Universitat de Girona
Escola Politècnica Superior

Treball Final de Màster

Estudi: Màster en Ciència de Dades

Títol: Ajustament d'un model generatiu de llenguatge per a la creació de xatbots personalitzats per administracions públiques

Document: Memòria

Alumne: Martí Mas Fullana

Tutor: Josep Suy Franch
Tutor: Miquel Tarragona Margarit

Departament: Departament d'Informàtica, Matemàtica Aplicada i Estadística
Àrea: Intel·ligència Artificial

Convocatòria (mes/any): Setembre 2024

Universitat de Girona
Escola Politècnica Superior

MASTER'S THESIS

Tuning a Generative Language Model for the Creation of Customized Chatbots for Public Administrations

Author:

Martí MAS FULLANA

Setembre 2024

Master's in Data Science

Supervisor:

Josep SUY FRANCH

Summary

This document presents the design and implementation of a customized chatbot system utilizing GPT (Generative Pre-trained Transformer) and RAG (Retrieval Augmented Generation) technologies to enhance public administration services in Catalonia, specifically for social rights and benefits. The system integrates a frontend interface in Angular with a backend that manages conversational flow and connects with Azure services such as Speech-to-Text and a PostgreSQL database. The backend also employs a Vector Search API for information retrieval and response generation. Key components include data collection and preparation from public websites, implementation of the RAG framework for precise information retrieval, and the development of a user-friendly, multilingual interface with accessibility features. The system's architecture is modular, enabling adaptability for future applications, and is validated through user testing and performance analysis, aiming to improve accuracy, relevance, and user satisfaction while minimizing operational costs.

Acknowledgements

I would like to express my gratitude to my tutor Josep Suy Franch for his guidance and support throughout the development of this project. I would also like to thank Benjamí Fuertes, Sergi Martinez and Joan Oller for their hard work and dedication to the project, and for their patience answering my questions, as well as thank DXC Technology for providing me with this opportunity. I thank my family and friends for their encouragement and understanding during this time. Finally, I would like to thank my fiancée for her unwavering love and support.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

We present a chatbot system that uses GPT (Generative Pre-trained Transformer) technology and RAG (Retrieval Augmented Generation) to provide assistance with social rights and benefits in Catalonia. Our client is the department of social rights of the Generalitat de Catalunya (*Departament de Drets Socials* or DSO). This thesis has been developed in collaboration with ***DXC Technology***.

1.1 About DXC Technology

Figure 1.1: DXC Technology Logo

DXC Technology is a global IT services company that provides a wide range of services to clients in various industries. It has a strong presence in the public sector, providing IT solutions to government agencies and public administrations. The company has a long history of working with public sector clients to develop innovative solutions that improve service delivery and enhance citizen engagement.

DXC Technology is a multinational corporation employing over 130,000 people in over 70 countries, headquartered in Tysons, Virginia, USA. In Spain it has offices in Madrid, Barcelona, Valencia, Bilbao, Sevilla, Madrid and Girona.

1.2 Key Terms and Definitions

- **DSO:** *Departament de Drets Socials*, the Department of Social Rights of the Generalitat de Catalunya. This department is responsible for managing social benefits and services in Catalonia.

- **SNOMED:** A SNOMED, short for Systematized Nomenclature of Medicine, is a code that represents a medical concept. For example, the SNOMED code for "diabetes" is 73211009.

  For reasons not relevant to this project, the Department of Social Rights uses SNOMEDs to represent a user's situation. These SNOMEDs are used to determine which social benefits a user might be eligible for. For example, they have a SNOMED for "Sexist Violence at Home". As we will see in chapter 5, we have a component called *IA Social* that can infer which SNOMEDs might apply to a user based on the conversation.

- **Products:** These are the actual social benefits that the user might be eligible for. These might entail things like monetary aid, legal assistance, or other forms of support. The products are determined by the SNOMEDs that apply to the user. The Department of Social Rights provides us with an API called "kSocial" ("k" is for "Katalog") that we can use to query which products are available for a given set of SNOMEDs.

- **Embedding:** An embedding is a vector representation of a piece of data. Embeddings capture high-level semantic information about the data they represent, and are used in machine learning models to perform tasks such as information retrieval and classification. Embeddings are ususally high-dimensional, with hundreds or thousands of dimensions.

Embeddings are computed by encoder neural networks, and can be used either to later decode them for a specific task (like generating text, as in the case of GPT models) when paired with appropriate decoder neural networks; or directly used for tasks like information retrieval.

In this document, embeddings will refer to Float32 vectors of 3072 dimensions, which store the semantic information of text chunks; except in the case of the embeddings generated by the Hotword Detector (see section 5.1.1), which are 2048-dimensional vectors storing the semantic respresentations of 1.5-seconds-long Mel spectrograms. In all cases these vectors will have a modulus of 1, which simplify the computation of cosine similarity to merely the dot product of the vectors, for compute efficieny.

CHAPTER 2

Background

2.1 Introduction to Dialogue Systems

Dialogue systems, also known as chatbots, have experienced a significant step-change in the last few years. Initially these systems were based on predefined rules and decision trees [2, 3], limiting their capacity for understanding and answering user queries in a natural and flexible manner. These rudimentary systems, commonly referenced as rule-based chatbots, might have been enough for simple tasks, but could not have managed the full complexity and variability of natural language.

2.2 Towards Language Models

As the first machine learning-based language models appeared, such as the Sequence-to-Sequence (Seq2Seq) model [4], and more recently the transformer-based models such as GPT (Generative Pre-trained Transformer) [5, 6], the capacity of chatbots to understand and generate natural language has improved significantly. These models are trained on large datasets of text, learning the complex patterns and structures of language, and are able to generate text that is coherent and contextually relevant.

2.3 GPT and its Contribution

The GPT model [6], developed by OpenAI, has been one of the most notable advances in this field. GPT uses the transformer architecture [5], which is a type of neural network that is particularly well-suited for processing sequences of data, such as text. Its capacity for generating coherent and contextually relevant responses has been leveraged in a wide range of applications, from virtual assistance to automated content generation.

In the following subsections we will go over the evolution of the GPT model, from GPT-1 to GPT-4.

2.3.1 GPT-1

GPT-1 [6], introduced by OpenAI in 2018, marked the beginning of the Transformer-based language models. It used the Transformer architecture, a neural network model that significantly improved over earlier models like recurrent neural networks (RNNs) and Long Short-Term Memory (LSTM) networks. The key innovation in GPT-1 was its ability to generate coherent text by predicting the next word in a sequence, trained on a vast dataset of books and articles. With 117 million parameters, GPT-1 demonstrated that a generative model, pre-trained on a large corpus of text data, could be fine-tuned on specific tasks to achieve good performance across a variety of NLP applications. Although it was relatively small in terms of model size and capacity, GPT-1 established the foundational approach of using unsupervised pre-training followed by supervised fine-tuning.

2.3.2 GPT-2

Building on the success of GPT-1, GPT-2 [7] was released in 2019 and significantly increased in size, with versions ranging up to 1.5 billion parameters.

GPT-2 demonstrated substantial improvements in language understanding, coherence, and the ability to generate more human-like text. It was trained on a much larger and more diverse dataset, which included a wide range of web pages, allowing it to handle more complex language tasks. GPT-2 could perform zero-shot learning, meaning it could complete tasks without specific training on those tasks, just by leveraging the general knowledge it gained during its pre-training phase. This version showcased the model's capacity to handle a wide range of tasks, from text completion and summarization to translation and question answering, making it a significant milestone in AI research. It was the first model to demonstrate the potential for large-scale language models to generate high-quality text across a variety of domains, and the potential of simply scaling up model size to improve performance.

2.3.3 GPT-3

Released in 2020, GPT-3 [8] took another leap in scale and capability with 175 billion parameters, making it one of the largest language models at the time. GPT-3's vast size allowed it to exhibit unprecedented levels of language understanding and generation. It could produce highly coherent and contextually relevant text across diverse topics, maintain longer and more complex conversations, and even exhibit rudimentary reasoning skills. GPT-3 was capable of few-shot and zero-shot learning, further extending its versatility. However, GPT-3 was also computationally expensive and required significant resources for both training and inference, which limited its accessibility.

2.3.4 Reinforcement Learning from Human Feedback (RLHF)

[9] introduced Reinforcement Learning from Human Feedback (RLHF), which marked a significant step forward in the development of GPT models. RLHF is a technique where the model is fine-tuned using feedback from human evalu-

ators who assess the quality of the model's outputs. This method involves the following steps:

- **Pre-training:** The model is initially pre-trained on a large text corpus using unsupervised learning, just like previous versions.
- **Human Feedback Collection:** Human evaluators review the model's responses to various prompts and rank them based on quality, coherence, relevance, and other criteria.
- **Reward Model Training:** A separate reward model is trained on these rankings to predict the human evaluators' preferences.
- **Fine-tuning:** The GPT model is further fine-tuned using a reinforcement learning algorithm, such as Proximal Policy Optimization (PPO) [10], where the reward model guides the optimization to produce outputs more aligned with human preferences.

By incorporating RLHF, GPT models learn not only from the data they are trained on but also from direct human feedback, allowing them to better align with human values, reduce harmful or biased outputs, and improve overall user satisfaction.

2.3.5 GPT-3.5 Turbo

To optimize GPT-3's capabilities for commercial use, OpenAI released GPT-3.5 Turbo, a more efficient variant of GPT-3.5, designed to offer improved performance while reducing computational costs. GPT-3.5 Turbo maintained much of the language understanding and generation quality of GPT-3 but was optimized for lower latency and more cost-effective deployment, making it suitable for a wider range of applications, including real-time interactive systems like chatbots. It also incorporated the improvements from RLHF to enhance the model's ability to generate high-quality text, follow instructions and align with human preferences.

2.4 Retrieval Augmented Generation (RAG)

One of the most recent advances in the integration of language models has been the use of retrieval augmented generation (RAG) [11]. RAG combines the strengths of information retrieval from databases with the generative capacity of language models. In this context, when a user query is received, the system first retrieves relevant information from a database, and then the language model generates a coherent and precise response based on this information. This approach has been shown to improve the accuracy and relevance of the responses generated by chatbots [11].

2.5 Applications and Benefits of RAG-based Chatbots

RAG-based chatbots offer a variety of benefits compared with more traditional systems. They are able to generate responses that are more coherent and contextually relevant. In this way users are both less frustrated and more satisfied, thus improving user experience. These systems also allow the chatbots to have access to newer, more up to date information than the data the model was originally trained on, as the data provided to the information retrieval component can be updated by simply adding new entries to the database. This makes the chatbot more adaptable and flexible, and allows it to provide more accurate and relevant information to users, reducing the necessity of performing full or partial retraining of the model, which can be prohibitively expensive.

CHAPTER 3

Objectives

The main goal of this project is to develop an advanced chatbot system that uses GPT (Generative Pre-trained Transformer) technology and RAG (Retrieval Augmented Generation) to provide responses to user queries based off of the content of a database. This general goal can be broken down into the following specific objectives:

1. **Pick an appropriate GPT Model**
   - Choose a GPT model that is well-suited for the task of generating responses to user queries based on the content of a database.
2. **Integrate RAG Technology**
   - **Information Retrieval:** Develop and implement a system for retrieving relevant information from a database based on user queries.
   - **Combine Retrieval and Generation:** Integrate the information retrieval system with the GPT model to generate coherent and contextually relevant responses to user queries.
3. **Facilitate User-Chatbot Interaction**
   - **UI Design** Develop a user interface that allows users to interact with the chatbot in a natural and intuitive way.
   - **UX Design** Ensure that the user experience is smooth and seamless, and that users are able to easily access the information they need.

4. **Accessibility**

   - **Multilingual Support** Implement support for multiple languages to make the chatbot accessible to a wider range of users.
   - **Accessibility Features** The system must be designed to be accessible to users with visual or motor impairments. As such, it should support voice input. The voice input feature must support activation through a voice command.

5. **Evaluate and Validate the System**

   - **User Testing** Conduct user testing to assess the usability and effectiveness of the chatbot system.
   - **Results Analysis** Analyze the results of the different tests to identify areas for improvement and optimization.

CHAPTER 4

Methodology

4.1 SCRUM Methodology

The development of this project was carried out using the SCRUM methodology, an agile framework that facilitates iterative and incremental progress. SCRUM was chosen for its flexibility and ability to adapt to changes, making it well-suited for the dynamic nature of software development projects that are partially research-based.

4.1.1 SCRUM Roles

In our SCRUM process, the following roles were defined:

- **Product Owner:** The Product Owner was responsible for defining the vision of the project and prioritizing the backlog. They ensured that the development team focused on delivering features that provided the most value to the stakeholders.

- **SCRUM Master:** The SCRUM Master facilitated the process by removing impediments, ensuring that the team adhered to SCRUM practices, and fostering a collaborative environment. They acted as a bridge between the development team and the Product Owner.

- **Development Team:** Composed of multidisciplinary members, the Development Team was responsible for delivering the product increments. Each team member contributed to different aspects of the project, including frontend and backend development, database management, and model integration.

4.1.2 Sprint Planning

The project was divided into multiple sprints, each lasting two weeks. During Sprint Planning meetings, the Development Team collaborated with the Product Owner to select user stories from the product backlog, which were then broken down into smaller, manageable tasks. Each task was assigned a story point value based on its complexity, and the team committed to completing a set of tasks during the sprint.

4.1.3 Daily Stand-ups

Daily stand-up meetings were held to maintain transparency and ensure smooth communication within the team. During these brief meetings, each team member shared what they accomplished the previous day, what they planned to work on that day, and any blockers or impediments they faced. This practice helped identify issues early and allowed for quick adjustments to the plan if necessary.

4.1.4 Sprint Review and Retrospective

At the end of each sprint, a Sprint Review meeting was conducted, where the team demonstrated the product increment to the Product Owner and stakeholders. Feedback was gathered, and any necessary adjustments were noted for future sprints.

Following the Sprint Review, the team conducted a Sprint Retrospective. This meeting focused on evaluating the sprint process, identifying what went well, what could be improved, and creating actionable plans for enhancing future sprints. The Retrospective was critical in fostering continuous improvement within the team.

4.1.5 Backlog Refinement

Backlog refinement sessions were held regularly to ensure that the product backlog remained up-to-date and aligned with the project goals. The team reviewed and re-prioritized user stories, added details to upcoming tasks, and estimated their complexity. This process helped the team prepare for future sprints and maintain a clear understanding of the project trajectory.

4.1.6 Benefits of Using SCRUM

The application of SCRUM in this project provided several benefits:

- **Adaptability:** The iterative nature of SCRUM allowed the team to quickly adapt to changes in requirements or priorities, ensuring that the project stayed aligned with stakeholder expectations.

- **Collaboration:** Regular communication and feedback loops fostered a collaborative environment, enabling the team to work more effectively and efficiently.

- **Transparency:** The use of daily stand-ups and sprint reviews increased transparency, allowing stakeholders to stay informed of progress and providing opportunities for early feedback.

- **Continuous Improvement:** The focus on retrospectives ensured that the team continually refined their processes, leading to better performance and higher-quality deliverables.

By adhering to SCRUM principles and practices, the team was able to deliver a high-quality chatbot system that met the needs of the Department of Social

Rights of the Generalitat de Catalunya in a timely and efficient manner.

4.2 Data Collection and Preparation

- **Data Collection:** We are given by the stakeholders a set of websites documenting the laws related to social rights in Catalonia and also documenting available social benefits. These websites contain a variety of information, including the text of the laws and the social benefits, what conditions are necessary to access them, and how to apply for them.
- **Data Preparation:** We will extract the text from the websites using web scraping and convert them into a format that can be used by the information retrieval system. This will involve cleaning the text and removing any extraneous characters. This will be done using custom web scraping scripts.

4.3 RAG Implementation

- **Information Retrieval:** Develop a system capable of retrieving relevant information from a database based on user queries. This will involve creating an index of the laws related to social rights and available social benefits in Catalonia, and implementing a search algorithm that can return the most relevant laws based on the user query. In practice this will be done using the LlamaIndex Python library, which chunks the text into smaller pieces and indexes them.
- **Combining Retrieval and Generation:** Integrate the retrieval system with the GPT model to generate coherent and contextually relevant responses to user queries. This will involve passing the retrieved information to the GPT model, which will generate a response based on this information.

4.4 User Interface Design

- **UI Design:** Develop a user interface that allows users to interact with the chatbot in a natural and intuitive way. This will involve creating a chat interface that allows users to input queries and receive responses from the chatbot.

- **UX Design:** Ensure that the user experience is smooth and seamless, and that users are able to easily access the information they need. This will involve testing the user interface with a group of users to identify any areas for improvement, as well as demoing the system to the stakeholders.

CHAPTER 5

Architecture

We develop an Angular Frontend conversation UI (similar to other messaging apps) that communicates with a Backend that manages the calls to the language models and the database. The Backend also handles the management of the database and the indexing of the information. The diagram of the app's architecture is shown in Figure 5.1.

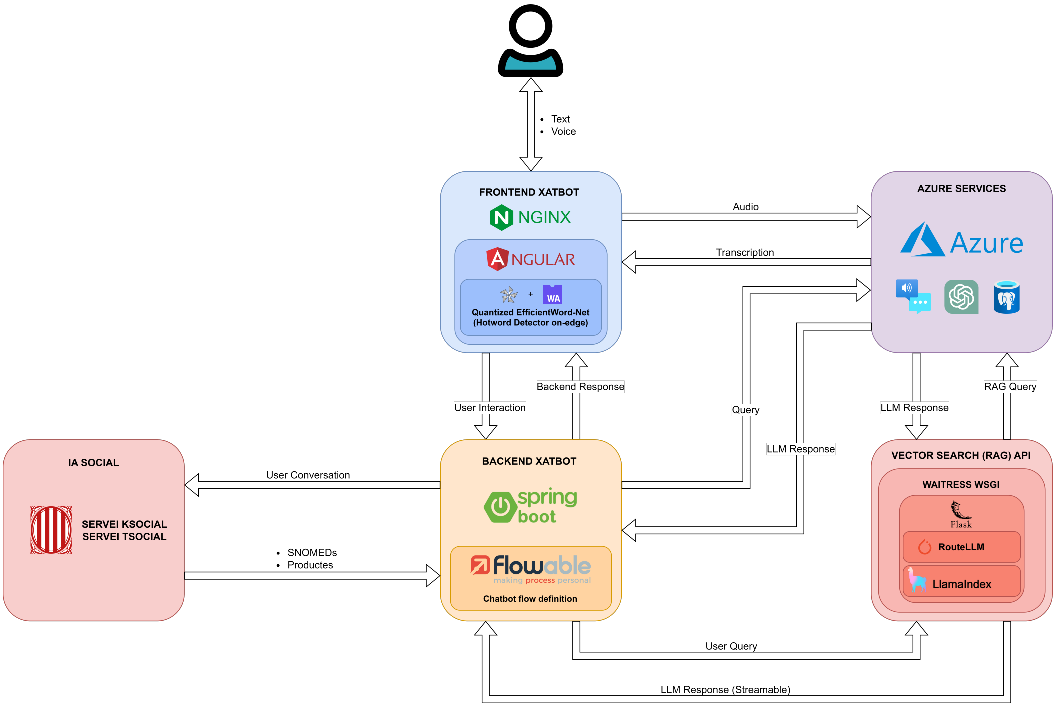
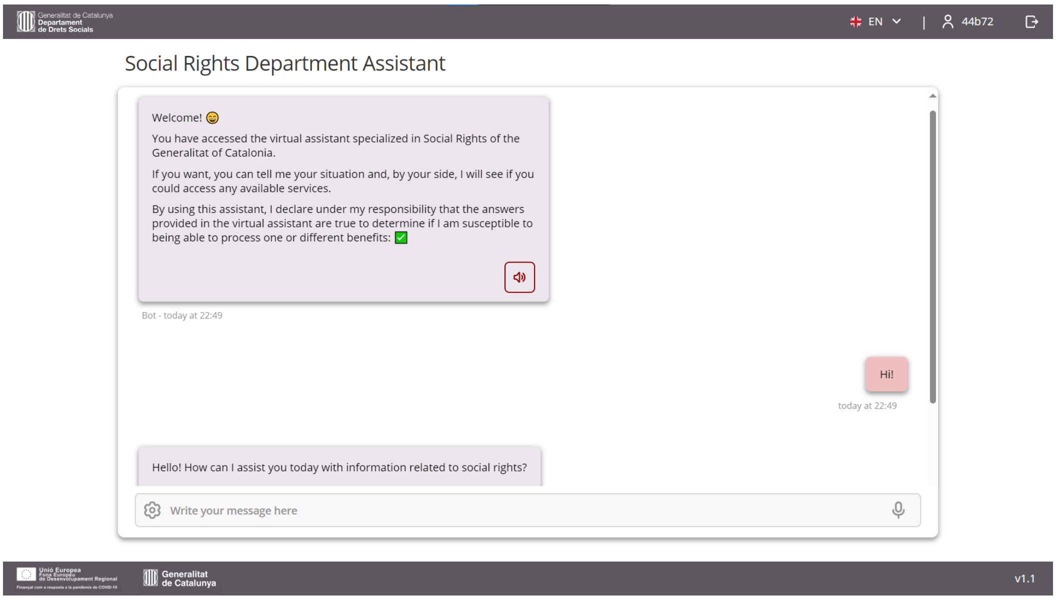


Figure 5.1: App Architecture

Over the following sections we will describe the different components of the system in more detail, and how a user's interaction is processed through the system.

5.1 Frontend

The frontend is implemented in Angular, a popular web development framework. The frontend is responsible for managing the user interface and handling the user's interactions with the chatbot. The frontend communicates with the backend to send user queries and receive responses from the chatbot. It also communicates with the Azure Speech-to-Text service to convert audio data into text. Figure 5.2 shows an example of the frontend interface.



Social Rights Department Assistant

Welcome! 😄

You have accessed the virtual assistant specialized in Social Rights of the Generalitat of Catalonia.

If you want, you can tell me your situation and, by your side, I will see if you could access any available services.

By using this assistant, I declare under my responsibility that the answers provided in the virtual assistant are true to determine if I am susceptible to being able to process one or different benefits: ✅

Bot - today at 22:49

Hi!

today at 22:49

Hello! How can I assist you today with information related to social rights?

Write your message here

Unió Europea
Fons Europeu
de Desenvolupament Regional
Finançat com a resposta a la pandèmia de COVID-19
Generalitat
de Catalunya
v1.1

Figure 5.2: Frontend Interface

5.1.1 Hotword/Wakeword Detection

On the frontend we implement a custom version of the EfficientWord-Net [1] hotword detector, that has been quantized [12]. To implement it we use ONNX [13] and WebAssembly (Wasm) [14], which run directly in the browser, ensuring no data leaves the client's machine inadvertently. The hotword detector listens to the microphone data in real-time, and when it hears the keyword "chat" or "chatbot", it activates the microphone and starts recording. We will see how this data is processed in the 5.2 section.

The EfficientWord-Net model has been quantized to 8-bits, which has reduced the size of the model from 80MB to 20MB, and has improved the response time by 100%. This has been achieved without perceptibly sacrificing the accuracy of the hotword detection.

The hotword detector analyzes audio data in chunks of 1.5 seconds, overlapped by 0.75 seconds. The raw audio signal is first converted to a Mel spectrogram (Figure 5.3), which is then passed through a ResNet [15] model to generate semantic embedding vectors. These vectors are then compared to the embedding vectors of reference recordings of the hotword (which are prerecorded) using cosine similarity. If the similarity is above a certain threshold, the hotword is detected. We use the pretrained model as provided by the EfficientWord-Net library, with no further training.

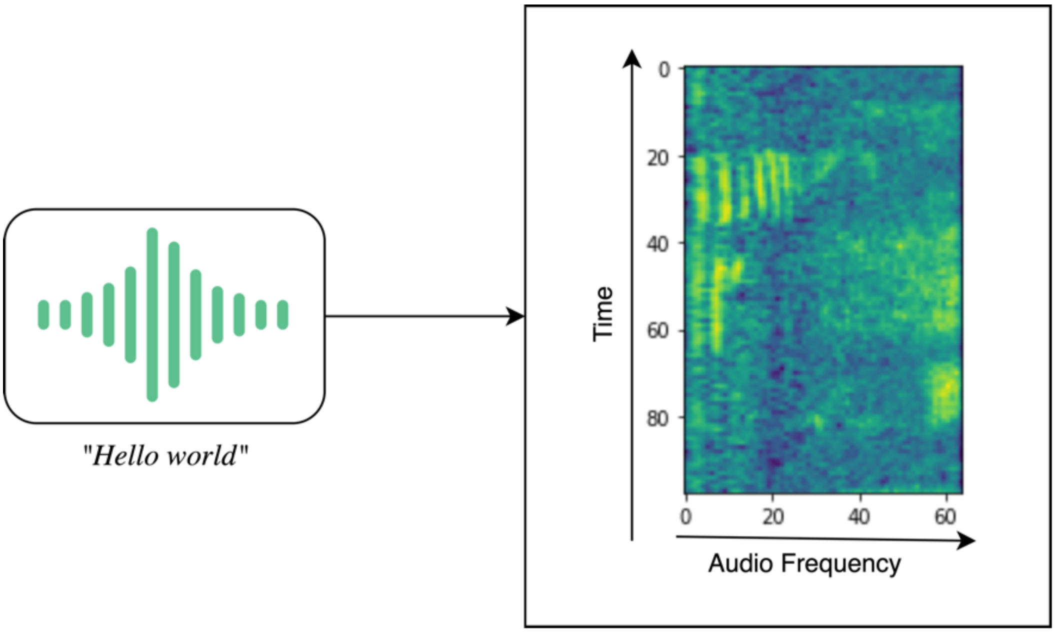


Figure 5.3: Mel spectrogram of the audio “Hello world” (image taken from [1])

Because this system needs to run on the browser, and there is no existing implementation of a Mel spectrogram converter, we implement our own converter directly in TypeScript. This converter is as close as possible to a direct translation –from Python to TypeScript– of the original code from the EfficientWord-Net library. By doing so we ensure data generated by our implementation is equivalent to that generated by the original implementation. We evaluate that this is the case by making bitwise comparisons of the output of both implementations (subtracting one image from the other), and find there are no differences (all pixels in the resulting image are exactly 0).

On our reference system, which consists of a Dell Latitude 3440 laptop with a 13th Gen Intel(R) Core(TM) i5-1345U CPU, the hotword detector has a response time of around 80-100 milliseconds, which is well within the acceptable range for real-time applications.

5.2 Azure Services

5.2.1 Speech-to-Text

When the hotword detector activates the microphone, the audio data is sent to the Azure Speech-to-Text service. This service converts the audio data into text, which is then sent to the Backend for processing. The Azure Speech-to-Text service uses the Whisper [16] model to accurately transcribe speech into text.

5.2.2 Text-to-Speech

From the frontend, a user can choose to have the chatbot's responses read out loud, by clicking on a button. When this happens, the text of the response is sent to the Azure Text-to-Speech service, which converts the text into audio data. This audio data is then sent back to the frontend, where it is played through the user's speakers.

5.2.3 PostgreSQL Database

The PostgreSQL database contains the text of the laws related to social rights and available social benefits in Catalonia. This data is indexed using the LlamaIndex Python library to chunk the text into smaller pieces, index them and help create semantic embeddings. When a user query is received, the information retrieval system searches the database for the most relevant laws and benefits based on the query, and returns this information to the GPT model for generation.

5.2.4 GPT Model

The GPT model is used to generate coherent and contextually relevant responses to user queries based on the information retrieved from the database. The GPT model is a transformer-based [5] language model that is trained on a large dataset of text to generate human-like responses to user queries.

We use the GPT-4o and GPT-4o Mini models, which are the latest versions of the GPT model developed by OpenAI at the time of writing.

5.3 Backend

5.3.1 Conversation Flow

The Backend implements, the conversation flow followed by the chatbot. It implements multiple stages and follows a state machine to manage the conversation. User queries at each stage are routed to the appropriate model.

Conversations flows are implemented using Flowable. Figures 1, 2, 3, 4, 5, and 6 show the flow diagrams for the different stages of the conversation.

At each stage the backend can decide to talk to one of the other components of the architecture. For example, during the RAG stage it uses our Vector Search

API component to generate a response based on the information retrieved from the database. In another case, in the scenario where we are discovering the user's situation, the backend talks to the IA Social component to and asks it if, given the current conversation, there are any SNOMEDs that might apply to the user.

The backend also implements any error handling that might be necessary. The error handling is implemented directly on the flow diagram itself.

5.4 Vector Search API

As we needed to implement a few custom components all related to the RAG system, we decided to abstract them into a single separate component, the Vector Search API. This component is responsible for managing the information retrieval system, and for generating responses based on the information retrieved from the database. This component is abstract enough to be reused in chatbot systems other than the one we are developing for the DSO.

This API is implemented in Python using Flask to serve the API endpoints, and with Waitress as the WSGI server, as depicted in Figure 5.1.

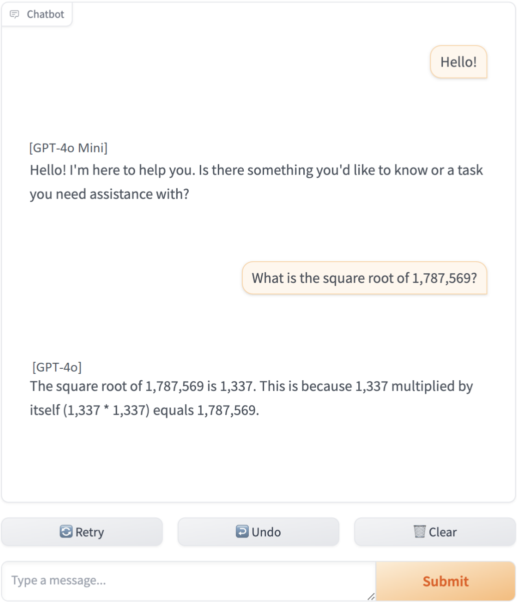
The following sections describe the features that this API provides.

5.4.1 Model Routing

We use a slightly tweaked version of the fairly new RouteLLM Python library. This allows us to route a certain percentage of queries to a "stronger", more expensive model and the rest to a "weaker", less expensive model. With this system we are able to achieve X% of the performance of the stronger model at a fraction of the cost. The RouteLLM system uses a small BERT classifier model that decides which queries should be routed to the stronger model and which to the weaker model. This classifier was trained by the original library authors on a dataset of human preferences augmented with synthetic data generated using GPT-4. They report good generalization performance, so we apply the system on a pair consisting of GPT-4o and GPT-4o Mini. We have also made the necessary changes to the library to make it compatible with the Azure OpenAI models, which didn't have official support.

We have translated the dataset the authors used to train the BERT classifier to Catalan, and are in the process of retraining the classifier on this new dataset, at the time of writing.

The translated training dataset consists of two parts: the translated texts and the embeddings of the texts. The translations were generated using the GPT-4o model and the embeddings using the *text-embedding-3-large* model. The total cost of generating the translations and embeddings was around 500 euros. The datasets can be found here [translated texts] and here [embeddings].



Hello!

[GPT-4o Mini]
Hello! I'm here to help you. Is there something you'd like to know or a task you need assistance with?

What is the square root of 1,787,569?

[GPT-4o]
The square root of 1,787,569 is 1,337. This is because 1,337 multiplied by itself (1,337 * 1,337) equals 1,787,569.

Figure 5.4: Example of conversation messages being routed to different models according to their complexity

5.4.2 Information Retrieval

We do RAG using the LlamaIndex python library.

We use two versions of RAG: normal RAG and Small To Big Retrieval (STBR).

- **Normal RAG:** This is the standard RAG algorithm. It simply creates an embedding of the user query and does cosine similarity with the embeddings of the chunks in the database to retrieve the N most relevant chunks. In practice the database implements this with the Hierarchical Navigable Small World (HNSW) [17] algorithm, which is a fast approximate nearest neighbor search algorithm, in order to avoid having to compare the user query with all the entries in the database's table. However this implementation is invisible, as the database itself executes it.

- **Small To Big Retrieval (STBR):** This is a less common RAG algorithm that uses hierarchical embeddings of chunks, allowing us to capture both fine-grained and coarse-grained information. Figure 5.5 shows our hierarchy-of-embeddings setup. Each level's embeddings correspond to smaller and smaller chunks of the text. We have 3 levels of embeddings. This can be thought of as analogous to the way a CNN model captures both fine-grained and coarse-grained information, where deeper layers have a larger receptive field. We implement the algorithm using the RecursiveRetriever class from the LlamaIndex library.

  The STBR algorithm is implemented in the Vector Search API and runs at the top level, so the embedding searches are still performed by the database itself, still using the HNSW algorithm. Therefore STBR is more expensive because it performs multiple database queries (one per hierarchy level), but it is also more accurate for retrieving the most relevant information.

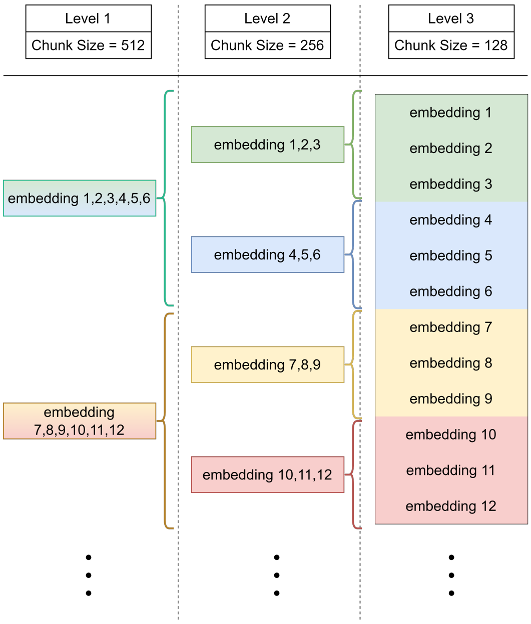


Figure 5.5: Small To Big Retrieval (STBR) embeddings hierarchy

CHAPTER 6

Results

RESULTSSSS

CHAPTER 7

Conclusions

7.1 Our Contributions to Open-Source

During the development of this project we have iterated and made changes to a few existing tools and components. Here we link to our forks of the repositories.

- EfficientWord-Net: https://github.com/SupremeLobster/EfficientWord-Net - Our fork of the EfficientWord-Net library, which has been modified to reduce response time, client machine requirements, and the size of downloaded files, through Quantization methods.

  We have opened a pull request with these changes to the original repository, which is still pending.

- RouteLLM: https://github.com/SupremeLobster/RouteLLM - Our fork of the RouteLLM library, which allows us to route a certain percentage of queries to a “stronger”, more expensive model and the rest to a “weaker”, less expensive model. Our changes are in the branch “feature/support-azure-openai-embeddings”. These changes made the library compatible with the Azure OpenAI models, which didn’t have official support.

  As of the time of writing, we are in the proces of cleaning up the code and opening a pull request with these changes to the original repository.

Bibliography

[1] R. Chidhambararajan, A. Rangapur, S. Sibi Chakkaravarthy, A. K. Cherukuri, M. V. Cruz, and S. S. Ilango, "EfficientWord-Net: An open source hotword detection engine based on few-shot learning," *Journal of Information & Knowledge Management*, vol. 21, no. 04, p. 2250059, 2022. (Cited on pages vii, 20 and 21.)

[2] J. Weizenbaum, "ELIZA: A computer program for the study of natural language communication between man and machine," *Commun. ACM*, vol. 9, p. 36–45, jan 1966. (Cited on page 5.)

[3] B. Abushawar and E. Atwell, "ALICE chatbot: Trials and outputs," *Computación y Sistemas*, vol. 19, 12 2015. (Cited on page 5.)

[4] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014. (Cited on page 5.)

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. (Cited on pages 5, 6 and 23.)

[6] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018. (Cited on pages 5 and 6.)

[7] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019. (Cited on page 6.)

[8] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss,

G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. (Cited on page 7.)

[9] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," 2022. (Cited on page 7.)

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. (Cited on page 8.)

[11] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," 2021. (Cited on page 9.)

[12] J. Zhang, Y. Zhou, and R. Saab, "Post-training quantization for neural networks with provable guarantees," 2023. (Cited on page 20.)

[13] ONNX Community, "ONNX: Open neural network exchange." https://onnx.ai, 2024. https://onnx.ai. (Cited on page 20.)

[14] World Wide Web Consortium (W3C), "WebAssembly: A binary instruction format for a stack-based virtual machine." https://webassembly.org/, 2024. https://webassembly.org/. (Cited on page 20.)

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. (Cited on page 21.)

[16] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, “Robust speech recognition via large-scale weak supervision,” 2022. (Cited on page 22.)

[17] Y. A. Malkov and D. A. Yashunin, “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs,” 2018. (Cited on page 26.)

Flowable Diagrams

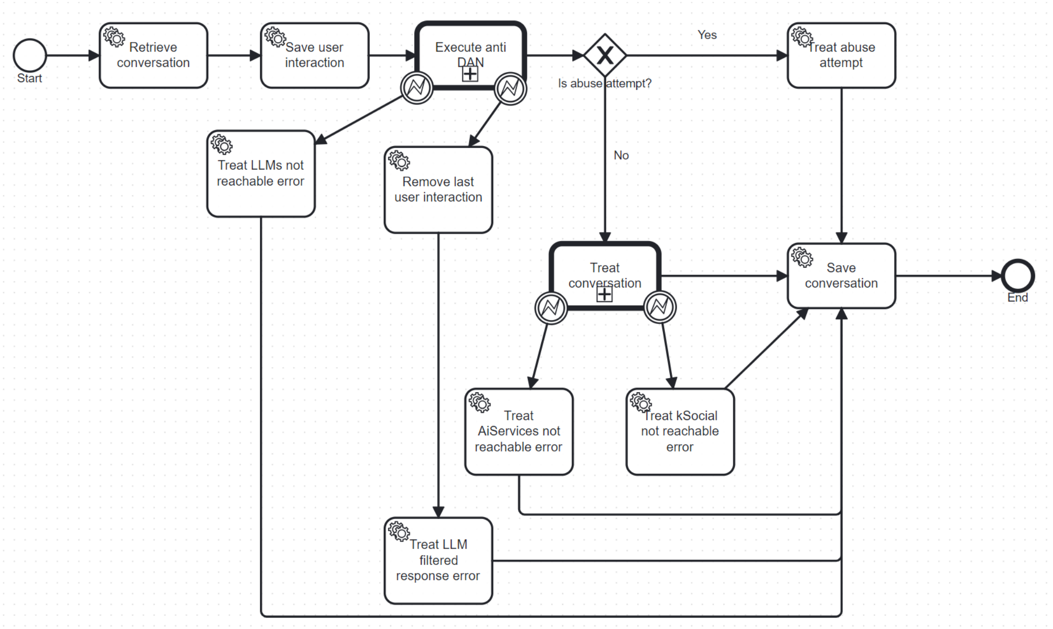

Start
Retrieve conversation
Save user interaction
Execute anti DAN
Is abuse attempt?
Yes
Treat abuse attempt
No
Treat LLMs not reachable error
Remove last user interaction
Treat conversation
Save conversation
End
Treat AiServices not reachable error
Treat kSocial not reachable error
Treat LLM filtered response error

Figure 1: Top level flow diagram for the conversation

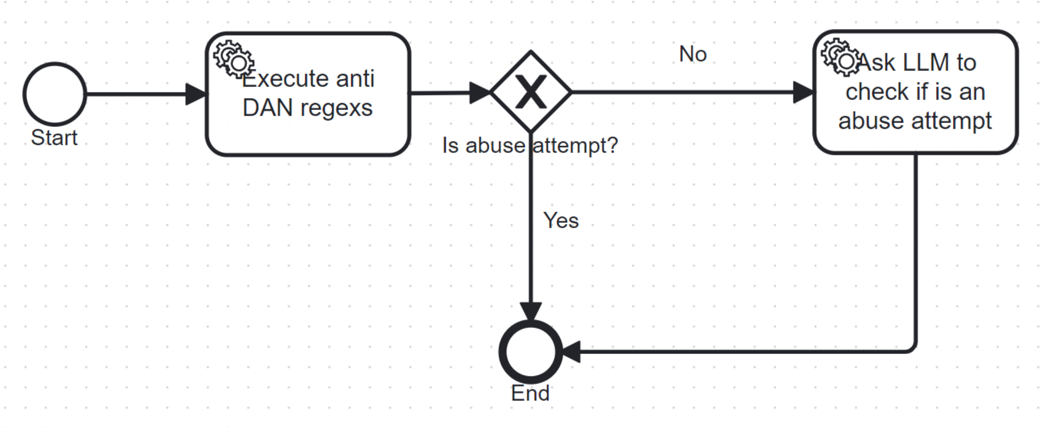


Figure 2: Flow diagram for the Anti DAN stage

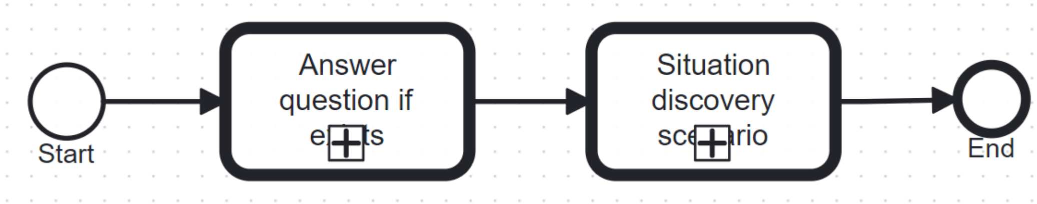


Figure 3: Flow diagram for treating a scenario

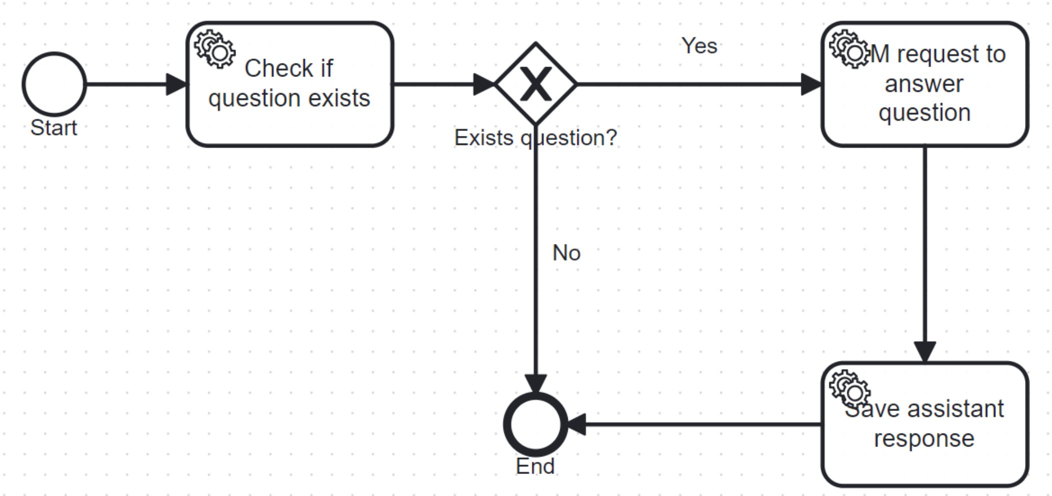


Figure 4: Flow diagram for the answering a question using RAG (if a question exists)

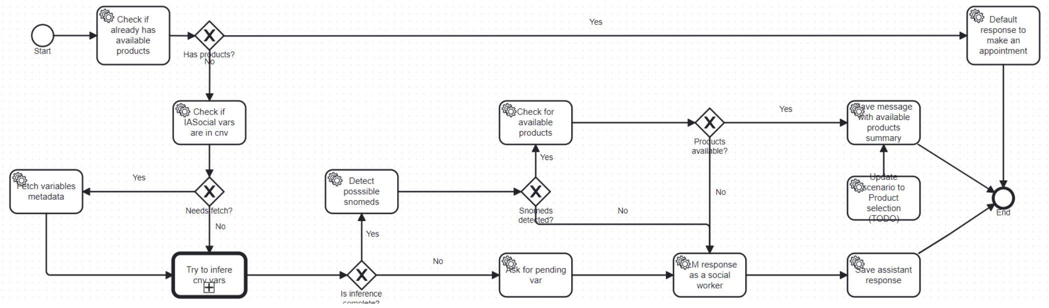

Start
Check if already has available products
Has products?
Yes
No
Default response to make an appointment
Check if IASocial vars are in cnv
Needs fetch?
Yes
No
Fetch variables metadata
Try to infere cnv vars
Is inference complete?
Yes
No
Detect posssible snomeds
Snomeds detected?
Yes
No
Check for available products
Products available?
Yes
No
Save message with available products summary
Update scenario to Product selection (TODO)
Ask for pending var
M response as a social worker
Save assistant response
End

Figure 5: Flow diagram for the scenario where we need to discover the user’s situation

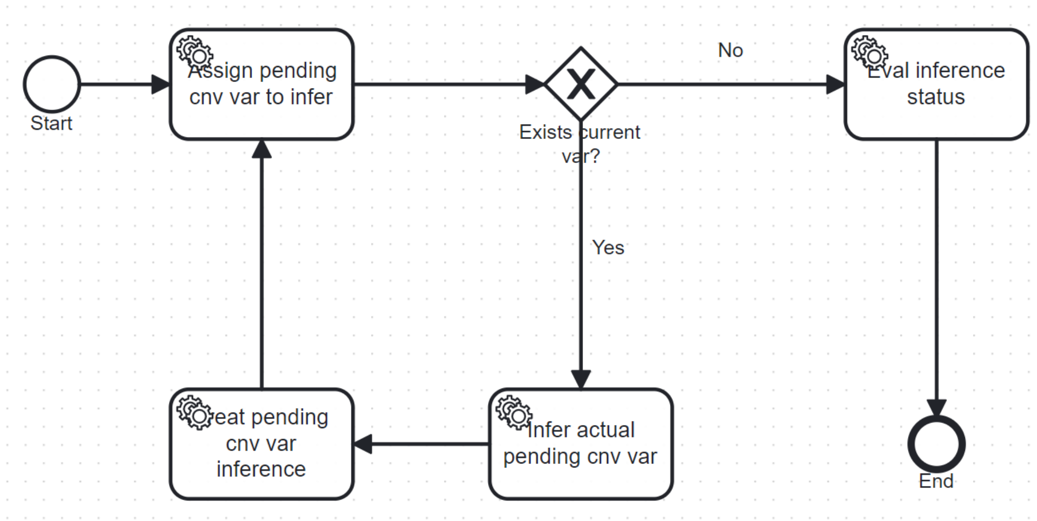

Start
Assign pending cnv var to infer
Exists current var?
No
Yes
Eval inference status
End
Infer actual pending cnv var
Treat pending cnv var inference

Figure 6: Flow diagram for inferring necessary variables for the current scenario based on the conversation