
Chatbot to Support the Customer Service Process

Submitted 18/02/24, 1st revision 16/03/24, 2nd revision 20/04/24, accepted 16/05/24

Tomasz Smutek¹, Marcin Marczuk², Michał Jarmuł³,
Ewelina Jurczak⁴, Damian Pliszczyk⁵

Abstract:

Purpose: This article aims to discuss the potential benefits and challenges associated with implementing chatbots in customer service. With their ability to automate tasks, answer FAQs, and engage in conversations, chatbots offer unique opportunities for enhancing customer service.

Design/Methodology/Approach: This article provides a comprehensive analysis of chatbots' potential advantages, such as 24/7 availability, quick response times, cost reduction, and increased customer engagement capacity. The challenges that must be addressed for effective implementation are also highlighted.

Findings: The analysis indicates that chatbots can significantly enhance customer service by offering immediate assistance, reducing wait times, and automating repetitive tasks. This automation allows customer service agents to focus on more complex issues, improving customer satisfaction and reducing operational costs.

Practical Implications: The practical implications include reducing the workload on human agents, cost savings due to automation, and providing consistent and efficient customer support at any time. Chatbots' scalability can help organizations meet customer demand without expanding the support team.

Originality/Value: This article offers valuable insights into how chatbots can transform customer service through automation and efficiency. It provides guidance on maximizing chatbots' potential while identifying and addressing challenges that arise during implementation.

Keywords: Chatbot, Customer Service Automation, RASA, Natural Language Understanding, Machine Learning.

JEL codes: M15, M31, C45, D83, L81, L86.

Paper type: Research article.

¹Corresponding Author: WSEI University, Lublin, Poland,
e-mail: Tomasz.Smutek@wsei.lublin.pl;

²University of Economics and Innovation in Lublin, Lublin, Poland,
e-mail: Marcin.Marczuk@wsei.lublin.pl;

³University of Economics and Innovation in Lublin, Lublin, Poland,
e-mail: Michal.Jarmul@wsei.lublin.pl;

⁴Wyższa Szkoła Biznesu - National Louis University, e-mail: ejurczak@wsb-nlu.edu.pl;

⁵Netrix, Lublin, Poland, e-mail: damian.pliszczyk@netrix.com.pl;

1. Introduction

This document presents an approach to creating a bot using the RASA tool (Mishra *et al.*, 2022). The first part describes the tool's functionality, and the next one presents the concept of building a voice assistant using only open-source components. The use of race allows you to create an engine and interfaces for the chatbot, which can be extended with voice functionalities appropriately (Cieplak *et al.*, 2021).

Conversation-driven development (CDD) involves gathering user feedback to refine and enhance AI assistants (Nichol, 2020). It serves as a fundamental best practice for chatbot development. Crafting effective AI assistants is difficult since users often ask unpredictable questions.

The core idea behind CDD is that in every conversation, users clearly express their needs in their own words. Applying the CDD methodology throughout the bot development process allows us to align the assistant with natural language and user behavior. CDD encompasses the following activities:

- Make your assistant available to users as soon as possible.
- Review conversations regularly.
- Annotate messages and use them as NLU training data.
- Check that your assistant always behaves as you expect.
- Track when your assistant goes down and measure its performance over time.
- Define the procedure your assistant will use in case of unsuccessful calls.
- Of course, CDD is not a linear process, and as the bot is created and improved, the same activities mentioned above are returned. The tool built for this purpose is Race X, described briefly in the next chapter.
- In the early stages of bot development, it may seem that CDD has no role to play - especially when we don't have any conversations in the history yet. However, CDD activities can be implemented at the very beginning of bot development.
- Review best practices for NLU data and history for details on creating training data with CDD in mind.
- allow the bot to test users in advance.

CDD focuses on user feedback, so finding test users early is crucial. Test users should be unfamiliar with the bot's inner workings, as they shouldn't be part of the development team. Team members know the bot's limitations, whereas test users should have only as much information as regular end-users (Gołabek *et al.*, 2023).

With CDD, bots receive frequent, incremental updates informed by user interactions. Establishing a CI/CD pipeline early in development enables quick responses to observations from conversations (Singh, 2023).

Adopting this approach from the outset of bot creation is beneficial. Consider both "happy paths" and "unhappy paths" in crafting conversational flows." A happy path occurs when a user follows a conversation predictably and provides the required details when prompted. However, users often deviate from this ideal scenario, introducing questions, chit-chat, or additional queries. This deviation is known as the unhappy path.

The bot must handle unhappy paths well, but predicting what path a given user might take is impossible. Often, developers try to account for every possible divergent path when designing unhappy paths.

Planning every possible state in the state machine (many of which will never be reached) requires a lot of extra work and significantly increases training time. Instead, a conversation-based development approach is recommended when designing unhappy paths.

2. Literature Review

Currently, humanity finds itself in the era of advanced language models like GPT-3 and GPT-4 (Generative Pre-Trained Transformer) (Koubaa, 2023) and BERT (Bidirectional Encoder Representations from Transformers) (Bello *et al.*, 2023). These models can conduct conversations with correct grammar and semantics. However, understanding the history of NLP is essential to fully grasp how these models work (Maj *et al.*, 2023).

Humans created language to communicate and efficiently share information. We've developed complex paradigms for language, which has evolved while retaining its core function of information exchange. For instance, hearing the word "apple" instantly evokes the mental image of a fresh, red fruit. Our brains can seamlessly associate words with mental images, sensations, and feelings.

Computers, however, only comprehend binary data. Explaining something as intricate as a language to a computer is challenging because computers don't follow the same principles we do. Understanding the basics of linguistics is vital before attempting this task (Hamilton and Lahne, 2022).

Linguistics is the scientific study of human language. It methodically and objectively examines all aspects of language. Unsurprisingly, many foundational NLP principles are linked to linguistics. This connection leads us to Ferdinand de Saussure, the father of 20th-century linguistics. In the early 1900s, Saussure introduced a systematic approach to language as a network of interconnected elements during a course at the University of Geneva.

Russian linguist Vladimir Plungyan described Saussure's "revolution" as changing language perception from a chaotic collection of facts to an ordered system.

For Saussure, acoustic sounds in language represent concepts that shift with context. His posthumous book, **Cours de linguistique générale**, highlighted his structuralist approach to language, now fundamental to modern NLP techniques. Saussure and his students advocated understanding language as a system where elements are correlated, identifying contexts through causality (Khurana *et al.*, 2023).

The next linguistic breakthrough came in the 1950s when Alan Turing published "Computing Machinery and Intelligence," also known as the Turing Test. This test determines a computer program's ability to mimic human conversation with an independent judge present (Strawn, 2014).

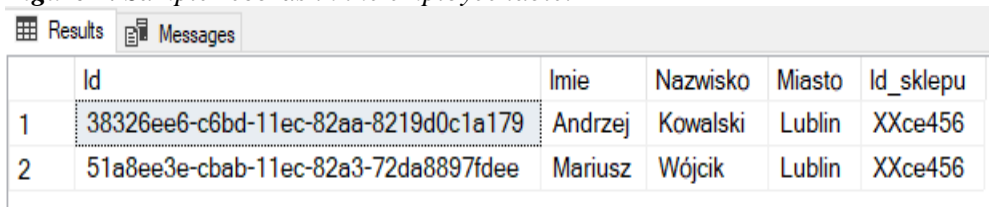
3. Research Methodology

This section will present an example solution that allows you to place a chat window with the option of greeting the user and starting a conversation with the agent. This solution uses the functionality described in the section Adding an assistant to your website, i.e., adding a few lines of code to the website's HTML code, but this time using an initial message.

To better understand the potential of such a solution, a website was created for employees using Flask, containing data on the orders they handle. User data is saved in the database. When a given employee logs in to his account, an action containing his ID will be launched; on this basis, a chat window will open, and the assistant will welcome the person. The subsequent components are described in the following sections.

The first component is, of course, the employee database. The presented example is a simple table containing only the employee's ID, his name, surname, city where he works, and the ID of the store where he works. An example table containing two employees is shown in the figure below.

Figure 1. Sample records in the employee table.



	Id	Imie	Nazwisko	Miasto	Id sklepu
1	38326ee6-c6bd-11ec-82aa-8219d0c1a179	Andrzej	Kowalski	Lublin	XXce456
2	51a8ee3e-cbab-11ec-82a3-72da8897fdee	Mariusz	Wójcik	Lublin	XXce456

Source: Own creation.

After successfully logging in, the user is redirected to the page of orders they manage. The next drawing shows such a page. As you can see, the employee currently has no active orders. But let's focus on what is essential from the chatbot's point of view. As you can quickly notice, after transferring to the website, in

addition to its other functionalities, such as tabs or a greeting in the header, a chat window opens, where the assistant greets the user. This is done with simple lines of HTML code inserted into the page's code. This is an extension of the approach presented in the Adding an Assistant to your Website chapter with additional parameters.

Thanks to this, when the user logs in to his account, a message will be sent containing his ID, and the assistant will greet the employee by mentioning his name. Of course, this approach can be freely extended, but it is only an illustrative example showing the possibilities of such a solution. We can use this functionality after starting the Rasa engine and the action server on the appropriate address.

Figure 1 shows an example dialogue using the model presented in the previous parts of the documentation. Of course, at this stage, the topics and how the assistant can talk depend only on the specificity of his tasks and the functioning of a given company.

This part will present how to conduct a conversation between different assistants. Let's assume that our essential assistant is the one whose properties were described in the previous sections. Apart from him, many other assistants may be related, for example, to the appropriate departments of a given company.

This example will show how our initial assistant can send a query to another bot that monitors currency rates - it can be assumed that this bot is responsible for supporting business decisions. Communication between bots will be carried out using REST API (Wolde and Boltana, 2021).

The following sections will present a simplified process of building an assistant that informs the user about current currency rates, modifications to the assistant presented so far that allow obtaining appropriate information, and an example conversation.

The first important issue is creating a bot that can inform the user about currently applicable currency rates (currency assistant). This is a very simplified approach and is only intended to show how several assistants can be connected. The NBP API will be used to build this functionality, and the assistant will be implemented as a Docker container.

Therefore, it is a solution that allows you to appropriately combine many assistants' functionalities and direct queries. This can be used in various enterprises, where, for example, employees have assistants who support them in fulfilling orders, and management staff have assistants who support them in making business decisions. Then, by adequately defining intentions, management staff can also obtain information about the execution of orders by a given employee without making many changes to their assistant's model.

4. Research Results and Discussion

The presented solution uses the Jina AI tool that implements neural search and is built based on deep learning and artificial intelligence. Using cutting-edge NLP and computer vision techniques, Jina offers a straightforward way to index and search documents, audio, images, and videos. More about the tool itself is included in another part of the documentation.

In the presented example, the goal was to connect the mentioned tool with an assistant built using Rasa (that is why English was considered at the beginning). This allows the user to search data (generally from multiple sources) - in this case, recipe titles. The hugging face dataset was used for this task, which contains recipe titles with full descriptions and an external link (Wolf *et al.*, 2020).

As described in the previous sections of the documentation, Rasa Open Source allows you to define custom actions so that the virtual assistant can run custom Python code. As previously shown, this can be used, for example, to interact with third-party databases or APIs. Because we can run everything that we can define using Python, we can consider another functionality of connecting the assistant with the Jina AI information retrieval engine to recommend the best products to the user based on his description.

To search for a set of information, we can, of course, store the entire data set in memory and use standard string-matching modules to find appropriate recipes. This is, of course, a very inefficient solution, firstly due to storing the data set in memory (as long as the set is small, it will not be a big problem) and secondly, due to not taking into account the context of the descriptions contained in the data set and their substantive meaning - for these things, there are models such as BERT (Bidirectional Encoder Representations from Transformers) that Jina uses (Günther *et al.*, 2023).

The following section will briefly present Jin's principle of operation and how to use the search server using a custom action in the Rasa assistant. As mentioned, Jina allows the use of trained language models (used for various tasks) to increase the accuracy of the searched information.

However, before we get to the code, it is worth mentioning how contextual search works. Of course, our data is indexed to prevent the entire data set from being stored in memory and to ensure faster searching. The indexes in the solution proposed in Jina AI are not based on tokens but on embeddings.

So, in the first step, the analyzed text is embedded as a numerical vector. This step can be performed using a contextual language model like BERT. This procedure is performed for each document in the considered dataset until a collection of vectors representing each document is reached. Jina allows the configuration of any

embedding model and provides a set of models stored in the JinaHub repository.

Once the input documents have been converted into a collection of vectors, we can index them. Therefore, when a query for specific information is made, the vector representation of the input query will be compared with the vector representations of the set of documents that have been indexed. To make this process efficient, we can use, e.g., nearest neighbor search algorithms. Jina also offers many indexing techniques. In this example, we will use indexing with SimpleIndexer, located in their repository.

Technically, the transfer process is the same whether the bot transfers the conversation to a human or another bot. The post-handoff user experience will be different, but bot-to-bot handovers were the first method explored for demonstration and testing purposes.

In any handover, the first consideration is whether the bot will continue participating in the conversation after the handover. The conclusion is that it is almost always better for the bot to be wholly disconnected from the conversation after it is transferred because after the bot is transferred, messages coming through the channel may be entirely outside the bot's domain, have no labels (human transfer) or have inconsistent labels (transfer by bot).

Of course, we can look at human conversations later to see if there's anything we can use to improve the bot. Still, the communicated parts of the conversation should be completely separate from the bot's standalone discussions. This makes it easy to view each conversation by who the user was talking to.

For this reason, it was decided that it was best to configure forwarding from the front end. The flow would work like this: bot #1 would ask the user if they wanted to be sent to bot #2, and the message feed would switch the chat endpoint it was listening to. Once the chat endpoint was changed, bot #1 would no longer receive any messages unless the user requested to be forwarded back to bot #1; in this case, the conversation would continue where it left off.

The way to build the NLU pipeline in Rasa is defined in the config.yml file. This file describes all the steps in the pipeline that Rasa will use to detect intents and entities. This approach begins by receiving text as input, and then through subsequent steps; the individual modules continue parsing until entities and intents are obtained as output.

The first step is to divide the statement into smaller pieces of text called tokens. This must happen before text can be highlighted for machine learning, which is why a tokenizer is usually listed at the beginning of the pipeline. Some tokenizers also add additional information to tokens. For example, spaCy library functions can also generate token lemmas that CountVectorizer can use later.

Features create numerical features for machine learning models. Sparse features produce feature vectors with many zeros, usually consuming significant memory. We use sparse features to optimize storage, recording only the non-zero values and their positions in the vector. This technique minimizes memory usage and allows for training on larger datasets.

Features can generate two types of features: sequential and sentence features. Sequential features form matrices sized (number of tokens x feature size), providing a feature vector for each token in the sequence. Sentence features consist of a (1 x feature size) matrix with a feature vector representing the entire utterance. Both feature types are usable in any model, and the classifier can choose which features to employ.

In addition to features for tokens, we also generate features for the entire sentence. This is sometimes also referred to as the CLS token. The rare features in this token are the sum of all the rare features in the individual tokens. Dense features are either a collective sum/average of word vectors (in the case of spaCy) or a contextualized representation of the entire text (in the case of hugging face models).

What is essential from the point of view of building your solutions and, above all, the project is that you can freely add your own components using non-standard feature creation tools.

5. Conclusions, Proposals, Recommendations

Chatbots are a valuable tool in customer service, benefiting both customers and companies. Continuous optimization of chatbots is necessary, both technically and content-wise, to provide users with the best possible experience. Integrating chatbots with other customer service systems can increase their effectiveness and usability.

Despite the advantages of chatbots, they should not replace interactions with live consultants, especially in the case of more complex and emotional customer problems.

Upcoming research should focus on improving chatbots' intelligence, personalization, and integration with advanced analytical systems to better understand customer needs and provide them with comprehensive service.

Additionally, exploring the use of artificial intelligence-based technologies, such as natural language processing and machine learning, may contribute to further developing chatbots in customer service.

References

- Bello, A., Ng, S.C., Leung, M.F. 2023. A BERT Framework to Sentiment Analysis of Tweets. *Sensors*, 23(1). <https://doi.org/10.3390/s23010506>.
- Cieplak, T., Rymarczyk, T., Kosowski, G., Maj, M., Pliszczyk, D., Rymarczyk, P. 2021. Methods of process mining and prediction using deep learning, *Metody eksploracji i prognozowania procesów z wykorzystaniem głębokiego uczenia*. *Przegląd Elektrotechniczny*, 97(3), 146-149.
- Gołabek, Ł., Pliszczyk, D., Maj, M., Bogacki, S., Rzemieniak, M. 2023. Artificial intelligence in a distributed supply chain control model for personalizing and identifying products in real-time. In: *Innovation in the Digital Economy* (pp. 85-97), Routledge. <https://doi.org/10.4324/9781003384311-8>.
- Günther, M., Milliken, L., Geuter, J., Mastrapas, G., Wang, B., Xiao, H. 2023. JINA EMBEDDINGS: A Novel Set of High-Performance Sentence Embedding Models. 3rd Workshop for Natural Language Processing Open Source Software, NLP-OSS 2023, *Proceedings of the Workshop*. <https://doi.org/10.18653/v1/2023.nlposs-1.2>.
- Hamilton, L.M., Lahne, J. 2022. Natural Language Processing. In: *Rapid Sensory Profiling Techniques: Applications in New Product Development and Consumer Research*, Second Edition. <https://doi.org/10.1016/B978-0-12-821936-2.00004-2>.
- Khurana, D., Koli, A., Khatter, K., Singh, S. 2023. Natural language processing: state of the art, current trends and challenges. *Multimedia Tools and Applications*, 82(3). <https://doi.org/10.1007/s11042-022-13428-4>.
- Koubaa, A. 2023. GPT-4 vs. GPT-3.5: A Concise Showdown. Preprints, March.
- Maj, M., Rymarczyk, T., Maciura, Ł., Cieplak, T., Pliszczyk, D. 2023. Cross-Modal Perception for Customer Service. *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. <https://doi.org/10.1145/3570361.3615751>.
- Mishra, D.S., Agarwal, A., Swathi, B.P., Akshay, K.C. 2022. Natural language query formalization to SPARQL for querying knowledge bases using Rasa. *Progress in Artificial Intelligence*, 11(3). <https://doi.org/10.1007/s13748-021-00271-1>.
- Nichol, A. 2020. Conversation-Driven Development. *The Rasa Blog: Conversational AI Platform Powered by Open Source*.
- Singh, N. 2023. CI/CD Pipeline for Web Applications. *International Journal for Research in Applied Science and Engineering Technology*, 11(5). <https://doi.org/10.22214/ijraset.2023.52867>.
- Strawn, G. 2014. Alan turing. *IT Professional*, 16(1). <https://doi.org/10.1109/MITP.2014.2>.
- Wolde, B.G., Boltana, A.S. 2021. REST API composition for effectively testing the Cloud. *Journal of Applied Research and Technology*, 19(6). <https://doi.org/10.22201/icat.24486736e.2021.19.6.924>.
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., Von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., ... Rush, A.M. 2020. Transformers: State-of-the-Art Natural Language Processing. *EMNLP 2020 - Conference on Empirical Methods in Natural Language Processing, Proceedings of Systems Demonstrations*. <https://doi.org/10.18653/v1/2020.emnlp-demos>.