repository for the company. Cognos is an effort of centralizing information scattered across different platforms such as Zeus and Athena, with the main goal of helping employees access information in a streamlined and easy way. To access Cognos, employees use their credentials, just like with Zeus and Athena, and additionally, a VPN connection is required to access the platform. Logged users may create additional pages about new topics or add pages to existing ones. An interesting feature of Cognos consists in the availability of an API that allows one to search for content either by using a specific identifier or by using the Confluence Query Language.

The fourth and last component comprises the Virtual Machines present in the company's infrastructure. This component does not interact directly with the Conversational Agent, being that it is accessed indirectly by it, through knowledge bases with the target of accessing information.

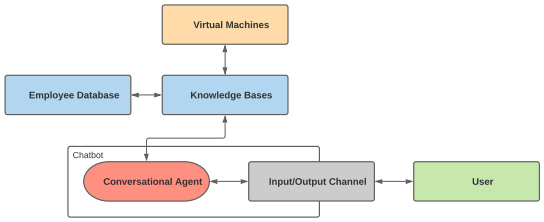

Figure 2. Triton high-level architecture

Figure 2 illustrates a high-level overview of Triton's architecture, including the interactions between the different components. This picture is an abstraction of the implemented solution, lacking any technical details to avoid unnecessary complications. Information about tools used and implementation details are presented in the following section.

4. TRITON – IMPLEMENTATION

To implement Triton, the tool of choice was Rasa (Rasa, 2022). This tool was chosen largely due to two reasons: allowing the deployment in the company's internal infrastructure and including an open-source component to develop the bot (Rasa Open Source). Additionally, Rasa includes an SDK that allows developers to write custom actions that can be triggered with certain user messages, being this a desirable trait for the problem at hand. These custom actions are executed by the Action Server, which will be the component of the bot responsible for interacting with the knowledge bases. As previously mentioned, the bot was integrated with Google Chat given that the company uses Google's suite of tools internally, and due to the fact that by integrating the boot in Google Chat, the bot gains support for both desktop and mobile interfaces. Triton's architecture is shown below in Figure 3, including the details regarding Rasa and the Google Chat integration.

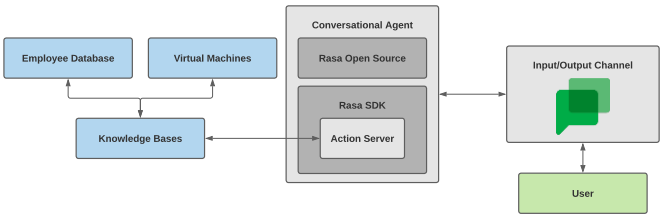

Figure 3. Detailed architecture

Before starting the implementation of the chatbot itself, a process of intent definition and gathering of respective utterances was undertaken. The result was a set of intents that the bot should be able to recognize and answer, being that for each intent, a set of approximately 10 utterances was gathered to train the bot's model. The process of gathering utterances was done using forms that collected data supplied by various employees that will become users in the future. This training data was used throughout the development stages of the bot and was tested at different moments to ensure that the model was behaving as expected.

4.1 Deployment and infrastructure

Regarding the deployment of the solution, there were three options available. The first option consisted in deploying the bot on the Google Cloud Platform, using the Compute Engine service. This service consists of customizable VMs that can be created in Google's infrastructure, and where one can deploy, for example, a chatbot built using Rasa. The second option consisted in deploying the solution in a machine present in VILT's own infrastructure. The third and last option consisted in deploying the bot in a cluster using Kubernetes.

After discussions with the engineering team, from the three options available, it was decided that deploying in the company's own infrastructure was the most adequate approach. By using an instance present in the infrastructure there is a higher degree of control regarding the access to information present in the instance, as well as allowing to save costs since there already exist machines waiting to be allocated to services. Besides the cost saving, deploying in the internal infrastructure may also grant some time savings, since it is easier to deploy the bot on a server, than in a Kubernetes Cluster. During the discussions, it was agreed that deploying the bot using Kubernetes would make sense if it was targeting a considerable number of users, which is not the case. Deploying in the GCP was also discarded since the costs to run the service would be significant.

Despite not using Kubernetes to deploy the solution, using containers is still a very valid proposition. By deploying the chatbot using Docker there are some inherent benefits such as performance compared to Virtual Machines and the portability that containers offer. By containerizing the application, it becomes very simple to deploy it in any other system, having the assurance that it will behave exactly in the same way. With that in mind, the deployment shall be done using Docker and Docker Compose.

Concerning the machine that will be used to deploy the bot, it is a remote Linux host with Ubuntu 20.04.3 LTS as its operating system. The machine has a Intel Xeon Gold 6138 processor, paired with 16GB of memory, and 20GB of storage.

4.2 Conversational Flows

The normal conversational flows are those the bot is trained to reply to, and that the bot is able to identify with a high confidence level. These scenarios may be considered the best-case scenarios when designing the bot since they are those that developers thought about and that match the training data with a higher classification confidence level.

The unsupported conversational flows are those which the bot is not trained to reply to or those which the bot attributes a low confidence level. If not handled properly, these scenarios may lead to a substandard experience for the user since the bot will either crash or will not reply to those specific conversations.

The conversations that the bot is not trained to reply to are designated as out-of-scope conversations. In these scenarios, the bot should reply to the users informing them that it is not able to handle that conversation. Despite being a simple solution, it is a very effective one for the problem at hand given that the bot that is being implemented is a Business Bot, implying that conversations should be concise, which is the case (Shevat, 2017).

There are also cases where the bot has a low classification confidence when predicting intents, in other words, the bot may not be sure if the message sent by the user matches a certain intent due to various reasons such as confusing message syntax or messages being too long or too short. Due to the frequency of these scenarios, it is important to control the outcome so that the user may have a more pleasant experience. This is done by implementing a fallback mechanism that either replies with a default message or tries to solve the existing ambiguity. To solve this ambiguity the bot may ask the user to rephrase the previous message. This is a very useful approach given that it is very likely that the user will rephrase the message sent in other words, thus helping the bot to classify the message with a higher confidence score.

Considering the tool being used to build the bot, there exists the possibility of activating an existing fallback mechanism and tuning some aspects of it. Rasa's fallback mechanism is triggered when a message sent by a user is below a predetermined classification confidence level. This minimum threshold is defined in a configuration file, and through testing, it was agreed that the best relation between the number of fallback situations and classification accuracy resulted in a minimum threshold of 0.9.

4.3 Action Server

The Action Server is a critical component of the solution being implemented given that it contains all the code that executes custom actions triggered by messages received from the users. The Action Server is implemented in Python and uses the SDK made available by Rasa.

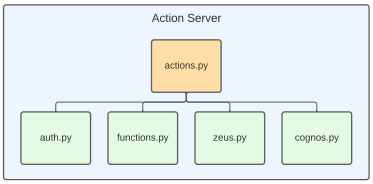

Figure 4. Action Server file structure

Figure 4 illustrates the file structure of the Action Server. The actions file includes the core of the Action Server, mainly all the structure of every available action. The auth file includes all functions related to the authentication process. The functions file consists of a small library of various functions needed to execute parts of different actions. The zeus file contains, as the name indicates, all the functions responsible for communicating with Zeus, similar to the cognos file which is responsible for all the functions related to searches on the Cognos wiki. The code isolation presented was chosen for maintainability reasons.

The implementation of the search capabilities in the knowledge bases followed two different approaches. The first approach consists in using a local Redis database to store data and generate search URLs for the missing information. The second approach uses the Confluence REST API to search for information. Regarding the first approach, after implementing an testing it, some issues were discovered, being that the most important was the necessity of constantly updating the database so that its content does not become obsolete. Given this issue, the second approach using the Confluence REST API was implemented. With the REST API, the consistency problems were solved, and the overall architecture of the solution became a lot simpler, as can be seen in Figure 5. Another interesting feature of the REST API lies in the usage of the Confluence Query Language used to search for information in Cognos. Regarding this approach, the search flow has between one and three steps. When the bot receives a keyword to search for, the first step will be to search by the resource labels. If a match is found, the result is sent to the user, otherwise, the chatbot proceeds with the following step, which consists of repeating the search using the title as a search parameter. As with the previous step, if a result is found, it is sent to the user, otherwise, the third and last step is executed. The last step consists in performing a query that searches for resources that include the given keywords in their page content. Just like before, if a result is found, it is returned to the user. In this specific case, if no result is found, a negative value is returned by the Cognos module in the Action Server, which is then translated into a specific message informing the user that no content was found.

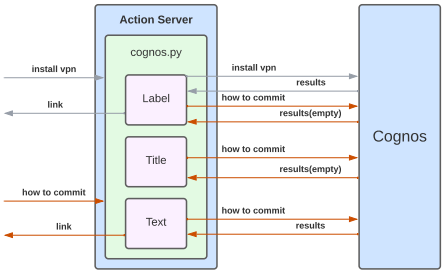

Figure 5. Cognos search flow

The security of the solution being developed was a major concern present in all decisions regarding the implementation of several functionalities, especially the ones interacting with Zeus. When interacting with these components, some prudence was needed regarding the approach that is used to implement the proposed requirements, given that some information should only be visible to specific users, and not all users should have the capability of executing the same set of actions. An example of this could be when a user wants to access information regarding a VM. Some VMs are only visible to a given set of users who have permission to do so. By allowing the bot to access information about VMs, this role authorization structure should be preserved.

Regarding the solution being developed, the first challenge that surfaced was related to the fact that by default, Rasa's implementation for the Google Chat channel identifies the users by their display name, which consists of the user's first and last name. This is a serious problem given that there may exist multiple users

with the same display names, leading to a scenario where the chatbot is incapable of differentiating users. To solve this issue, the source code of the Google Chat channel was adapted to deliver the user's email as its identifier, instead of the display name. The email of a user is always unique given that it is built by joining the user's unique username, to the company domain, therefore solving the issue.

The ensuing challenge faced during the implementation of the chatbot was related to the authorization flow needed to access platforms like Zeus. When a user accesses Zeus, there is an assurance that the user only access resources to which it has access. Given that the bot will access resources on behalf of all users, it is important that the user permissions are preserved. To ensure this, the solution used consists in performing impersonation on top of the Zeus authentication workflow based on OAuth 2.0. To impersonate a user, the chatbot performs the steps described in Figure 6.

In this flow, the chatbot starts by requesting an access token to the Authorization Server, which is then used to make another request to translate a given username into a User ID present in the Authorization Server. With this User ID, the chatbot is able to request an impersonation token for that user to the Authorization Server. After obtaining the impersonation token, the chatbot is able to use this token the same way as a regular access token, impersonating a given user. In the scenario where a user tries to access a resource for which it does not have permissions, the Authorization Server will deny access to the resource returning an HTTP 403 status code. With this flow, the user permissions are preserved when the chatbot accesses Zeus, and therefore, the security concern is solved.

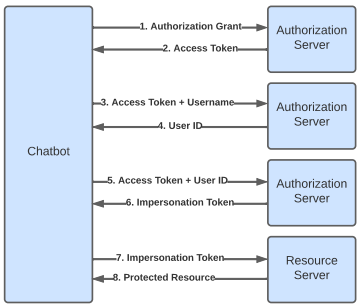

Figure 6. Impersonation flow

5. TRITON – TOOL OVERVIEW

The result of the implementation process is presented in this Section, including some examples of the most relevant functionalities. The first functionality to be shown is related to accessing the status of two different VMs, being that the impersonation flow is shown in action. This can be seen in Figure 7. The retrieval of information about a given VM is shown in Figure 8.

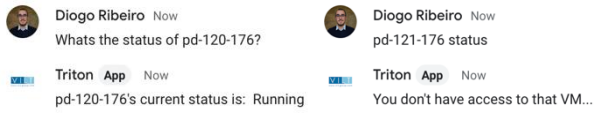

Figure 7. VM status examples

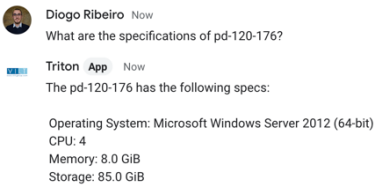

Figure 8. VM specifications example

Figure 9 illustrates the search capabilities of the chatbot in VILT's knowledge bases. In this case, two possible outcomes are shown: an outcome where the chatbot finds a match and returns it to the user, and an outcome where the chatbot is not capable of finding any matches, therefore, returning a message informing the user of the situation.

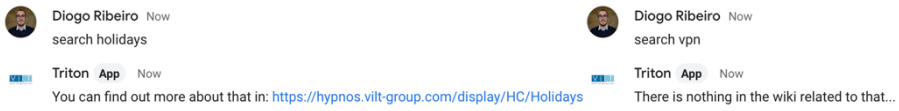

Figure 9. Search examples

The last two examples to be shown include a simple scenario where a user asks for a new VM, receiving a link with VM templates shown in Figure 10 and a scenario where a user expresses the same intent in two different ways as shown in Figure 11.

Figure 10. New VM ticket example

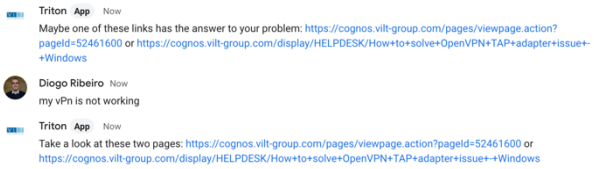

Figure 11. VPN troubleshooting examples

As part of the evaluation process of Triton, an experiment was conducted using a group of 9 employees at VILT with various roles inside the company. 78% of the enquired elements considered Triton's answers to be clear; 89% considered the conversational experience as pleasant. About 67% considered that the bot correctly classified all the messages sent. Overall, 78% of the enquired users considered that Triton was capable of answering their questions or solving their problems and did not need to contact the helpdesk team. Regarding the researched metrics, two were calculated: the Goal Completion Rate, which is around 90%, and the Fallback Rate, which is about 4%. These results are encouraging and show that overall, Triton is well received by the users which tested it and is capable of helping them.

6. CONCLUSION

In this paper, a chatbot named Triton was presented. Triton aims to assist a helpdesk team of a consulting company in helping employees with common problems. The implemented solution was built using Rasa and is integrated into Google Chat, being capable of fetching information from the company's various knowledge bases while preserving the user's security permissions. This implementation differs from other existing ones[1] given the strong customization present in the Action Server made to suit VILT's needs.

For the sake of space, it was not possible to include all research about various chatbot concepts and tools, as well as research related to the Action Server. Nevertheless, this research was included in a Master's dissertation about this topic (Ribeiro, 2022).

The developed chatbot is currently in the final testing stages with a restricted group of employees. This testing process reveals the actual behavior of the chatbot with real users and is helpful in finding errors and bugs. Some of the most interesting lessons learned with the development of this solution lie in the complexity of integrating a bot inside a company's infrastructure while preserving user permissions and without compromises.

[1] https://github.com/RasaHQ/helpdesk-assistant or https://github.com/ldulcic/customer-support-chatbot

Future work includes working on the chatbot's model so that it may handle a wider range of conversations, aiming to minimize the out-of-scope scenarios. Another direction for future work is the research for ways to overcome the limitation of only accessing information, making the bot also capable of executing actions, such as resetting VMs. Some aspects related to the security of the solution were considered in Subsection 4.3, while performance improvements are a topic to be studied in the future.

ACKNOWLEDGEMENT

This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

REFERENCES

Gilchrist, K., 2017. *Chatbots expected to cut business costs by $8 billion by 2022*. https://www.cnbc.com/2017/05/09/chatbots-expected-to-cut-business-costs-by-8-billion-by-2022.html

Google, 2022. *Dialogflow.* https://cloud.google.com/dialogflow

IBM, 2022. *IBM Watson.* https://www.ibm.com/watson

Mctear, M., 2020. *Conversational AI: Dialogue Systems, Conversational Agents, and Chatbots.* Morgan & Claypool, Sebastopol, USA.

Microsoft, 2022. *Microsoft Bot Framework.* https://dev.botframework.com/

Peras, D., 2018. Chatbot Evaluation Metrics: Review Paper. *36th International Scientific Conference*

on Economic and Social Development. Croatia, Zagreb, pp. 89-97.

Rasa, 2022. *Rasa*. https://rasa.com/

Ribeiro, D., 2022. *Chatbot for VILT's Helpdesk Team* (Master's thesis, University of Minho, Braga, Portugal). To Be Discussed

Shevat, A., 2017. *Designing Bots: Creating Conversational Experiences.* O'Reilly Media, San Rafael, USA.

Statista, 2022. *Most popular social networks worldwide as of January 2022, ranked by number of monthly active users*. https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/

Wit.ai, 2022. *Wit.ai.* https://wit.ai/