INTERNSHIP REPORT

Engineering internship at Hoomano

July 2023 - Dec 2023

Valentin Lopez

v1lopezc@enib.fr

→ Internship supervisor: ROUSSEL Kelly

→ Referent teacher: REDOU Pascal

Acknowledgment

My time at Hoomano as an intern was invaluable for my professional development. I had the privilege of working alongside highly skilled and motivated people in an incredibly engaging work environment.

I would like to express my deepest gratitude to Kelly Roussel, AI engineer at Hoomano's team, who was my direct supervisor during this period. I am very grateful for all the technical knowledge and first-hand experience she shared which significantly contributed to my professional growth.

I'd also like to extend my sincere gratitude to Xavier Basset, founder of Hoomano, for welcoming me into his team, and especially for advising me not only during this internship but also for my future endeavors.

Abstract

This paper is a report of the end-of-study internship I carried out as part of my academic training at Brest National School of Engineering in France. It covers the activities I was assigned to during my time as an intern at Hoomano, a French startup building an AI agent for professional use cases. I share my acquired experience and my whole research.

Contents

List of Figures

Glossary

LLM Large Language Model. 7, 8, 15, 17–19, 39

Mojodex Hoomano's AI Agent. 5, 7, 8, 11, 12, 29, 31, 32, 40

PEFT Parameter-Efficient Fine-Tuning. 16, 18

Introduction

I wanted to dive into the world of AI, so I focused my final engineering internship on this field. After discussions with various companies, I ended up choosing Hoomano. They offered me a chance to work on the hottest topic in AI right now, LLMs. In this report, I'll delve into the analysis I conducted for Hoomano, evaluating the feasibility of implementing an LLM for **routing** within their product pipeline, Mojodex. Additionally, I'll explore whether an open-source model could match the performance of OpenAI's GPT models in this same assignment.

In this report, I'll detail my learning journey, covering everything from LLM cloud deployment to prompt engineering techniques, making also an analysis of the drawbacks I encountered as well as the skills I acquired.

To start, let me introduce Hoomano.

Hoomano presentation

Hoomano, a French startup, delves into the AI field to develop its own AI Agent, called Mojodex. The team consists of three key individuals: Xavier Basset, Founder and CEO; Kelly Roussel, AI engineer and CTO; and Cyril Maitrejean, co-founder and CFO.

Taking advantage of the huge revolution we all are living around AI development these last years, especially with LLMs. Hoomano wraps this technology into an out-of-the-box product that allows, entrepreneurs, salespeople, marketing teams, and even freelancers to have access to a personal AI Agent that enhances productivity by managing repetitive daily tasks, granting users more time to focus on crucial aspects of their work.

Mojodex isn't designed to be a generic agent for all purposes, but a tailored solution catering to diverse user needs within Hoomano. Consequently, it's not just an out-of-the-box product, but a high-quality, user-centric solution.

Hoomano works under a truly valuable and noble philosophy: **"To put AI to the service of people, not to replace them"**. Consequently, even though feasible, Mojodex never initiates any action without first obtaining the user's explicit permission.

Corporate Social Responsibility

Relations and working conditions

At Hoomano, there's a genuinely engaging working environment. The small team size allows everyone to be heard and contribute their opinions. They operate dynamically, as is typical in a tech company. Daily meetings are a constant, ensuring the team remains informed about each other's work progress. These meetings also provide a space to discuss individual progress and address any potential obstacles in assignments that might appear.

Moreover, there's a level of schedule flexibility that depends on your progress. This means you have the freedom to manage your time, as long as the work is done.

Consumers-related issues

Hoomano values a close relationship with its clients, crafting products that fit their needs exactly. Transparency and protecting consumer privacy are really important to them. In the field of AI and their product development, they prioritize user data protection, making sure the system stays secure without needing to use user data for customization.

Timeline overview

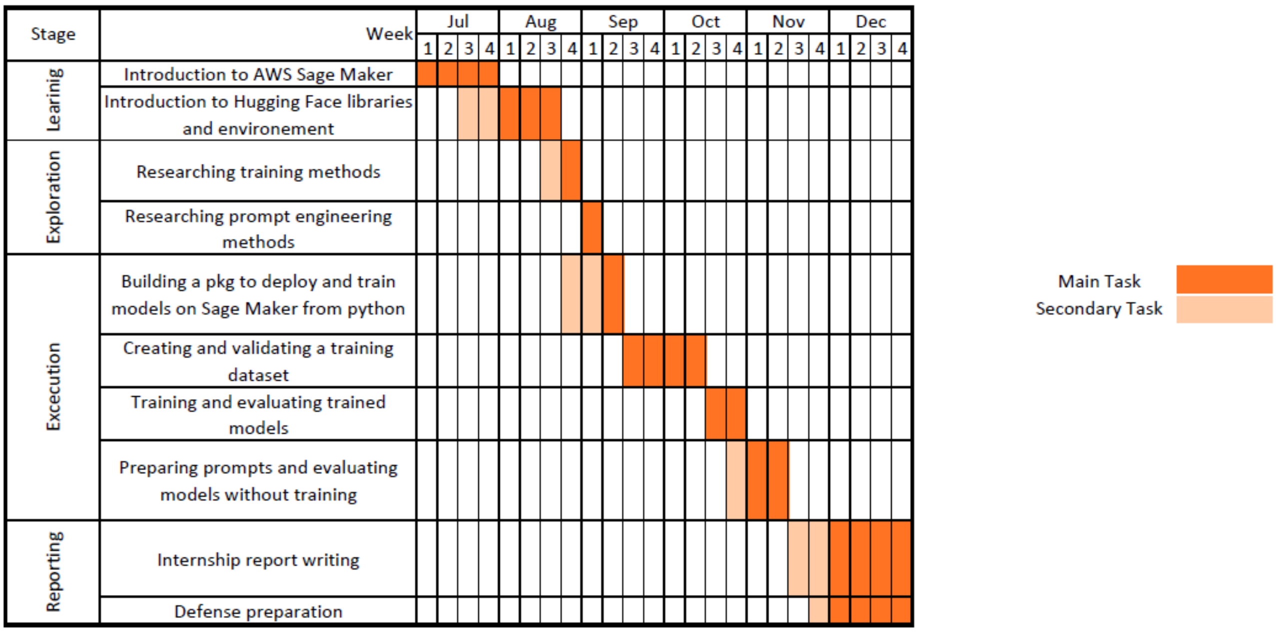

Figure 1: Gantt chart

Tools and frameworks presentation

Mojodex

Hoomano's main product is Mojodex, an AI agent tailored for both large companies and individuals. It comes in web and mobile versions, offering a variety of pre-built tasks aimed at streamlining your workday. Moreover, Mojodex offers real-time suggestions throughout the day and organizes your subsequent steps following the completion of specific tasks.

Mojodex can assist you in preparing meeting recaps, follow-up emails, writing down your ideas in convenient memos, and crafting a LinkedIn post, among other tasks. All interactions take place through a chat interface, similar to ChatGPT, where Mojo prompts you for the necessary information required for the chosen task.

One difference to remark in the mobile app interface is that all the interactions happen through voice for more convenient usage.

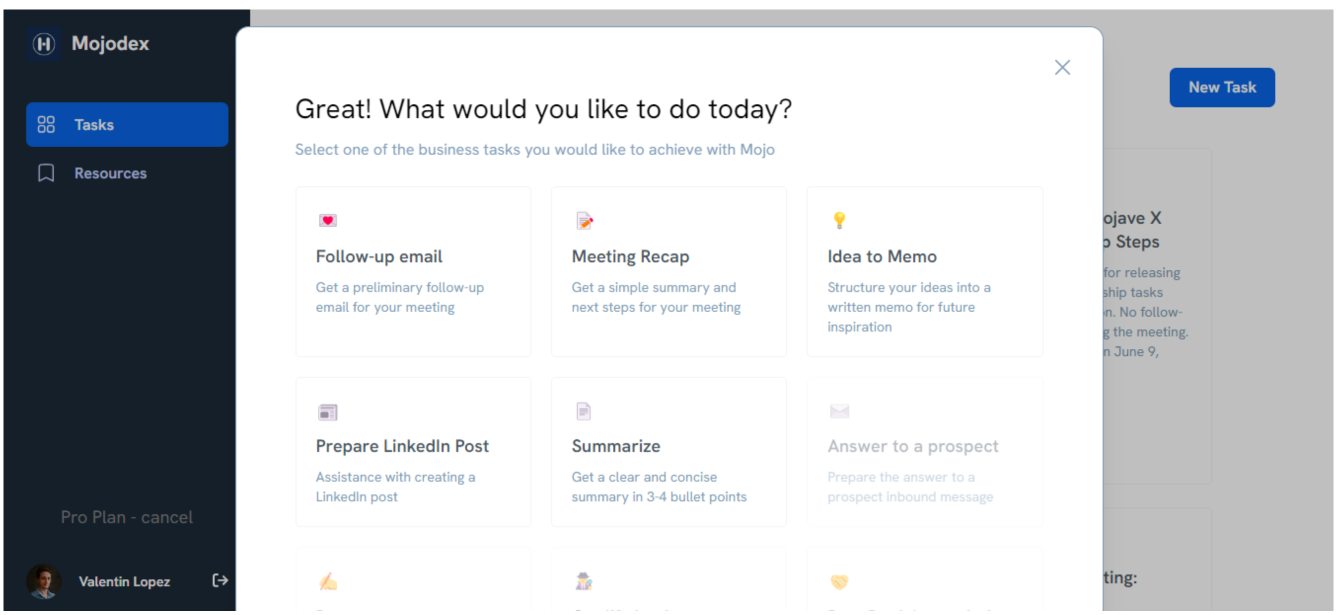

Figure 2: Mojodex web overview

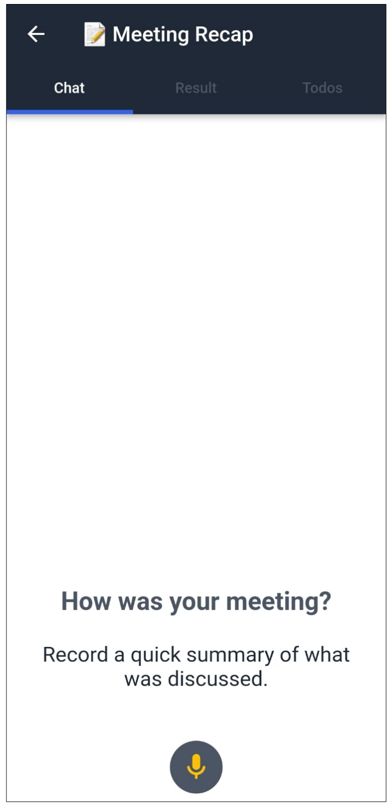

Figure 3: Mojodex mobile app overview

The way Mojodex works is that you have to manually choose the task you want to perform before starting to talk with Mojo itself. But what if you could just start talking to it, and Mojo automatically guesses what you are trying to do among its possible tasks? That would be an interesting feature! It not only helps you go faster but also easily switch between tasks while you're working on something else.

This feature is called **"routing"**, because the system, relying on your inputs, routes Mojo's behavior towards the appropriate task. It is important to notice that this is not a classification problem. The list containing Mojodex's tasks is dynamic, meaning that at any point, a new task could be added, or an old task modified or deleted. **The system should still be capable of selecting the right behavior, regardless of the tasks contained in this list.**

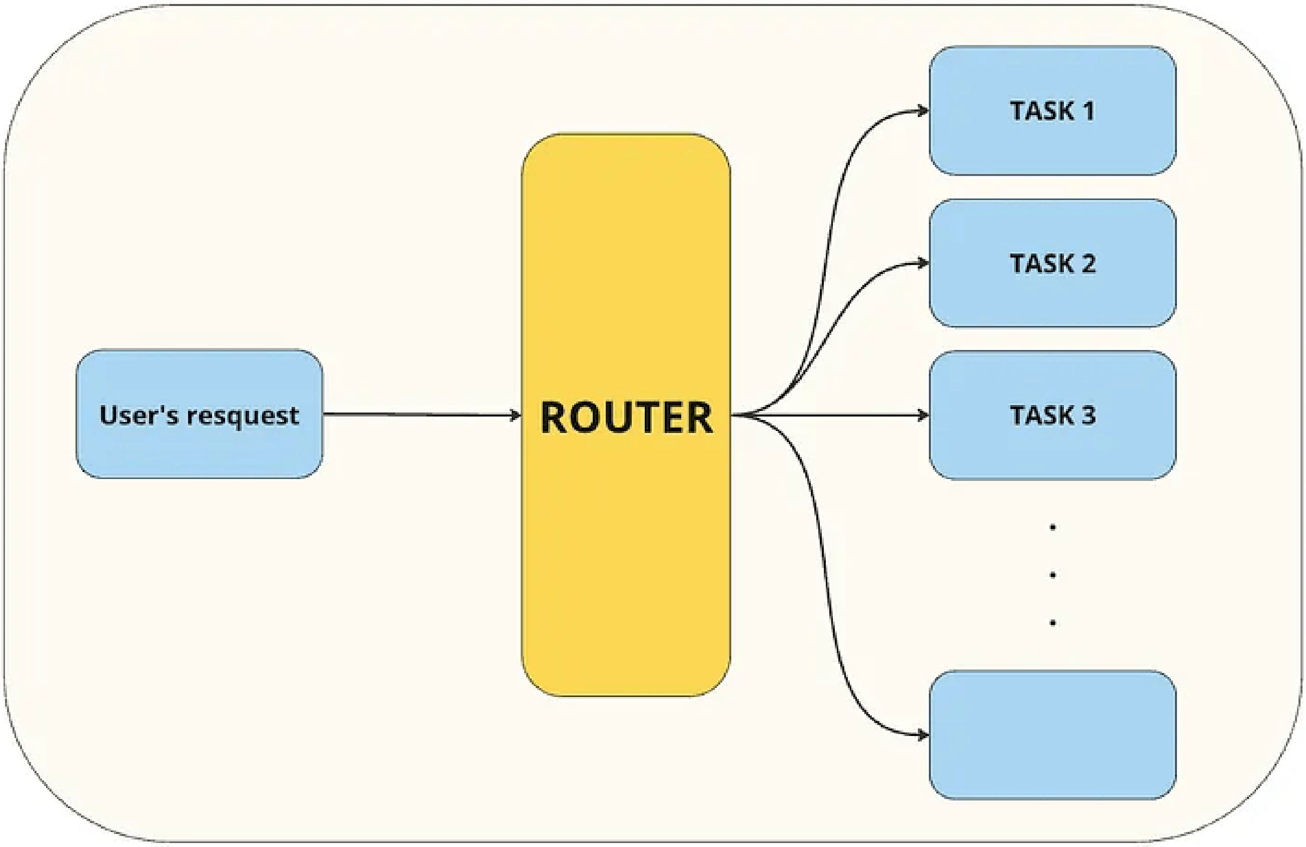

Figure 4: Routing concept

During my initial week, I started by introducing myself to the Mojodex App, understanding its functionalities, and familiarizing myself with the tools and frameworks necessary to accomplish the assigned task.

Amazon Web Service and Sage Maker

Amazon Web Service (AWS) is a well-known online platform, offering scalable and cost-effective cloud computing solutions.

AWS is a broadly adopted cloud platform that offers several on-demand operations like compute power, database storage, content delivery, etc., to help companies scale and grow. One of the main services of AWS is Sage Maker, a whole cloud environment built to train and deploy AI models on the cloud.

Sage Maker has several dedicated spaces. Some of them allow you to use pre-built training methods but it is limited to just some models. The others, and the ones I worked with, give you more flexibility to use a larger number of models and to use training loops of your own.

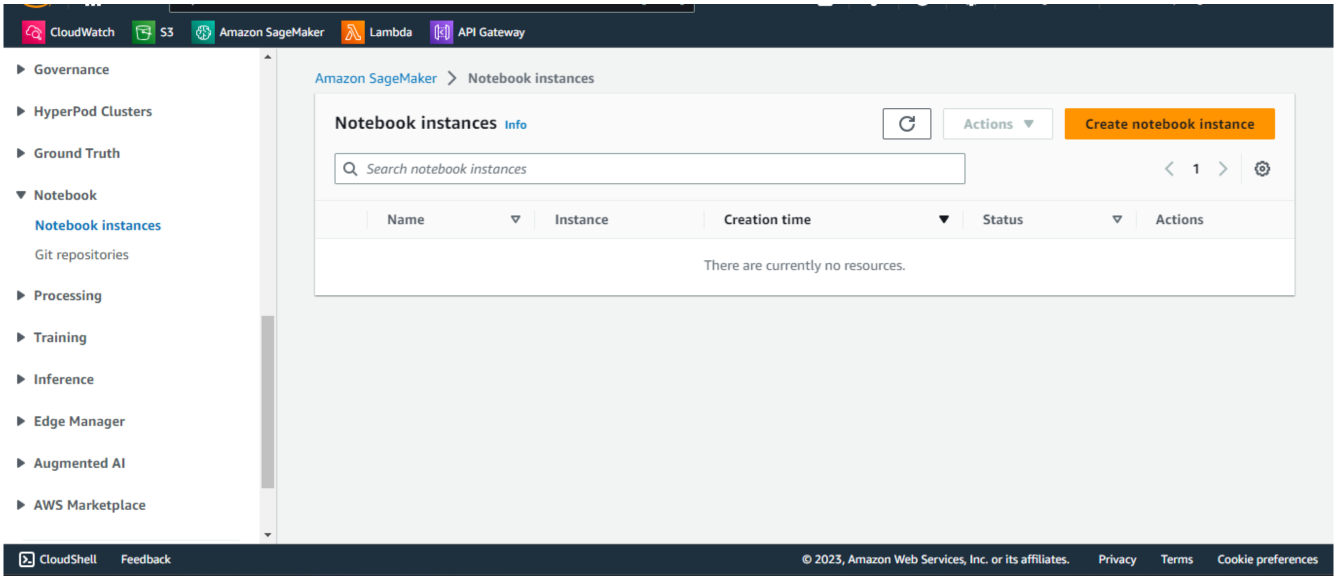

Figure 5: AWS Sage Maker console overview

All the models I tested during my internship came from Hugging Face Hub as well as the framework I used to build the training loops to train the models and deploy them once trained.

As I explained Sage Maker has many different purpose spaces.

Training Jobs

For training purposes, Amazon SageMaker has a service they call **Training Jobs**. A Training Job is a dedicated environment built to upload a model, pass the training data, and customize your own training loop and its hyperparameters in a compact and easy way.

This service can be accessed and launched by AWS's own interface or by a script. In this guide, we will use the script path due to its convenience and simplicity to change the Training Job's configurations. In order to write code on SageMaker the simplest way is through a Jupyter Notebook inside AWS.

Deploying Models

To deploy a model on Sage Maker you have to configure three things first

- **Create a model**

  A model in SageMaker is a space where the model's artifacts and requirements are uploaded and set. You can also define here the environmental variables you'll need. This configuration will be passed to the endpoint through the endpoint's configuration.

- **Create an endpoint configuration**

  The endpoint's configuration is a place where the model created previously is defined to be used on the actual endpoint. At this point, you'll also define the instance type to use, and the number of instances to launch, among other parameters.

- **Create the endpoint**

  Once you have the model set and associated with the endpoint's configuration. You can just create and launch the actual endpoint associating the configuration previously created.

Hugging Face and Transformers library

Hugging Face is a huge online hub where a huge amount of models for a large number of different purposes can be found. All my study and work during those months was based on Large Language Models (LLMs). Models like Llama 2, Falcon, Bart, and the well-known GPT from Openai are examples of them.

The reasons why I chose to work with the hugging face environment were two.

- **The Transformers library**

  The Transformers library is an out-of-box Python library to interacts with models hosted on the Hugging Face Hub or locally on your computer. This library not only provides the flexibility of working with models trained in different frameworks like Tensorflow and PyTorch but also creates custom training loops.

Besides, thanks to other libraries also built by Hugging Face's team, you can implement quantization and PEFT methods which were key things in my analysis.

- **The SageMaker Hugging Face Inference Toolkit**

  Thanks to the SageMaker Hugging Face Inference Toolkit, it is possible to integrate the advantages of Hugging Face's libraries with AWS cloud resources, in a really easy way.

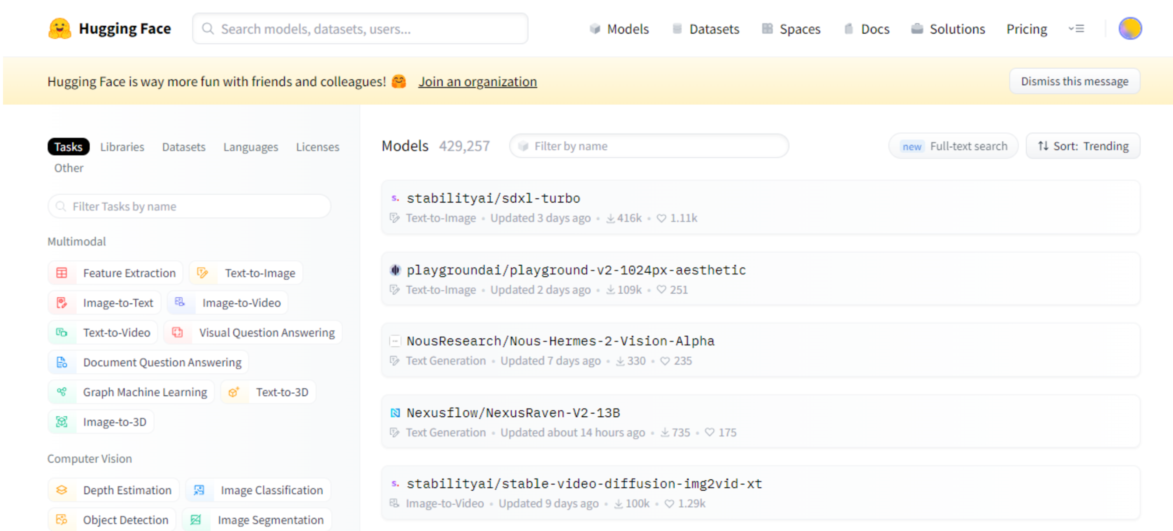

Figure 6: Hugging Face Hub overview

Researching and Training methods

To solve my assigned task there were two paths to follow. Fine-tuning a pre-trained model for this task or making prompt engineering work. I made a study of the two options starting by fine-tuning the model.

Training methods

I began by researching all the possible state-of-the-art training methods for LLMs. This personally allowed me to identify good sources and pages for professional papers. During my final year at ENIB, I had the opportunity to read other papers and gain experience in this area. However, this time, I encountered numerous papers discussing a vast array of new research related to my area of interest. This allowed me to train myself to summarize this wealth of information and adopt a more effective approach to reading these papers. I believe this part of my journey significantly contributed to improving my ability to locate relevant information within papers, enabling me to decide later if it is worth reading in its entirety.

I gained substantial insights into the training methods currently used. Nowadays when someone says training, it's actually talking about fine-tuning a pre-trained model. Training a model from scratch is a resource and time-intensive process that requires a large and diverse dataset to be completed. This initial phase is typically done by companies, institutions, or communities. Subsequently, these models are published on platforms like the Hugging Face Hub, where they can be downloaded and fine-tuned for more specific tasks.

Fine-tuning involves customizing the original model for a specific use case by utilizing a new dataset. This process includes slight modifications to certain parameters of the LLM. The idea is to take advantage of the previous training. That's why this process is truly cheaper and faster than starting from scratch.

Peft methods

I work mainly with **Parameter-Efficient Fine-Tuning (PEFT)** methods. PEFT methods only fine-tune a small number of (extra) model parameters - significantly decreasing computational and storage costs - while yielding performance comparable to a fully fine-tuned model. This makes it more accessible to train and store large language models (LLMs) on consumer hardware.

The most known PEFT methods are:

- Low-Rank Adaptation (LoRA)
- Prompt tuning

LoRA

To make fine-tuning more efficient, LoRA's approach is to represent the weight updates with two smaller matrices (called update matrices) through low-rank decomposition. These new matrices can be trained to adapt to the new data while keeping the overall number of changes low. The original weight matrix remains frozen and doesn't receive any further adjustments. To produce the final results, both the original and the adapted weights are combined.

Prompt tuning

The key idea behind prompt tuning is that prompt tokens have their own parameters that are updated independently. This means you can keep the pre-trained model's parameters frozen, and only update the gradients of the prompt token embeddings.

Training dataset creation

To be able to train a model first of all I needed to have a representative dataset for the **routing** use case. But while Hoomano had access to numerous conversations from current users, their privacy policies prohibited the utilization of this information for such purposes. One potential solution that was discussed involved engaging new users and explicitly seeking their permission to use their data for building this dataset. Unfortunately, due to time constraints, this remained a potential plan for the future.

As an alternative way, I took another path, making my own synthetic dataset. The idea was to use another LLM to generate full conversations between an **user** and their **AI agent**, with each conversation correspondingly matched (labeled) with a task from a list collaboratively created with Hoomano's team.

To accomplish this task first of all I had to craft a prompt to make the LLM behave as intended and generate conversations in a specific format that emulates an actual interaction between a user and their AI Agent. To do that I researched all the most known prompt engineering techniques.

Prompt engineering methods

Prompt engineering is about creating clear and precise instructions for AIs. It's about giving the right words so the AI understands what we want and responds accurately. It is important to use the best words and structure to avoid ambiguity.

I learned a lot about methods that I was completely unaware of. Some of them:

- **Few shots**

  Few-shot prompting can be used as a technique to enable in-context learning where we provide demonstrations in the prompt to steer the model to better performance. The demonstrations serve as conditioning for subsequent examples where we would like the model to generate a response.

Prompt:

```
A "whatpu" is a small, furry animal native to Tanzania. An example of a sentence that uses
the word whatpu is:
We were traveling in Africa and we saw these very cute whatpus.
To do a "farduddle" means to jump up and down really fast. An example of a sentence that uses
the word farduddle is:
```

Output:

```
When we won the game, we all started to farduddle in celebration.
```

Figure 7: Few-shot prompt example

- **Chain of thoughts**

  Chain-of-thought (CoT) prompting enables complex reasoning capabilities through intermediate reasoning steps. You can combine it with few-shot prompting to get better results on more complex tasks that require reasoning before responding.

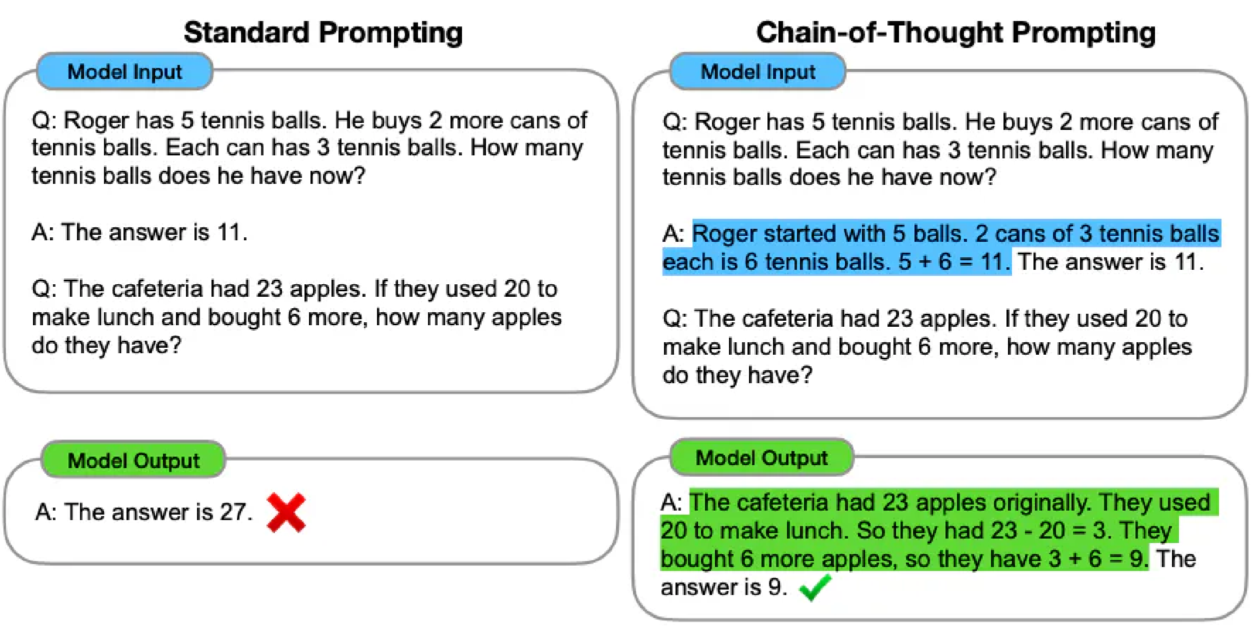

Figure 8: Chain of thoughts prompt example

• **Tree of thoughts**

Tree of Thoughts (ToT) maintains a tree of thoughts, where thoughts represent coherent language sequences that serve as intermediate steps toward solving a problem. This approach enables an LM to self-evaluate the progress intermediate thoughts make toward solving a problem through a deliberate reasoning process. The LM's ability to generate and evaluate thoughts is then combined with search algorithms (e.g., breadth-first search and depth-first search) to enable systematic exploration of thoughts with lookahead and backtracking.

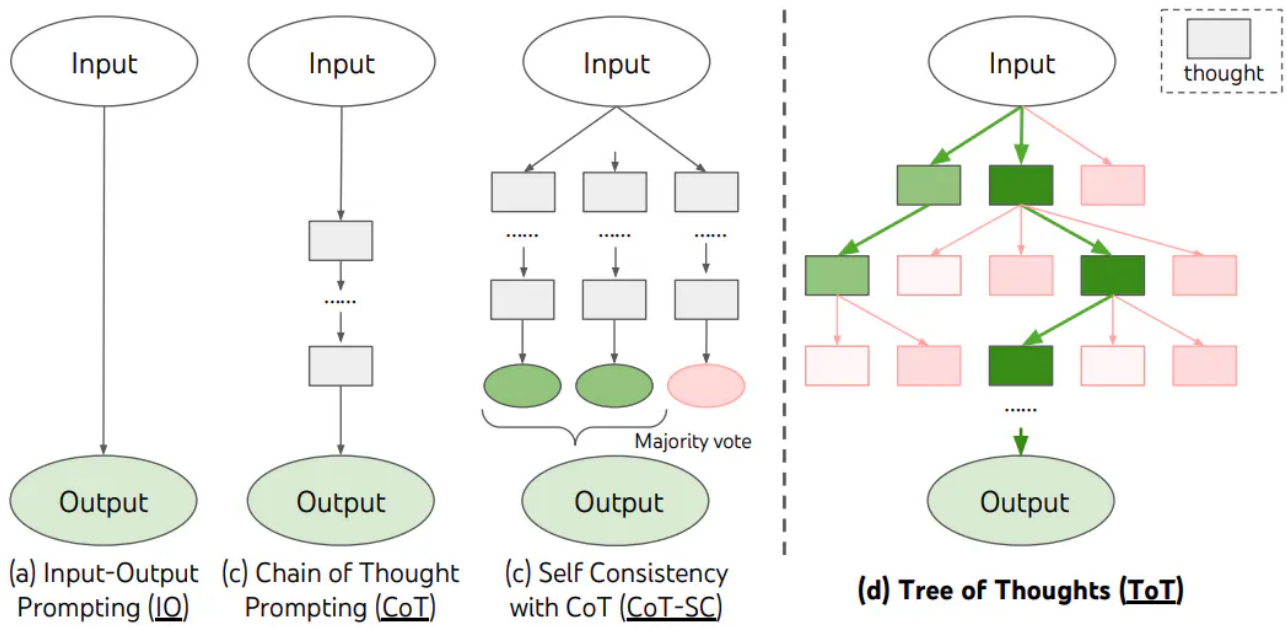

Figure 9: Tree of thought comparison

After testing various techniques with different examples and models I found out that the one that worked the best was the Few-shot technique. I experimented with open-source language models, including Falcon 40 and Llama 2, as well as the closed-source model, GPT-4. Among these, GPT-4 was the one that most respected the instructions I wrote and returned the kind of format I needed in all the examples, so I ended up picking up this model for generating the conversations. Here is the final prompt I used:

```
You are a conversation generator.
```

```
You will be provided the name of a sales task. You will generate a conversation
    between a salesman user and their assistant that will lead to run this task.

[EXAMPLES]
---------
TASK
answer_to_prospect : The user wants to prepare the answer to a prospect inbound
    message.

CONVERSATION
### User: Hey, I got a message from a prospect. What should I answer?
### Assistant: Hi, I'm your assistant. I can help you with that. What's the message
    ?
### User: They are asking for a discount.
-------
TASK
meeting_minutes : The user needs assistance to prepare a meeting minutes

CONVERSATION
### User: I've just finished a meeting.
### Assistant: How did it go?
### User: It was great. We agreed on the next steps.
### Assistant: What are the next steps?
### User: We will send them a proposal.
### Assistant: When will you send it?
### User: Tomorrow.
------
TASK
send_information_email_to_client : The user wants to send information to a client
    by email

CONVERSATION
### User: I've just finished a meeting.
### Assistant: How was it?
### User: Quite exhausting. We had to discuss a lot of details, many of which I
    didn't know. But finally I think I managed to answer all the questions.
### Assistant: Congratulations for that.
### User: I need to inform my client John about the new product.
```

```
-------
[END of EXAMPLES]

The target task is in [TASK] section.
Generate a conversation between a salesman user and their assistant that will lead
    to run this task.
No talk, just conversation.

[TASK]
{{task}} : {{task_description}}
```

Then I input this prompt to create a set of 2000 conversations (data points). Once generated I needed to pass it by a validation process before starting training.

Validation process

Since the dataset was created synthetically, I couldn't ensure the quality of all the data points so I created a simple interface in Python using PyQt5, a framework that I studied during my last year at ENIB. Then I asked for help from Hoomano's team to divide the dataset and complete the validation process faster.

The interface was really simple. the conversation appeared on the left, while the associated task (label) was displayed on the right. Below, there were buttons enabling navigation such as passing, going back, altering the label, or marking conversations for further review or removal from the final dataset.

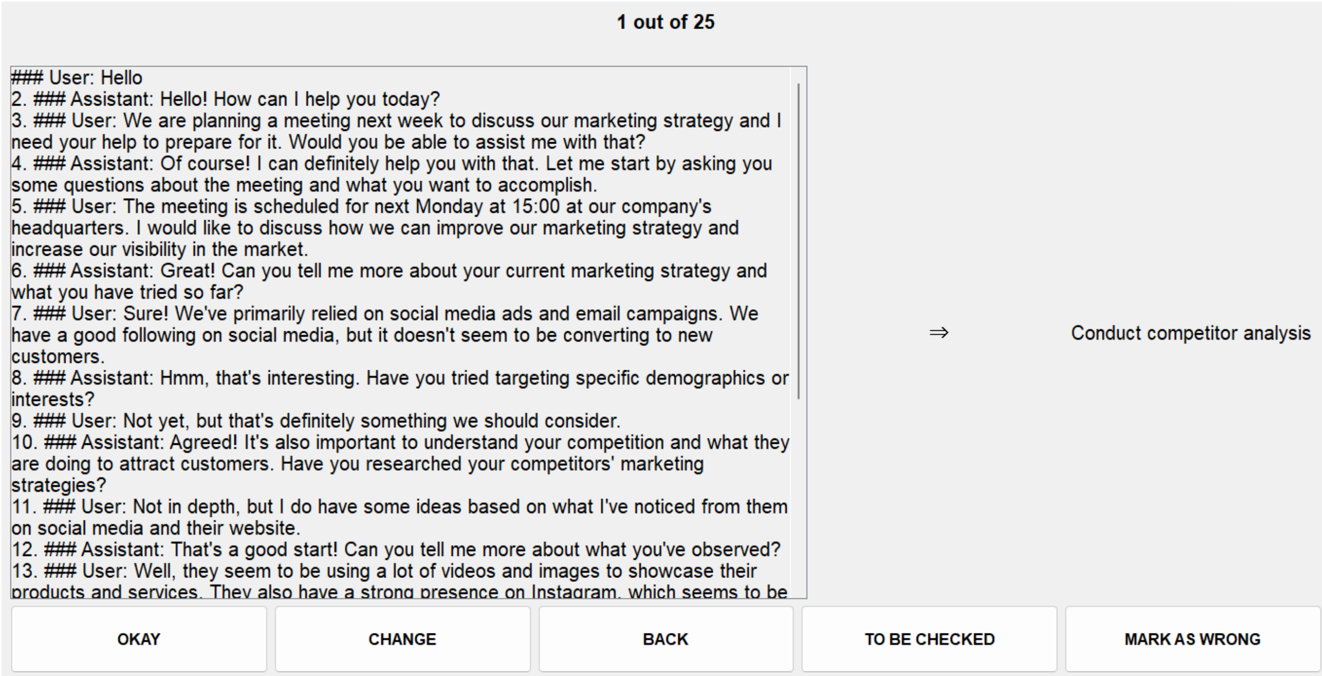

Figure 10: Validation interface overview

This process took two weeks to complete and I thank Hoomano's team for helping me out.

Mojo AI Toolbox

In order to make the work easier and allow someone else can take it up again, I created a Python package designed to streamline the process of training, deploying, and managing machine learning models on cloud platforms such as Amazon SageMaker and Microsoft Azure. This toolbox aims to simplify in a few lines of code the whole process you should do to start using an AI model. It allows to deploy models hosted in Hoomano's hub or HuggingFace's hub, and once deployed use them with a simple **predict** method.

I also prepared documentation to speed up its adoption. Essentially, it serves as a wrapper for OpenAI's transformers and SageMaker libraries, along with the Hugging Face Inference toolkit.

Administrator role

That way, being an administrator at Hoomano you can start a session by calling:

```
from mojo_ai_toolbox import HoomanoAIHub

os.environ["AWS_ACCESS_KEY_ID"] = ""
os.environ["AWS_SECRET_ACCESS_KEY"] = ""
os.environ["ROLE_NAME"] = ""

aws_hub = HoomanoAIHub(
    cloud_service="aws",
    region_name="us-east-1",
    )
```

and then deploy a particular model to Hoomano's AWS endpoints calling the **deploy()** method

Programmer role

Being a Hoomano's programmer who has to work with the already deployed model:

```
from mojo_ai_toolbox import AWSModel

aws_model = AWSModel(
    product="mojodex",          # Pass here the Hoomano's service to use.
    model_id="falcon-7b-fine-tuned",
    task="routing",
    region_name="us-east-1"
    stage="dev",                # You can choose 'dev' or 'prod'.
    api_auth_token="",          # Pass here the API Key.
)
```

Then use the model simply by calling the **predict()** method.

This was actually my first experience in developing a complete package, so it represented a significant learning curve for me. Throughout this process, I encountered various challenges that motivated me to learn and put into practice many new things of which I wasn't previously aware, such as some code structures and credential handling.

Training

For training, I chose to use the QLoRA method. QLoRA states for **Quantization and Low Range Adapters** and it is a variation of the LoRA method presented in a previous section (LoRA).

Train a model is memory costly. That's why in cases like that is a good option to consider quantization. Normally the weights (parameters) of a model are represented in 32 bits, quantifying a model means to represent these weights in other numeric representation that takes less memory. It could be an 8-bit representation or a 4-bit representation (known as 4-bit QLoRA).

As an important remark: just the model's weights are converted, but the LoRA adapter's weights remain in 32-bit representation.

It is easy to realize that the price to pay for saving memory is losing accuracy, but this is not exactly the case. Since we are fine-tuning the LoRA adapters's weights, the value obtained after training takes into account this loss of accuracy so in the end these trained adapters reach a value that compensates for the approximation done during quantization on the model's weights.

Models trained

The main models I worked with were the Llama model family from Meta and the Falcon model family from Technology Innovation Institute (TII). Both are open-source models.

For cost reasons, I picked just two of them to train:

- Llama 2 7b
- Falcon 7b

My goal at then was to compare their results before and after training against other versions of their family and GPT3.5. The process took 8 hours to train each model once. Because of the results obtained, I had to train them twice to test different configurations so it was a 4-day task.

Results

The results were as follows:

First training

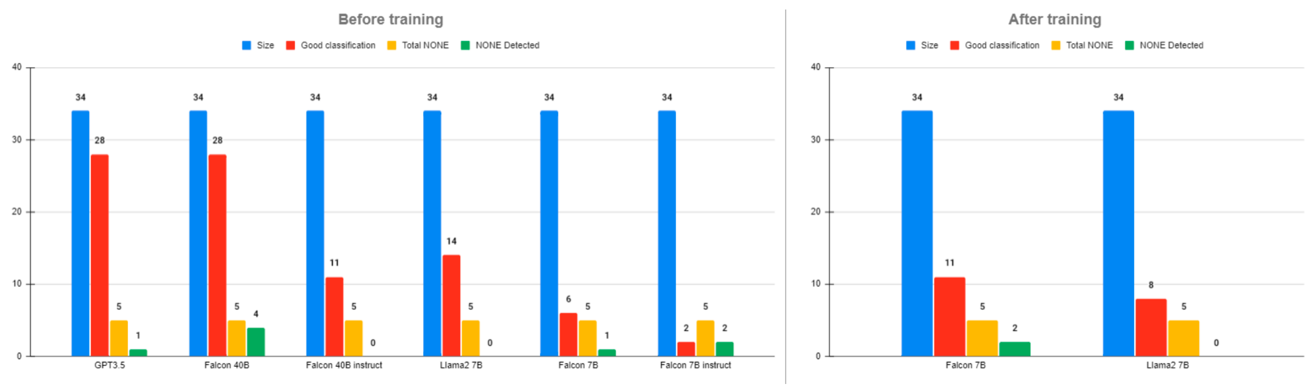

Figure 11: Results before and after the first training

The first training was a 2 epochs training using a training dataset of 955 conversations, evaluated into a test dataset of 34 conversations.

As I mentioned, I trained the 7b version (with 7 billion parameters) of these models. However, there are also 40b and instructed versions available. The displayed results compare these other model's versions from the Falcon family with GPT3.5. As expected, models with a higher number of parameters perform better without any fine-tuning.

Additionally, Falcon 40B presented better performance compared to Falcon 40B Instruct, despite their identical sizes. This is because the word **instruct** means that this model has been trained to follow instructions rather than just to generate text. This difference exists because of the prompt. A prompt craft for the foundation model won't work as well on the Instruct version of the same model.

The first training didn't return good results. After training, the results indicate improvement in Falcon 7B's results but a decline in performance for Llama 2 7B.

Another aspect that showed no improvement was the **NONE Detections**. So far, I have explained the system needed to route the appropriate task based on user conversation. However, during a real interaction, the user might want to do something that is not considered in the tasks list (NONE task). Those cases that were tagged in the analysis as ”NONE Detected”, are the ”NONE task” cases that the system successfully identified.

This wasn't accomplished for almost any model, including GPT3.5, and after training it didn't improve either. Following these results, I changed some parameters and tried a second training.

Second training

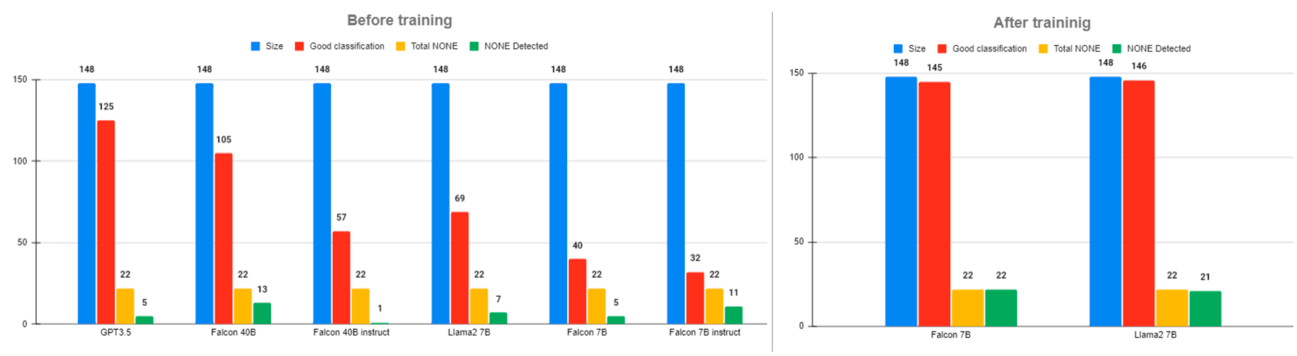

Figure 12: Results before and after the second training

Here, I doubled the number of epochs and increased the training dataset size to 1670 conversations, resulting in significantly improved outcomes. This included enhanced performance in well-routed conversations and better detection of **NONE tasks**.

Unfortunately, there was a considerable risk that overfitting happened but not in the conventional sense. During live tests using Mojodex, when observing the model's selections, the performance declined significantly. While it often made correct selections, it displayed a tendency to not be able to change the choice to another task or to stop detecting it when the task had finished. Such behavior wasn't desired for an application intended for one-on-one conversations.

What was the issue here?

Against expectations, the problem wasn't overfitting in the conventional sense. Normally, overfitting occurs when a model doesn't actually learn to perform the intended task; instead, it memorizes the correct answer for each input. This leads to high accuracy during training but a significantly lower one when analyzing new inputs. This wasn't the case here because the training and test datasets comprised entirely different conversations, and the high results shown in the figure above were from an analysis of the test dataset. Therefore, the model successfully learned to generalize its task to examples it hadn't looked at previously.

The most likely is that the problem lies in the quality of these synthetic conversations. Despite undergoing a validation process, these conversations were artificially created. Consequently, it is possible that the model adapted to this structured conversation style, which didn't fully represent a real conversation with a real human user, where the conversation might be less structured.

So after these conclusions, I started studying the other possible way: **prompt engineering**. But first I started by making some new definitions to analyze the results using some well-known metrics this time.

Prompt engineering path

Measuring points

Firstly, allow me to introduce what we have identified as key points in a typical conversation with the virtual assistant:

- **Starting point:**

  It's the user's input during the chat, which reflects the user's intention to initiate a specific task. For instance, in the context of Mojodex this could be:

```
User: Please help me out to summarize my meeting
```

  This input would prompt Mojodex to initiate the "Meeting Recap" task.

- **Ending point:**

  It's the user's input during the chat, indicating that the ongoing task has been completed. For instance, it could be:

```
User: 'Thats fine, thanks for the help
```

- **Switching point:**

  Depending on the user's requests to complete a task, there might be cases where Mojodex needs to switch to another task in the middle. These situations serve as switching points, as Mojodex switches during the execution of another task. For instance:

```
User: I need to prepare an email for the participant of 'yesterdays "meeting"

Mojodex: Of course, I can help you with that. To prepare the email for
the participant of yesterday's meeting, to start I need the following
information: Who was the participant you had the meeting with yesterday?""
```

```
User: Before giving you that information I would like to create the recap of
the meeting to include it on the email
```

In this example, the last user's input served as a switching point because Mojodex should transition from **"Prepare a follow-up email"** to **"Meeting Recap."**

- **Task selection:**

  Distinguishing when Mojodex should switch from one task to another is one challenge, but selecting the correct task is an entirely different matter. This was the true challenge for us because opting for the wrong task could result in providing an unintended response to the user.

- **Inference time:**

  Last but not least, the inference time is a crucial factor. Even if the model has exceptional accuracy performance, if it takes an eternity -more than 0.3s- to select a basic task, it won't provide a good experience for the end user.

Evaluation

Because of the previous results and in order to measure these new key points, I created a new conversation dataset consisting of discussions that better represent these points. Instead of generating fully synthetic conversations, I instructed GPT to answer as Mojodex, and then I crafted situations that exemplify the measuring points I explained earlier. As a result, the conversations were more realistic this time.

My aim this time was to assess how well the instructed version of these models performed in comparison to GPT-3 when using the same prompts. That said, let's introduce the evaluation process I used.

Regarding the process, I designed multiple prompts using various prompt engineering techniques, including:

  - Zero-Shot

– Few-Shots with Chain of Thoughts

– ReAct

For this evaluation, I selected the same family of models I used before but now I included a new model that was recently released at that moment: Mistral 7B. Each in their instructed versions.

Once I had gathered the necessary data, the models, and the prompts, I passed to the evaluation phase.

The evaluation process involved passing each conversation through each prompt for every model. Subsequently, I calculated a confusion matrix for each combination of model and prompt, enabling me to draw meaningful insights from the results.

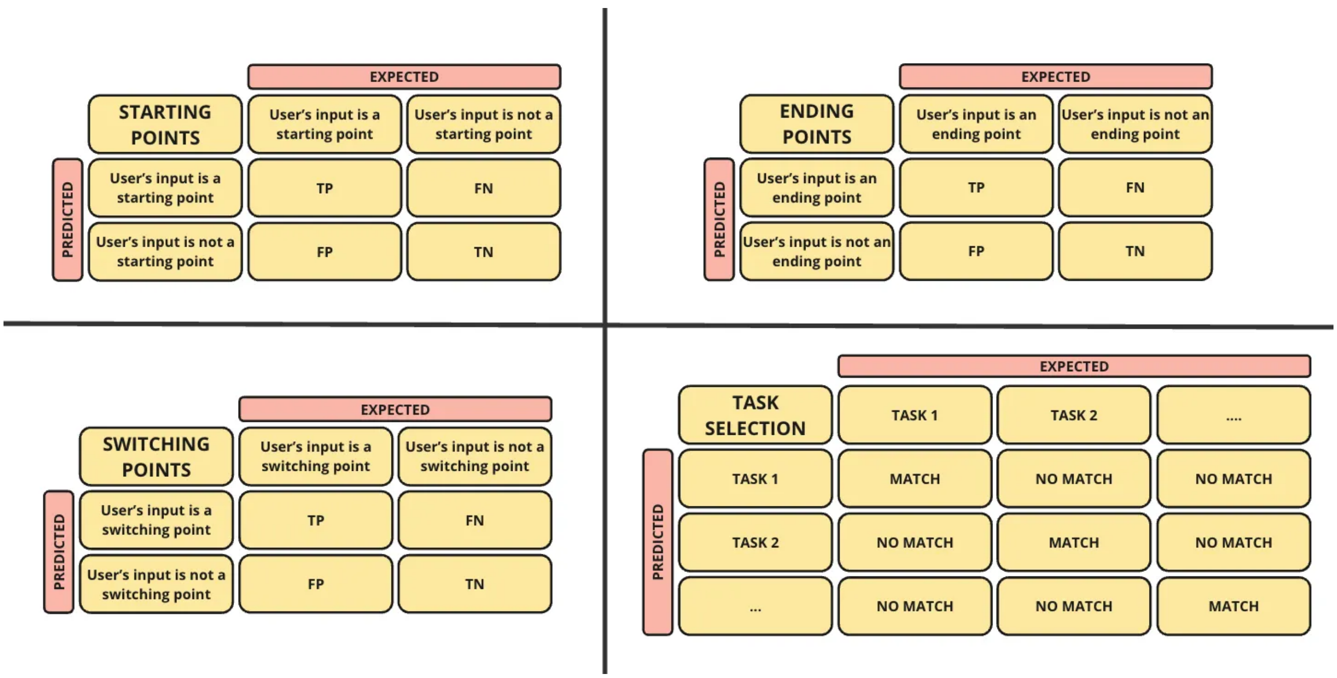

Figure 13: Confusion matrices presentation

Furthermore, using these results, I calculated the following metrics for each combination of model and prompt. This was done to compare both the model's performance and the potential of the prompt engineering methods:

- **Accuracy**: allows me to analyze how good the model and prompt are making the right detection.

$$\frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision**: allows me to analyze how many of the correctly predicted cases actually turned out to be true.

$$\frac{TP}{TP + FP}$$

- **Recall**: allows me to analyze how many of the actual positive cases were predicted correctly.

$$\frac{TP}{TP + FN}$$

Results

This time I got some pretty interesting results. Surprisingly, the open-source models did a really good job at picking the right task and nailing the starting, ending, and switching points.

Now, let's compare Llama 2 chat 70B, the open-source champs in this evaluation, to GPT-3.

Most models performed best when I used the ReAct technique for the prompt, so those results are presented as follows:

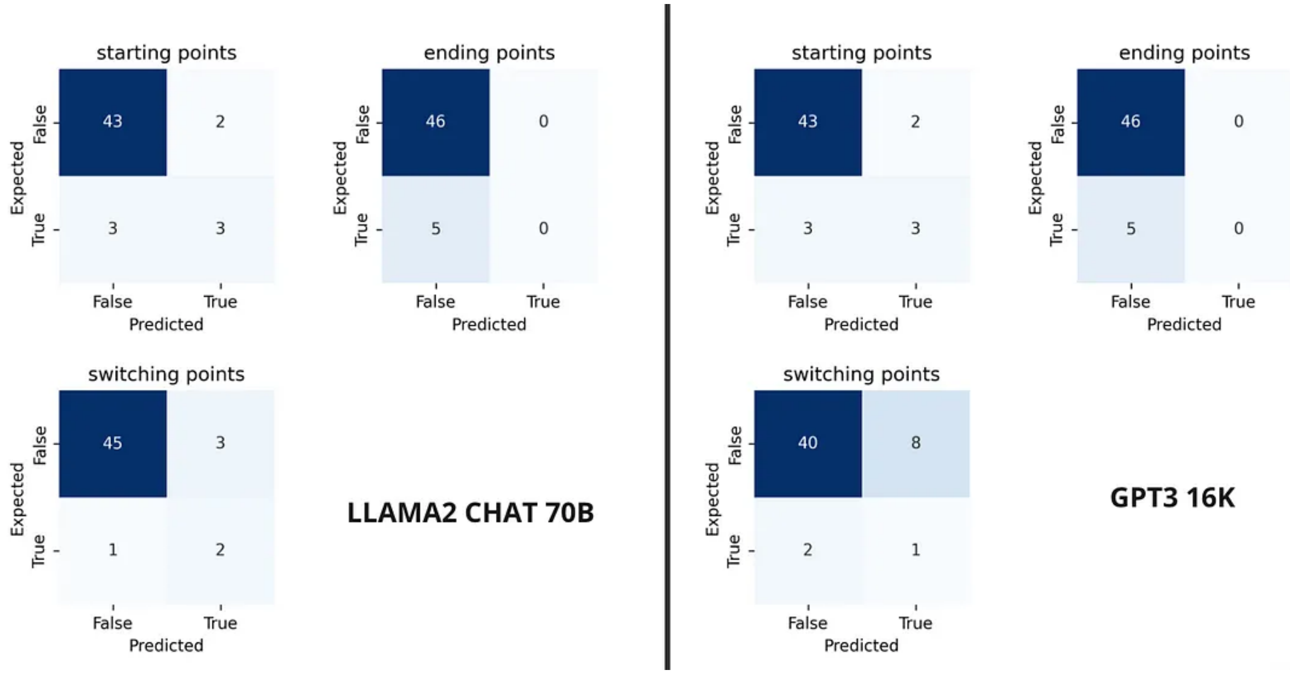

Figure 14: Confusion matrices: Llama2 vs GPT3

LLAMA 2 CHAT 70B			
/	Starting Points	Ending Points	Switching Points
Accuracy	90.20%	90.20%	92.16%
Precision	60%	0%	40%
Recall	50%	0%	66.67%

GPT3 16K			
/	Starting Points	Ending Points	Switching Points
Accuracy	90.20%	90.20%	80.39%
Precision	60%	0%	11.11%
Recall	50%	0%	33.37%

Figure 15: Metrics results: Llama2 vs GPT3

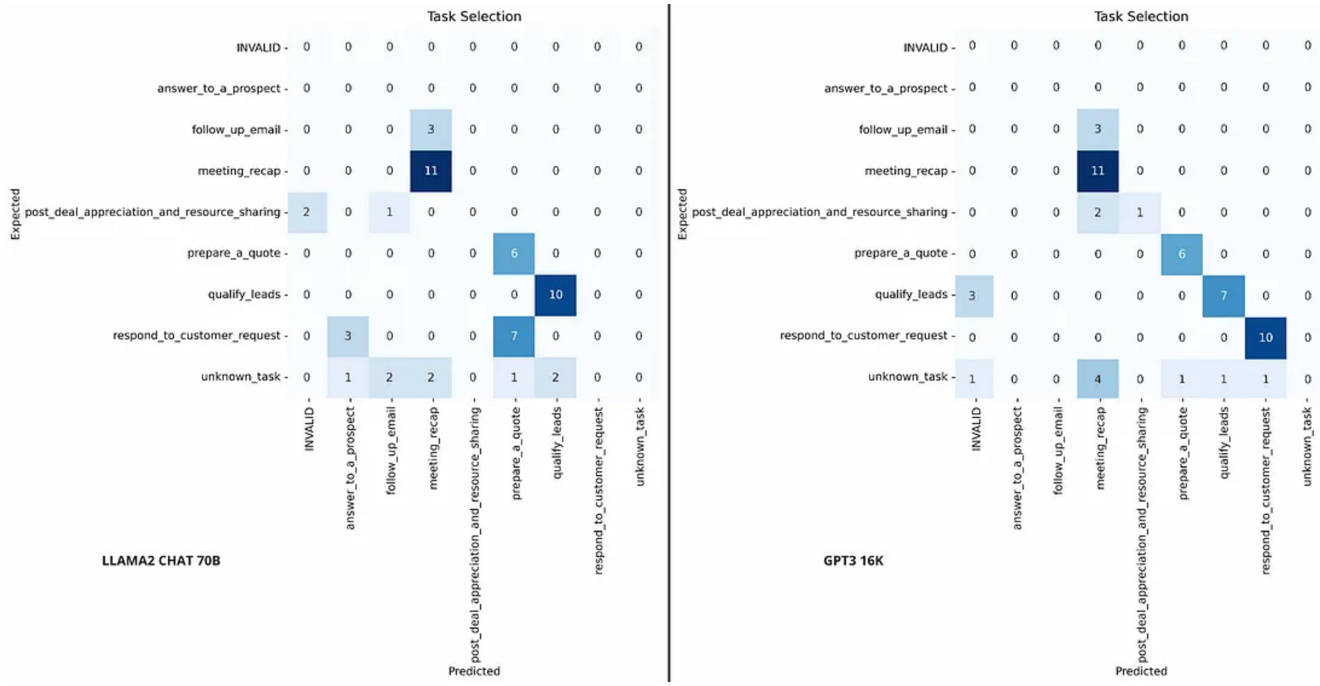

Figure 16: Task routing confusion matrix: Llama2 vs GPT3

Task well predicted		
/	LLAMA 2 CHAT 70B	GPT3 16K
Accuracy	52.94%	68.63%
Precision	36.21%	62.81%
Recall	52.94%	68.63%

Figure 17: Task routing results: Llama2 vs GPT3

As far as the results go, it appears that I can depend on open-source models and still achieve an acceptable level of performance. However, there's more to consider. While the open-source models did reasonably well in terms of predictions, their response times vary significantly from models like GPT-3, and this presents a significant challenge.

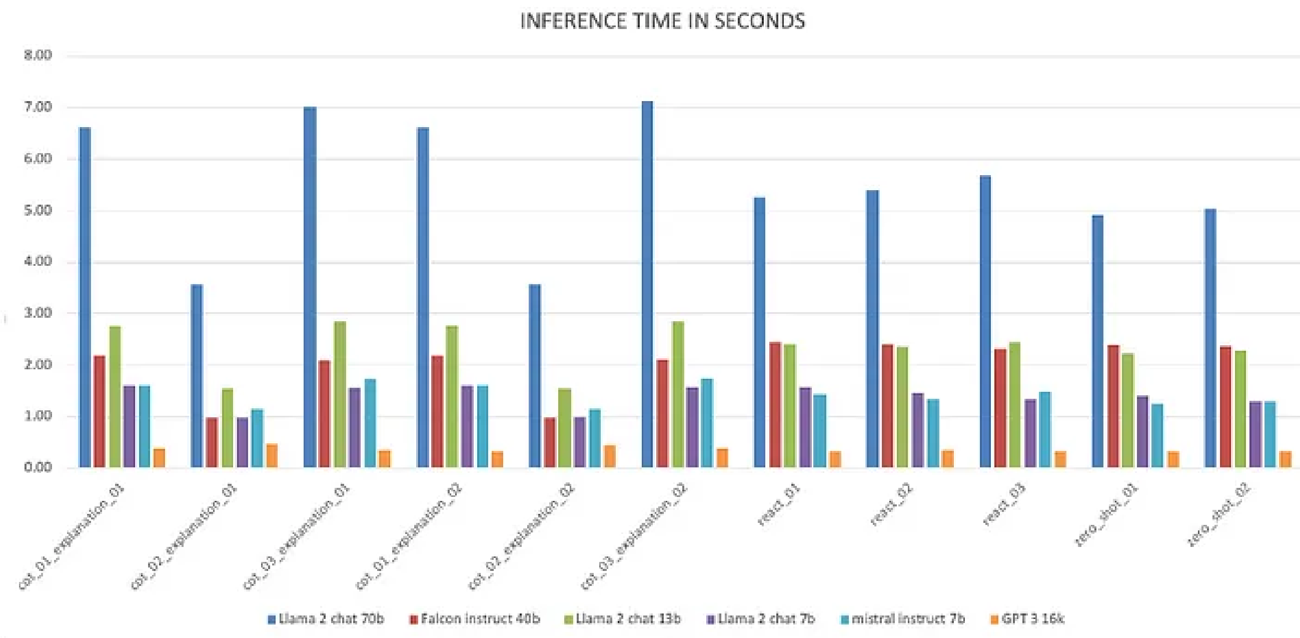

Figure 18: Inference time comparison

In terms of prediction results, it’s clear that Llama 2 70B delivered the best results among all the other open-source models. However, its inference time is six times longer than GPT-3’s — above 2s, way beyond what I need for our routing use case.

An important point to note is that I discovered Mistral 7B produced results that were on par with Llama 2 70B but with significantly faster inference times less than 1s. Yet, it wasn’t crowned the winner for a key reason that I hadn’t mentioned before: the output format. One of the significant challenges with open-source models is ensuring the correct output format. In Hoomano’s use case, the model was expected to return only the name of the selected task, but in many instances, it failed to do so.

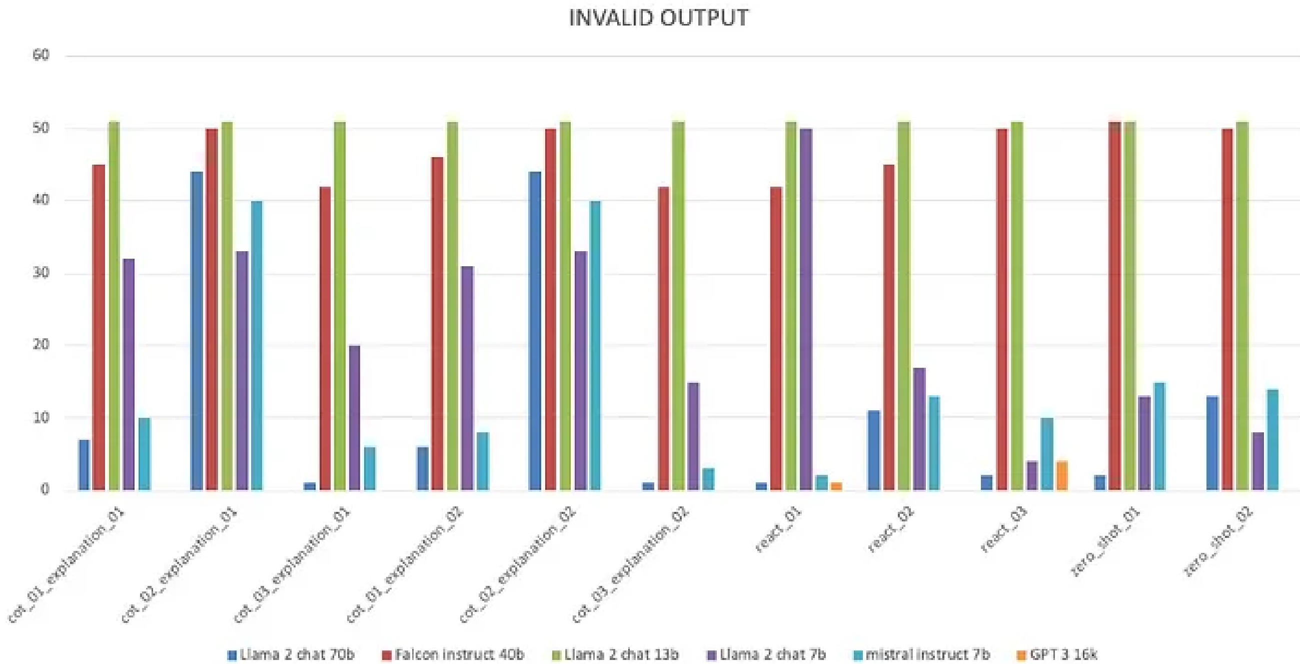

Figure 19: Invalid outputs obtained using different prompts engineering techniques

In the image above, we can observe that in nearly all cases, Llama 2 70B outperformed Mistral 7B in this regard. It's a reasonable assumption to make that this discrepancy is due to the vast difference in model size, with Mistral being a 7 billion parameter model and Llama a 70 billion parameter one.

Conclusions

As a conclusion of my two analyses, I've concluded that the best solution for the routing is to use GPT3.5 or GPT4, using a prompt engineering approach. Switching to an open-source alternative is not a viable option at the moment, primarily due to the significant difference in inference time. Nonetheless, this last evaluation also underscores that open-source LLMs could be a good alternative, getting closer to proprietary solutions' performance. It's clear that, with time, open-source models will likely catch up, with Mistral as a good example.

The primary drawback of the fine-tuning approach lies in the absence of high-quality data suitable for Hoomano's specific use case. Acquiring such data would significantly improve the viability of retrying the fine-tuning approach on a model.

On a personal side, I consider that my time at Hoomano allowed me to put into practice several skills that I learned over my career journey from its beginnings in Argentina to its culmination in France. It showed me how a truly dynamic team works and how to handle the real-world demands that not only change but also change quickly. At times, a quick response is necessary to keep your product or service on the right track, even if it is not perfect. I learned how to prioritize real user needs over minor performance details.

It also provided me with the opportunity to introduce myself, deeply to the AI world, a new whole revolution full of new applications and technologies that are reshaping how we work and learn. I find it particularly crucial to stay informed about this field not only because it passionately interests me but also because the more I understand it the more I know how to use it and how to take profit out of it.

This experience helped me solidify my future aspirations and opened the door for the possibility of working with a team that possesses an engaging dynamic and a clear vision.

Future development

Mojodex is an ever-evolving product that adapts daily to meet user needs. At Hoomano, they are well aware of this and work hard to keep it on the right path. Currently, Mojodex assists with simple tasks, prompting users for work-related information. However, its capabilities extend further, such as interpreting emotions and reacting accordingly. This aspect was initially part of my internship project, but due to time constraints, it wasn't feasible.

Mojodex's potential relies on rapidly evolving technologies that have shown no signs of slowing down. So, Hoomano's team remains open to integrating new releases into Mojodex if they enhance the user experience.

Bibliography

Sage Maker Documentation https://huggingface.co/docs/transformers/index

Transformer Library Documentation https://huggingface.co/docs/transformers/index

QLoRA: Efficient Finetuning of Quantized LLMs https://arxiv.org/abs/2305.14314

Tree of Thoughts: Deliberate Problem Solving with Large Language Models https://arxiv.org/abs/2305.10601

Chain-of-Thought Prompting Elicits Reasoning in Large Language Models https://arxiv.org/abs/2201.11903

Prompt Engineering Techniques https://www.promptingguide.ai/

Llama 2: Open Foundation and Fine-Tuned Chat Models https://scontent-cdg4-1.xx.fbcdn.net/v/t39.2365-6/10000000_662098952474184_2584067087619170692_n.pdf?_nc_cat=105&ccb=1-7&_nc_sid=3c67a6&_nc_ohc=4mv90LTCsgEAX8YoQCt&_nc_ht=scontent-cdg4-1.xx&oh=00_AfCcuIaQnep
oe=657C3EFF

LLMs for Your Use Case: A Simple Evaluation Approach https://blog.hoomano.com/llms-for-your-use-case-a-simple-evaluation-approach-dbae6cb51a0d