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

TREBALL FINAL DE MÀSTER

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

Autor:
Martí MAS FULLANA

Setembre 2024

Màster en Ciència de Dades

Tutors:
Josep SUY FRANCH
Miquel TARRAGONA MARGARIT

Resum

Agraïments

Per començar vull agrair molt especialment a . . .

Índex

Índex de figures

Índex de taules

CAPÍTOL 1

Introducció

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).

1.1 Antecedents

1.1.1 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.

1.1.2 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.

1.1.3 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.

1.1.4 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) [7]. 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 [7].

1.1.5 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. 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.

1.2 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 be able to be activated 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.

1.3 Methodology

1.3.1 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.

1.3.2 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.

1.3.3 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.

1.4 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 1.4.

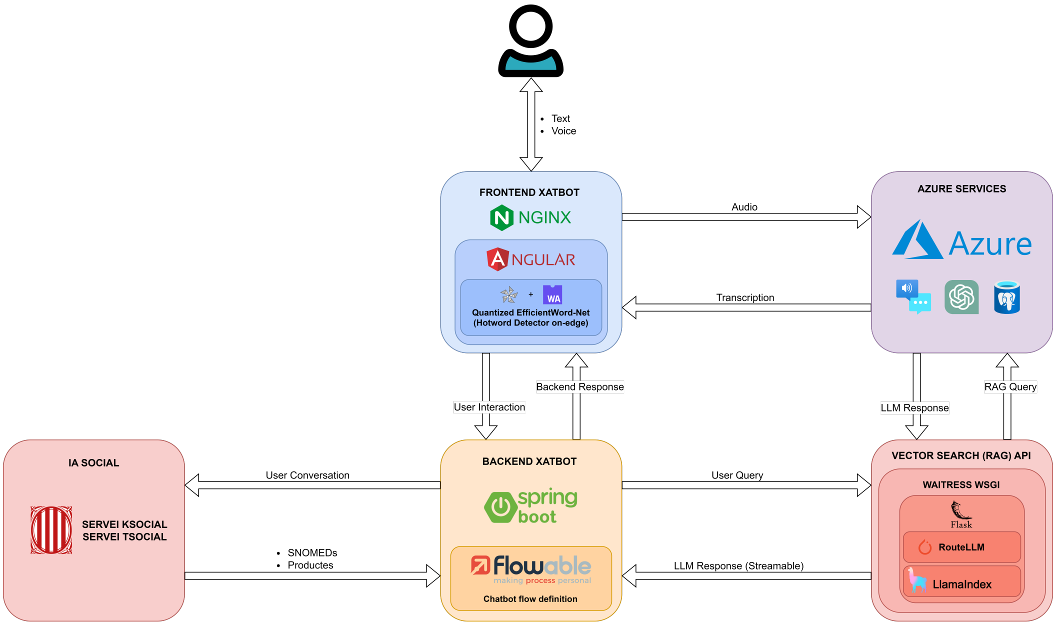


Figura 1.1: App Architecture

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.

1.4.1 Hotword/Wakeword Detection

On the frontend we implement a custom version of the EfficientWord-Net [1] hotword detector, that has been quantized [8]. To implement it we use ONNX [9] and WebAssembly (Wasm) [10], 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 1.4.2 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 1.2), which is then passed through a ResNet [11] 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.

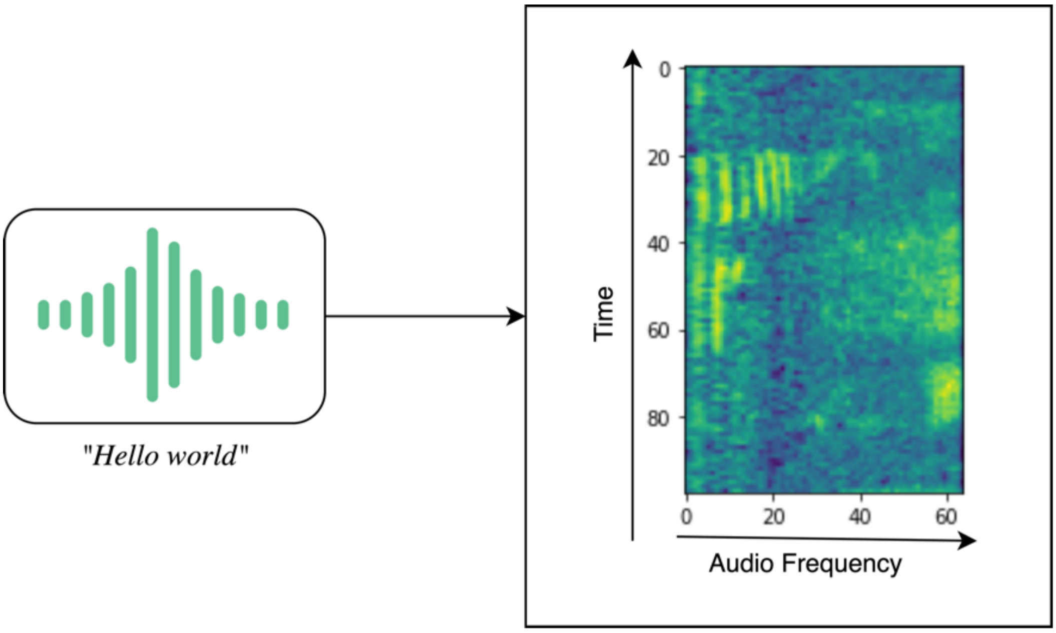


Figura 1.2: Mel spectrogram of the audio “Hello world” (image taken from [1])

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.

1.4.2 Azure Services

1.4.2.1 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 [12] model to accurately transcribe speech into text.

1.4.2.2 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.

1.4.2.3 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.

1.4.3 Backend

1.4.3.1 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 1.4.3.1, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, and 1.10 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.

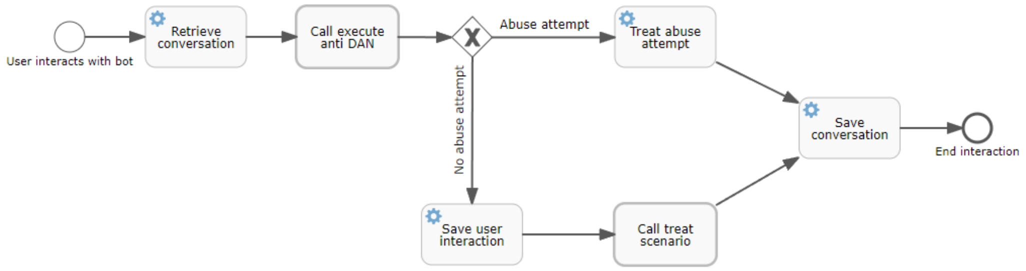


Figura 1.3: Top level flow diagram for the conversation

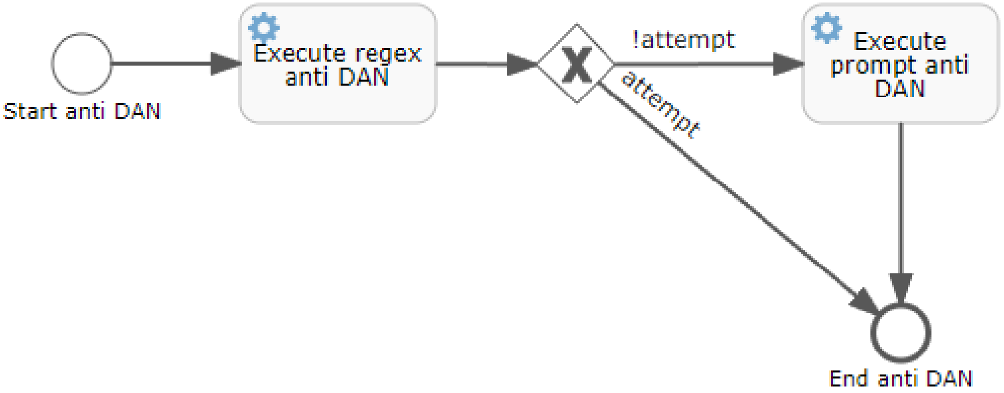


Figura 1.4: Flow diagram for the Anti DAN stage

1.4.4 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 1.4.

The following sections describe the features that this API provides.

1.4.4.1 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

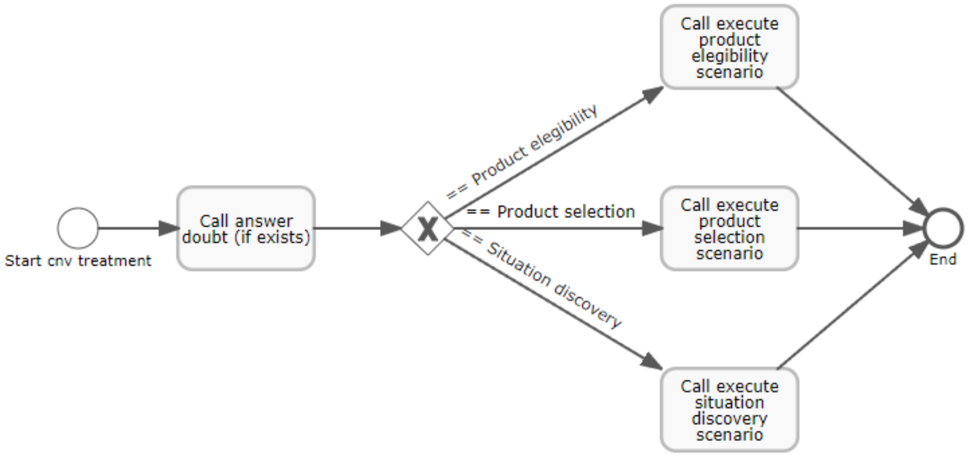

Start cnv treatment
Call answer doubt (if exists)
X
== Product elegibility
== Product selection
== Situation discovery
Call execute product elegibility scenario
Call execute product selection scenario
Call execute situation discovery scenario
End

Figura 1.5: Flow diagram for the Treat Scenario stage

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].

1.4.4.2 Information Retrieval

We do RAG using the LlamaIndex python library. We use a less common RAG algorithm that uses hierarchical embeddings of chunks, allowing us to capture both fine-grained and coarse-grained information. We call this method "Small To Big Retrieval"or STBR.

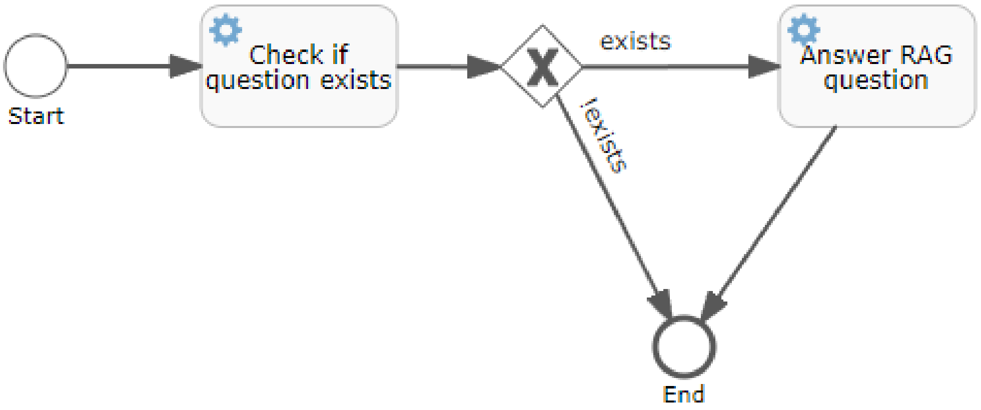


Figura 1.6: Flow diagram for the Answer RAG Question If Exists stage

1.4.5 Our Contributions

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.
- 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.

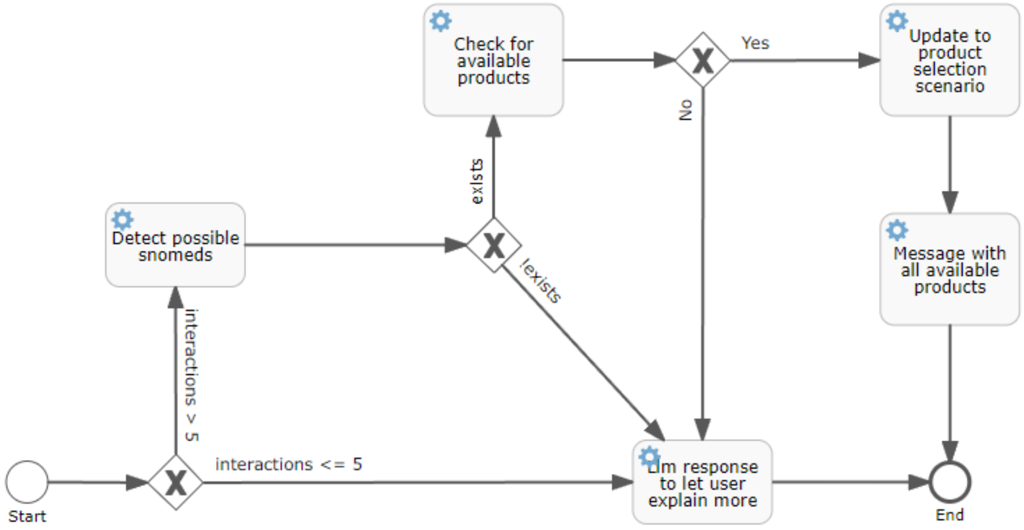

Check for available products
Yes
Update to product selection scenario
No
exists
Detect possible snomeds
!exists
Message with all available products
interactions > 5
interactions <= 5
Llm response to let user explain more
Start
End

Figura 1.7: Flow diagram for the scenario where we need to discover the user's situation

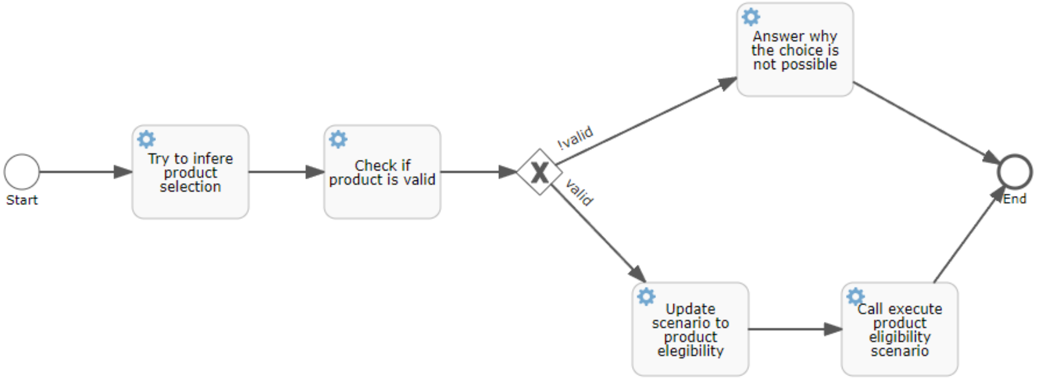

Answer why the choice is not possible
!valid
Start
Try to infere product selection
Check if product is valid
valid
End
Update scenario to product elegibility
Call execute product eligibility scenario

Figura 1.8: Flow diagram for the scenario where we need to select a product from the kSocial catalog given the conversation we have had with the user

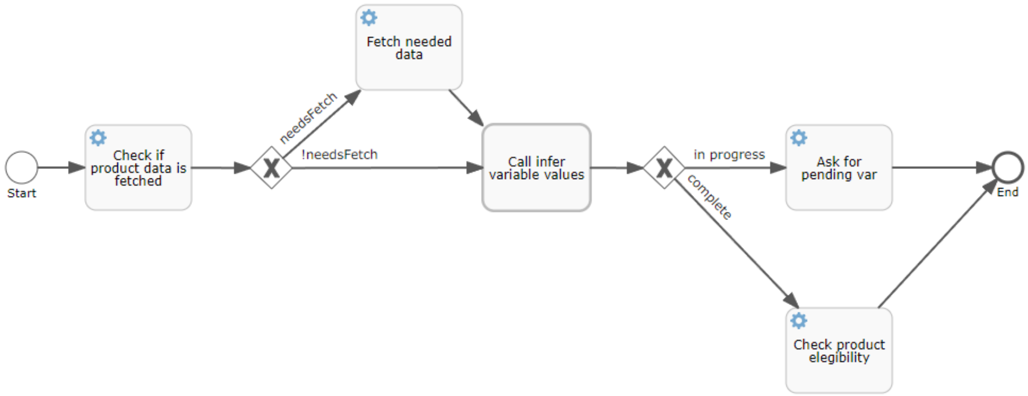


Figura 1.9: Flow diagram for the scenario where we need the user to select one of the products from the kSocial catalog

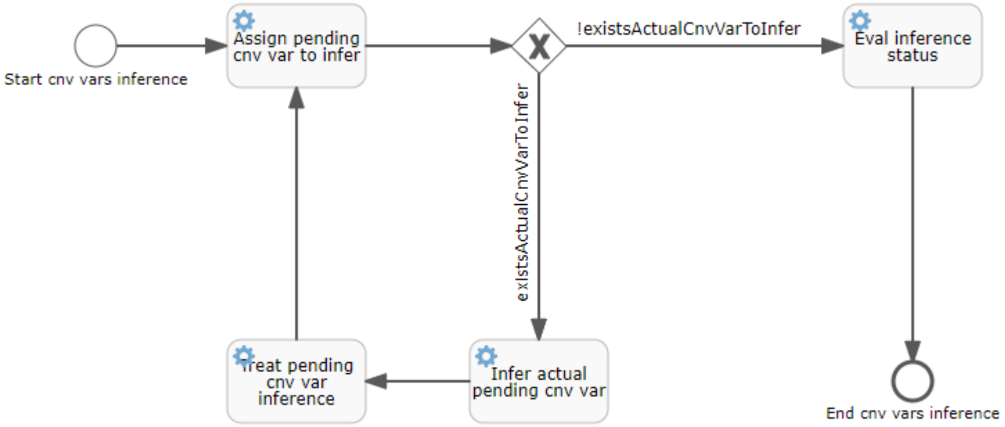
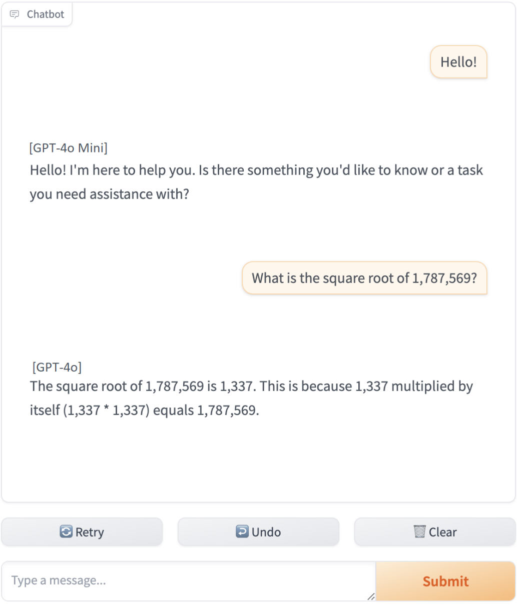

Start cnv vars inference
Assign pending cnv var to infer
!existsActualCnvVarToInfer
existsActualCnvVarToInfer
Eval inference status
Treat pending cnv var inference
Infer actual pending cnv var
End cnv vars inference

Figura 1.10: Flow diagram for inferring necessary variables for the current scenario based on the conversation



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.

Figura 1.11: Example of conversation messages being routed to different models according to their complexity

CAPÍTOL 2

Estat de l'art

2.1 Secció

2.1.1 Subsecció

CAPÍTOL 3
Preliminars

CAPÍTOL 4

Planificació i Metodologia

Capítol 5
Contribució Metodològica

CAPÍTOL 6

Resultats

CAPÍTOL 7

Conclusions i treball futur

Bibliografia

[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, 5 and 6.)

[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 1.)

[3] B. Abushawar and E. Atwell, “ALICE chatbot: Trials and outputs,” *Computación y Sistemas*, vol. 19, 12 2015. (Cited on page 1.)

[4] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” 2014. (Cited on page 1.)

[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 1 and 7.)

[6] A. Radford and K. Narasimhan, “Improving language understanding by generative pre-training,” 2018. (Cited on page 1.)

[7] 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 2.)

[8] J. Zhang, Y. Zhou, and R. Saab, “Post-training quantization for neural networks with provable guarantees,” 2023. (Cited on page 5.)

[9] ONNX Community, “ONNX: Open neural network exchange.” https://onnx.ai, 2024. https://onnx.ai. (Cited on page 5.)

[10] 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 5.)

[11] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015. (Cited on page 5.)

[12] 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 7.)