DEPARTMENT OF
COMPUTER SCIENCE

TIAGO VALENTIM HENRIQUES
Bachelor in Computer Science and Engineering

LEVERAGING LARGE LANGUAGE MODELS TO AUTOMATE PROCESSES

BUILDING AUTOMATED PROCESSES FOR SKILLS WORKFLOW USING A LARGE LANGUAGE MODEL

Dissertation Plan
MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
February, 2024

DEPARTMENT OF
COMPUTER SCIENCE

LEVERAGING LARGE LANGUAGE MODELS TO AUTOMATE PROCESSES

BUILDING AUTOMATED PROCESSES FOR SKILLS WORKFLOW USING A LARGE LANGUAGE MODEL

TIAGO VALENTIM HENRIQUES

Bachelor in Computer Science and Engineering

Adviser: João Vieira
COO, Skills Workflow

Co-adviser: Sérgio Duarte
Assistant Professor, NOVA University Lisbon

Dissertation Plan
MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
February, 2024

Abstract

The evolution of programming paradigms has taught us to simplify processes by introducing solutions that increase and facilitate their use, such as visual languages. Likewise, in software applications, a solution is always sought to benefit the user. This study then focuses on automating processes on the Skills Workflow platform taking advantage of the use of Large Language Models (LLM). Using this emerging artificial intelligence technology, the aim is to provide a better user experience, presenting ways to increase productivity in software applications, in our case in Skills Workflow, automating tasks and providing contextualized responses to users, developing what we call an assistant. The Skills Workflow platform already has a vast marketplace and a User-friendly interface with a *No-code* philosophy, making it perfect for this objective.

In addition to addressing the challenges of understanding user intent or avoiding potential user frustration by taking advantage of LLM features, we intend to investigate potential benefits, including improving accessibility and usability, increasing efficiency and productivity, and perhaps even reducing production costs in companies that use the Skills Workflow platform. The dissertation explores the user's needs and difficulties on the platform, identifying the features and actions that the assistant can perform, such as generating alerts, building dashboards, searching and entering data, or creating and managing projects, tasks, and budgets. The study involves selecting an appropriate LLM, integrating it with the assistant, and identifying and exploring automation possibilities for users, for existing workflows in Skills Workflow.

In this way, we improve and deepen the interaction between users and applications, providing a recipe for reasoning how to build assistants with LLMs. Users must be able to use the platform's resources by chatting with it and expressing their ideas through natural language or voice. Once the system interprets the request, it must simply orchestrate the necessary operations.

Keywords: Integration of large language models (LLM), No-code, AI Assitant.

Resumo

A evolução dos paradigmas de programação ensinou-nos a simplificar processos introduzindo soluções que aumentam e facilitam a sua utilização, como as linguagens visuais. Da mesma forma, nas aplicações de software procura-se uma solução em benefício sempre do utilizador. Este estudo foca-se então em automatizar processos na plataforma de Skills Workflow aproveitando o uso dos Large Language Models (LLM). Pretende-se, com o uso desta tecnologia emergente de inteligência artificial provar possível melhor a experiência do utilizador, apresentando formas para aumentar a produtividade em aplicações de software, no nosso caso em Skills Workflow, automatizando tarefas e fornecendo respostas contextualizadas aos utilizadores, desenvolvendo o que denominamos assistente. A plataforma Skills Workflow já possui um vasto marketplace e uma interface User-friendly com uma filosofia *No-code*, tornando-a perfeita para este objetivo.

Além de irmos abordar os desafios de como entender a intenção do utilizador ou evitar possíveis frustrações do utilizador aproveitando as vantagens dos recursos do LLM, pretende-se investigar potenciais benefícios, incluindo melhorar a acessibilidade e a usabilidade, aumentar a eficiência e a produtividade, e talvez até reduzir custos de produções nas empresas que usam a plataforma de Skills Workflow. A dissertação explora as necessidades e dificuldades do utilizador na plataforma, identificando as funcionalidades e ações que o assistente poderá realizar, como gerar alertas, construir dashboards, pesquisar e introduzir dados, ou criar e gerir projetos, tarefas e orçamentos. O estudo envolve a seleção de um LLM apropriado, integrando-o ao assistente, identificando e explorando recursos de possibilidade de automação para os utilizadores, para fluxos de trabalho existentes em Skills Workflow.

Desta forma, melhoramos e aprofundamos a interação entre utilizadores e aplicações, fornecendo uma receita para raciocinar como construir assistentes com LLMs. Os utilizadores devem poder utilizar os recursos da plataforma conversando com ela e expressando as suas ideias por meio de linguagem natural ou voz. Depois que o sistema interpreta a solicitação, ele deve simplesmente orquestrar as operações necessárias.

Palavras-chave: Integração de Modelos de Linguagem, No-code, Assitente de IA.

Contents

List of Figures **v**

List of Tables **vi**

Glossary **vii**

Acronyms **ix**

1 Introduction **1**

1.1 Context 2

1.2 Objective 3

1.3 Contributions 4

1.4 Document structure 4

2 Related Work Vs. Background **5**

2.1 What is an Assistant? 5

2.1.1 Assistant developed with Large Language Models 6

2.2 Skills Workflow 6

2.3 Large Language Models (LLMs) 7

2.3.1 Open-source vs proprietary models 8

2.3.2 Vulnerabilities of LLMs 8

2.3.3 Fine-tuning a LLM 8

2.3.4 Prompt engineering with LLMs 9

2.3.5 OpenAI models 9

2.3.6 Google models 10

2.3.7 META models 11

2.3.8 Models Comparation 11

2.3.9 Experiments with LLM 14

2.4 Application areas of the assistants 15

2.5 Solution to build an assistant for Skills Workflow 17

2.5.1 Overview 17
2.5.2 Building an assistant using Azure Semantic Kernel 18
2.5.3 Azure Bot 21
2.5.4 Google Dialogflow 21
2.5.5 Distributed Assistant 21
2.6 Discussion 21

3 Solution Proposal **23**
3.1 Previous work 23
3.1.1 Framing the architecture with use cases 24
3.1.2 Data Analysis use case 24
3.1.3 Dashboard Creation use case 25
3.1.4 Problems Notification use case 26
3.1.5 Summarize Project Information use case 27
3.1.6 Summarize Chat Talks use case 28
3.1.7 Project Task Creation use case 28
3.1.8 Visual Data Presentation use case 29
3.1.9 Resource Profiling use case 29
3.1.10 Allocation of People use case 30
3.1.11 Generate Tasks use case 31
3.1.12 Materials Extraction use case 31
3.1.13 Search Data from Images use case 32
3.1.14 Digest Data From Images use case 33
3.1.15 Project Creation use case 33
3.2 Execution plan 34

Bibliography **35**

List of Figures

2.1 Skills Workflow example. 6
2.2 How an assistant can be integrated into the platform. 18
2.3 Semantic Kernel architecture. 19

3.1 Use cases pyramid. 23
3.2 Sequence diagram for data analysis use case. 25
3.3 Sequence diagram for dashboard creation use case. 26
3.4 Sequence diagram for problems notification use case. 27
3.5 Sequence diagram for summarizing project information use case. 27
3.6 Sequence diagram for summarizing chat talks use case. 28
3.7 Sequence diagram for project task creation use case. 29
3.8 Sequence diagram for visual data presentation use case. 29
3.9 Sequence diagram for resource profiling use case. 30
3.10 Sequence diagram for allocation of people use case. 31
3.11 Sequence diagram for generating tasks use case. 31
3.12 Sequence diagram for materials extraction use case. 32
3.13 Sequence diagram for search data from the image use case. 32
3.14 Sequence diagram for digesting data from the image use case. 33
3.15 Sequence diagram for project creation use case. 33
3.16 Solution Roadmap. 34

List of Tables

2.1 LLM performance on text benchmarks. 12
2.2 LLM performance on image benchmarks. 13
2.3 Pricing comparison of LLMs. 14
2.4 General characteristics of LLM assistants. 17

Glossary

Caixa Geral de Depósitos sort	Caixa Geral de Depósitos is a Portuguese state-owned banking corporation and the second largest bank in Portugal. *(pp. 16, 17)*
Be My Eyes	Application that connects blind and low-vision people with sighted volunteers and company representatives for visual assistance through a live video call. *(pp. 16, 17)*
ChatGPT	ChatGPT is a chatbot developed by OpenAI, it enables users to refine and steer a conversation towards a desired length, format, style, level of detail, and language. *(p. 1)*
Integrated Development Environment	Integrated Development Environment is a software application that provides comprehensive facilities for software development. *(p. 1)*
Legacy Systems	Legacy Systems resists modifications and changes while remaining of significant value for its owner. *(p. 2)*
No-code	Software development approach that uses graphical interfaces to generate code, such as drag-and-drop tools as in this case. *(pp. i, ii, 2)*

Skills Workflow	Skills Workflow is a platform that allows the creation of micro-applications and solutions for various domains without the need for advanced programming knowledge. *(pp. i, ii, 2–4, 6, 10, 16, 23, 28, 29, 34)*
Stripe	Empowering the payments of both small and large businesses across the internet. *(pp. 16, 17)*
Transformers	Deep learning architecture that relies on the parallel multi-head attention mechanism. *(p. 7)*
User-friendly	User-friendly is something easy to learn, use, or understand related to a computer. *(pp. i, ii)*

Acronyms

AI	Artificial Intelligence *(pp. 1, 2, 10, 18–21)*
CSV	Comma-separated Values *(p. 24)*
DB	Database *(pp. 15, 32)*
GPT	Generative Pre-trained Transformer *(pp. 9, 12, 16, 17, 22)*
JSON	JavaScript Object Notation *(pp. 6, 15, 24, 31, 33)*
LLM	Large Language Model *(pp. i, ii, 1–9, 11, 12, 15–21, 23–31, 33, 34)*

1 Introduction

For some time now, people have been trying to simplify programming, using metaphors and visual languages so that anyone can program or make the entire creation process faster. Initially, any programming tasks were done through command lines, and then user interfaces began to appear with icons and structured menus. In parallel, programming has been simplified through visual languages that are extremely accessible to everyone, but also through Integrated Development Environments (Integrated Development Environments) that seek to detect errors, whether in syntax highlighting or code completion.

Artificial intelligence (AI) is also a solution to these issues. First, personal assistants such as Apple's *Siri* and Google's *Alexa* started to appear, being capable of solving simple everyday tasks, such as answering questions using the internet by voice, asking people to write and send emails, or turning the lights on and off at home. Now, through the emergence of the Large Language Models (LLMs), like the one used in ChatGPT, we are witnessing a trend towards integrating these assistants into most applications, like Bing Copilot [13], which can generate, understand, translate, and create code and text content.

With the LLMs, new opportunities appeared that significantly increased the level of productivity of existing software products. By integrating this technology into today's applications, it is possible that when sending data from the application to the language model, it can contextualize it and provide code or answers to questions about that data. Where once it was necessary to carry out a task manually, it is now possible to automate and simplify the processes, with commands given in our natural language, saving time for those who utilize the applications and, in many cases, also saving financial resources. However, natural language is not very rigorous and can be difficult to capture the user intent. If an AI assistant needs frequent fixing, there is a risk of becoming a source of frustration, and instead of the product improving, it may start to receive negative feedback. For example, Office Assistant was eventually discontinued due to its intrusive and annoying behavior, getting negative reviews.[41] These are only one of the few things that we need to consider when developing an AI assistant.

1.1 Context

In applications, we have noticed most of the time, some processes can be automated or basic tasks that are easy to perform and require wasting precious time on the user end. With that in mind, in software applications, we will always need to be concerned with providing the best for the user. So, let's think about this challenge applied for Skills Workflow.

Introducing the Skills Workflow platform, an application with the purpose of helping agencies and studios streamline their workflow. This version is implemented with a cloud-based microservices architecture. It already adopts an attractive *No-code* philosophy, which facilitates the creation of micro-applications without requiring users to have programming experience. The platform also has an extensive marketplace comprising an abundance of applications, ranging from time recording to vacation management, budgets, proposals, and project management, among many others.

An AI assistant could be a valuable addition to Skills Workflow, as it could assist users with various tasks, such as creating and managing projects, tasks, and budgets, or even accessing and analyzing data from the platform. Helping users automate repetitive or tedious tasks, such as filling out forms, creating components such as adding a new task into a project, or getting faster answers, like searching for the project that has the most numbers of employees involved. Skills Workflow imagine their product evolving into software capable of being used by anyone with the fastest possible workflow. It already has a large customer base, each with different requirements. To fulfill those requirements, Skills already offers an attractive way to let clients develop their applications in a versatile manner, using *No-code* tools and disposing of their huge marketplace from where the clients can choose the applications that best fit their necessities. So, we can see an assistant would meet the company's values, bringing advantages such as automation of processes; user accessibility with an even better user-friendly interface; saving in operating costs (may need to involve fewer people in the process); faster answers to our client's clients; and help users who cannot program or do not know all the platform possibilities. However, we can predict some limitations that exist, such as the potential for human error in the user instructions; in the beginning, the users may face a learning curve before getting a huge boost in performance; it is needed to ensure the cost efficiency of the solution using the LLMs; and it will probably be necessary to take into account continuous updates, given the growth of LLMs.

We believe that software products, which do not start thinking about a solution to improve user interaction with their application by automating processes will become the latest Legacy Systems because even though they will probably remain valuable to their users for a long time, as this new technology becomes common, users' expectations and needs begin to change. If an application cannot keep up with the user's needs, it will become less useful and eventually obsolete.

1.2 Objective

The main objective of this dissertation is to automate processes in the Skills Workflow platform by leveraging the use of the LLMs, building what we will call an assistant that works alongside the user. This assistant will help the user in tasks where the user has more difficulty or where he wastes precious time performing basic and monotonous tasks. In these cases, the assistant will facilitate the process and, more importantly, in a way so intuitive that the user does not need to learn how to use it.

Exploring the user tasks and difficulties, we aim to create an assistant in Skills Workflow platform that is connected to the application data and capable of automating processes in the platform, namely manual user tasks. We intend that by demonstrating how to improve applications workflow by automating processes in Skills Workflow, a kind of recipe to reasoning how to build assistants with LLM be expressed.

To achieve an assistant, we will harness the capabilities of powerful LLMs, which appear as a solution for this problem, for having an incredible capability of understanding and communication. The user will communicate with the assistant using an input or voice to transmit what they want to do to the system status. It is important to note that the LLM is not the primary focus of this work, but rather the assistant itself and its capability of serving the user in several tasks. The LLM is an innovative and efficient approach, which can overcome some of the limitations of traditional methods for natural language processing and generation. Following this line of reasoning, we are promoting to enhance the user experience by adding workflows that achieve tasks after interpreting the user prompt, significantly increasing user efficiency, productivity, and accessibility.

The assistant will be able to have characteristics such as generating alerts, building dashboards, searching system data, or even introducing new content explaining what is intended or referencing existing content. It should be able to understand the user intention and lead the system to perform some action in its data. In the state-of-the-art approach, first, we will select the best LLM to integrate and manipulate, and then find a way to integrate an LLM in the assistant and identify features that could be useful to users of the platform. We will think about the principal user needs in our platform, comprehending what can be automated and whether they are completely new use cases or improvements to current ones. Once we have identified the features we will also explore the best conceivable pipeline, highlighting the advantages. In a later phase, we can develop some of these features and create an automated assistant to Skills Workflow. There are challenges in this study, such as discovering if it is conceivable to create an assistant that is not intrusive or frustrating to perform tasks, which are not the user's intentions. Researching the most viable solution to build an assistant using a LLM, in addition to discovering the most advantageous set of features and how to achieve them accurately and persistently.

1.3 Contributions

In this dissertation, we are exploring a way to enhance the user experience of Skills Workflow platform by developing an assistant that is not intrusive and should help the user to automate some of their tasks. The deliverables may comprise:

- Create a well-functioning architecture to implement an assistant using the help of a LLM.
- Develop and implement concrete features in the Skills Workflow assistant, which automates processes for the user.
- For some of the features in the assistant, we may have to generate a fine-tuned model with a more specific purpose.
- Some feedback from the clients related to the feature's utility.
- Discoveries and methodologies can also be used as a foundation for future work in this field from others interested in the same subject. By exploring our proposed features to automate processes using a LLM.

1.4 Document structure

This document is organized into three chapters, each of which explores a different aspect of how to develop an assistant for Skills Workflow using Large Language Models (LLMs). The first chapter introduces the context, objective, contributions, and structure of the document. The second chapter reviews the related work and background information on assistants, LLMs, Skills Workflow features, and their application areas. The second chapter also compares and evaluates different LLMs based on performance, quality, and pricing, presenting some experiments we execute to test their capabilities. The third chapter is divided into previous work, with a few possible characteristics of the assistant for the proposed solution and then the execution plan for implementing and deploying an assistant for Skills Workflow using the best LLM selected from the previous chapter. But also describes previous work done to discover a few possible characteristics of the assistant for the proposed solution and ends with a roadmap to achieve the objective.

2 Related Work Vs. Background

In this section, besides presenting the basic concepts and theories to support our work to understand how to solve our problem, we also present a literature review exploring the uses of LLMs to develop automated assistants on software platforms, focusing on how they could be relevant to our work.

2.1 What is an Assistant?

The definition of assistant in the dictionary refers to someone who offers assistance, help, or a service. If the assistant is integrated into the software, it will offer intelligence capable of providing support and guidance in user tasks. In software, assistants can be used for different purposes, such as answering questions, performing actions, and automating processes, but focus on the objective of enhancing user experience and capable of communicating with the user clearly and naturally. Assistants can interact with users or other programs through voice, text, or a user interface.

An assistant can have similar names like bots, chatbots, virtual assistants, and assistants, but these terms are not always interchangeable and may have different meanings or functions depending on the context. For example, a bot can update application data, index digital information, or perform customer service. Chatbots are more focused on maintaining conversational interactions. Virtual assistants are more used to routine support, being more comprehensive about actions. For us, an assistant should be more focused on acting on behalf of a user, being comprehensive, and combining all of the mentioned before. We identify three categories of assistants based on their functions: assistants who communicate with the user, assistants who collaborate with the user, and assistants who can respond to system events. So, our assistant should be a hybrid version of these three categories, where it will be able to listen and respond to the user based on the platform's data, make suggestions, help the user complete certain tasks faster, and respond to some system events without user involvement.[16]

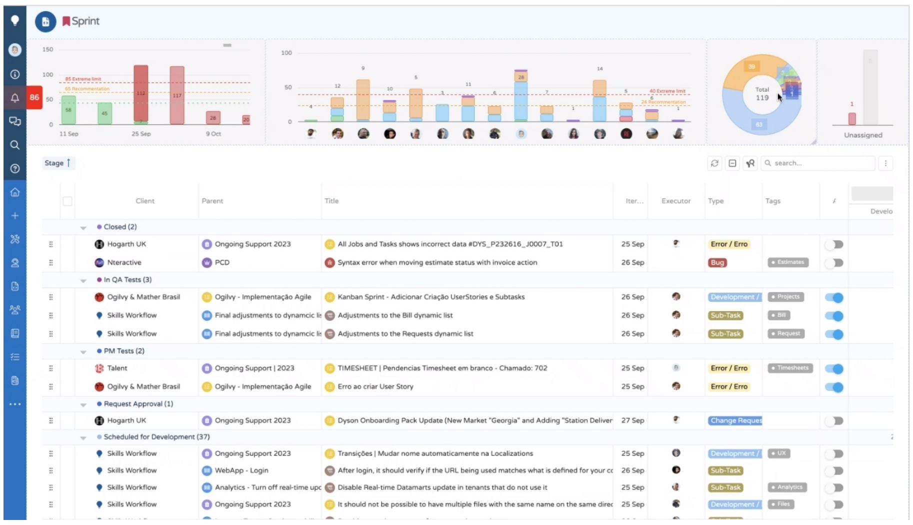
2.1.1 Assistant developed with Large Language Models

There are other types of assistants, such as Siri from Apple, or Google Assistant, which are assistants that use predefined commands and rules to perform some tasks but have a limited range of responses and actions. Sometimes, they fail when dealing with complex user commands. They are more commonly to be seen with IoT applications.[21][3]

On the other hand, intelligent assistants built with the new LLM, can understand and generate texts, analyze data, or even generate code. In addition to their vast knowledge through information displayed on the internet, they can also be trained for specific cases and generate even better quality responses.

2.2 Skills Workflow

Skills Workflow [42] is a company that with its platform helps other companies manage their projects, resources, invoicing, employees, leaves, etc.



Client	Parent	Title	Iter...	Executor	Type	Tags
Closed (2)						
Hogarth UK	Ongoing Support 2023	All Jobs and Tasks shows incorrect data #DYS_P232616_J0007_T01	25 Sep		Error / Erro	
Nteractive	PCD	Syntax error when moving estimate status with invoice action	26 Sep		Bug	Estimates
In QA Tests (3)						
Ogilvy & Mather Brasil	Ogilvy - Implementação Agile	Kanban Sprint - Adicionar Criação UserStories e Subtasks	26 Sep		Development /	Projects
Skills Workflow	Final adjustments to dynamcic li	Adjustments to the Bill dynamic list	26 Sep		Sub-Task	Bill
Skills Workflow	Final adjustments to dynamcic li	Adjustments to the Requests dynamic list	26 Sep		Sub-Task	Request
PM Tests (2)						
Talent	Ongoing Support | 2023	TIMESHEET | Pendencias Timesheet em branco - Chamado: 702	25 Sep		Error / Erro	Timesheets
Ogilvy & Mather Brasil	Ogilvy - Implementação Agile	Erro ao criar User Story	25 Sep		Error / Erro	
Request Approval (1)						
Hogarth UK	Ongoing Support 2023	Dyson Onboarding Pack Update (New Market "Georgia" and Adding "Station Deliver	27 Sep		Change Reques	
Scheduled for Development (37)						
Skills Workflow	Ongoing Support 2023	Transições | Mudar nome automaticamente na Localizations	25 Sep		Development /	UX
Skills Workflow	WebApp - Login	After login, it should verify if the URL being used matches what is defined for your cc	26 Sep		Sub-Task	
Skills Workflow	Analytics - Turn off real-time upc	Disable Real-time Datamarts update in tenants that do not use it	25 Sep		Sub-Task	Analytics
Skills Workflow	Ongoing Support 2023	It should not be possible to have multiple files with the same name on the same direc	25 Sep		Development /	Files

Figure 2.1: Skills Workflow example.

Through the example in Figure 2.1, we can understand the functioning of the platform by looking at a dashboard that is managing a company sprint. It shows the current tasks done and to do in each category, the person a task is attributed to, or even the status of the number of tasks for each employee. To contextualize, each dashboard corresponds to a micro-application capable of a more specific objective, like this example.

Skills Workflow allows users to fully customize their dashboards/workspaces with all the different features, adding the best graphic or changing it to a table. The dashboard in the platform is the most important component describing all the smaller components that appear on the screen in an JSON object (e.g., calendar, table, chart, etc), which can also have some functions to manipulate the dashboard data (e.g., to filter something). With the existing functionalities, the client can create a dashboard with a lot of different components suited to their needs. However, he has to do it manually, dragging widgets and making other customizations, instead of simply being able to say, "I want a workspace that shows all employees and the respective holidays that have already been enjoyed or have yet to be enjoyed by everyone.", and the assistant automatically creates the workspace. Similarly, the user may want to know which employees integrated into project "x" are going on vacation in April, having to go look for them instead of the assistant responding to him almost instantly with the names. Or even if the client wants to know if someone on a project has more than seventy hours of work per sprint iteration and has to check it constantly instead of asking the assistant to check this twice per sprint with a notification. Here are some examples of how the assistant can help the user, giving him more time to focus on important tasks.

2.3 Large Language Models (LLMs)

A language model is an algorithm that attempts to generate plausible language by predicting the next word to be written and learning the structure and meaning of the language from textual data. We can see it is a machine that repeats words it has heard with some understanding of their context. At a low level, a model has a well-defined and limited objective, while at a higher level, a model can learn and adapt to interact flexibly.[17][40] The difference from the traditional language models is that they are based on rules and linguistic characteristics developed by humans, having a limited understanding with smaller context windows to predict words. They are more used to specific tasks, being less versatile in different contexts.[32]

There are different models for text, video, and image, but they can rely on LLMs to process and generate text. LLMs are trained on a large amount of unstructured data from the Internet, where this training is a process where the parameters of a model are adjusted to minimize the error between predictions and actual data, determining how a neural network processes the input and produces the output data. [20] To answer, it first needs to transform the prompt given by the user into tokens that it can understand. Tokens are sets of characters with some meaning, but for better understanding, we can think of them as words. LLMs then observes each token in context with the other tokens in the input sequence and takes note of the nearby tokens, leaving us with a set of words mapped to word embedding vectors (numerical representation of a token, capturing its meaning, semantics, and relationship with other tokens). Two similar embedding vectors mean two tokens with similar meanings. To make the LLMs "smart" it used the Transformers architecture with a deep learning technique called self-attention. The self-attention mechanism computes the similarity between each pair of tokens in the input sequence, giving it scores for token pairs and creating a weighted representation of each token based on the similarity scores. Self-attention will allow the LLMs to take context from sentences and generate an answer predicting word after word.[37][19]

To use the LLM in applications, some companies expose an API that enables developers to interact with the LLMs and their features. For instance, we can access endpoints to turn audio into text or text into audio, to give chat completions, or even give an input image. All these endpoints will influence the possible functionalities our assistant can have.

2.3.1 Open-source vs proprietary models

Open-source LLMs allow anyone to access the model, modify, and distribute the source code and architecture. These language models are free, promoting collaboration and innovation among users, allowing them to have more control over the model. The downside is related to the fact they are vulnerable to community exploitation and depend on it for support, updates, and maintenance. On the other hand, proprietary models are owned by a company and protected by copyright and patents. They are paid and have integration and customization options limited by the company. However, performance and quality are guaranteed by the company, which may provide technical support if necessary. Therefore, we are more focused on the models that offer constant updates, support, and maintenance.[35][15]

2.3.2 Vulnerabilities of LLMs

LLMs are not comparable to search engines because they predict a sample from a probability distribution over possible words. Predicting words leads to models facing challenges like rare unseen words, overfitting to specific domains, or capturing complex linguistic phenomena. One of their biggest problems is a process called "hallucination" that results in incoherent or meaningless content. It can happen because the training data is insufficient or noisy or because a prompt is poorly formulated, ambiguous, or introduces a complex linguistic phenomenon. To avoid "hallucinations" the models must have verification and validation mechanisms, so inconsistencies are detected in the generated text.[30]

Other vulnerabilities related to the use of LLMs are security and privacy issues because we send our code and data to external services. To add to the list of vulnerabilities, we can have bias problems as models are trained over internet data. To solve the problem, LLM should have content moderation policies and something similar in the input.

2.3.3 Fine-tuning a LLM

Tuning a model is the process of tweaking an LLM to improve its performance on a specific task or domain. Pre-trained models are already powerful but may not perform optimally for some specific use cases that require a particular answer. With additional training, we can enhance the normal performance and accuracy of the LLM for our desired application. Therefore, fine-tuning a model will provide higher-quality results for the purpose it has trained and probably save tokens due to requiring shorter prompts.[24] Some common use cases where fine-tuning can improve results are [22]:

- Setting the style, tone, format, or other qualitative aspects.

- Improving reliability at producing a desired output.
- Correcting failures to follow complex prompts.
- Handling many edge cases in specific ways.
- Performing a new skill or task that's hard to articulate in a prompt.

> **Note:** Hugging Face is a platform that has some pre-done models to train the LLMs. For example, a dataset of TypeScript snippets [39], to train the model to generate code in TypeScript.

2.3.4 Prompt engineering with LLMs

Prompt engineering is a very important step to get better outputs from the LLMs. We have tried and searched for some good ways of using prompt engineering to get better results. This may vary from model to model but there are some tactics like the ones given by OpenAI [26]:

- Use delimiters to indicate distinct parts of the input.
- Specify the steps required to complete a task.
- Specify if we want a delimited output giving it the maximum length (e.g., 100 words).
- Use just provided articles delimited by triple quotes to answer questions. If the answer cannot be found in the articles, write "I could not find an answer.".
- Use code execution to perform more accurate calculations or call external APIs (e.g., write and execute Python code by enclosing it in triple backticks, e.g. """code goes here""". Use this to perform calculations.).

This is basically what the user needs to do to communicate with the Large Language Model in the best way possible.

2.3.5 OpenAI models

OpenAI is known for having the "better" language model in the market, the GPT-4, a transformer-based model estimated to have, without confirmation, around 1.7 trillion parameters. GPT-4 is more complex, expressive, and can handle more data, having a very good performance. Capable of doing complex tasks, generating coherent texts, and giving answers with human-like insight and understanding.[31] Also, GPT disposes of speech-to-text (Whisper model) and image (DALL-E) models for our use.

GPT 3.5 Turbo is smaller than the 4 version and has 20 billion parameters. The version is currently available on ChatGPT and can be used for free. Both GPT 3.5 Turbo and GPT-4 models are not open source, and their code is not available to the public.

OpenAI models monitor inputs and have some moderation policies categorized in hate, harassment, threatening self-harm (intended and instructions), sexual, and violence. It is an important point as it provides us with security when using the model to prevent misuse of the technology.[25]

2.3.5.1 Azure OpenAI

One of the reasons this can be a pleasing option for us is because the Skills Workflow platform is deployed using Azure services and OpenAI is a service in the Azure cloud platform, which is a significant advantage because it is designed thinking of enterprise customers and their requirements of production, such as availability, security, or privacy. In addition to that, Azure provides options like chat responses about business data through the Azure Cognitive Search index.

We can access Azure OpenAI services in several ways, from REST APIs, and web-based interfaces, to Python SDK in the Azure OpenAI Studio. The Azure offers a positive solution in our understanding, because it has private networking and regional availability, making it a very reliable solution. However, for now, they have some boundaries in some of the services provided (Whisper and DALL-E) that are still in preview.[18] In terms of Data and security, Azure ensures that no one but the customer has access to customer data, not being shared with third parties. Regarding content filtering, Microsoft evaluates prompts and completions with high-severity content because being the models trained from internet content, social bias is always present, and Microsoft tries to promote a responsible use of AI.

2.3.6 Google models

Bard is running under the Google AI Gemini language model. This is the most recent model from the ones presented here. For that reason, it was difficult to find much information about it. Bard chat is running the Pro version, which is currently with an estimated 130-250 billion parameters. The Ultra version is the most powerful, with an estimated 1.56 trillion parameters, and even though it may be a little smaller than GPT-4, it does not mean it is the worst, but sometimes rather beneficial, reducing training costs, improving efficiency, and reducing latency. Gemini is a proprietary model that is only accessible through the Gemini API, which developers can use to create applications leveraging Gemini's capabilities, but they cannot access or modify the underlying code of the model, like an open-source one. Gemini models have fine-tuning support but are limited to certain data types and unavailable for audio or video data. [34]

After searching about the Google Bard model, we found that with the PaLM 2 (previous version of Bard) Google has spent some money to generate multiple response drafts. But they are doing this with a purpose, to collect data for fine-tuning Gemini, which may be a good idea to improve their model.[33]

Bard AI, with the Gemini model, also has (besides speech-to-text and image models) an incredible feature still missing in OpenAI, which is video understanding. This feature can open up another range of possible use cases we can implement on our platform.

Concerning the collection of information Google gathers through Bard conversations, information about the use of related products, our location, and feedback.[14] The collection of conversations and information about related products we don't see as something catastrophic, understanding that it helps to improve the model. However, regarding location collection, we've always been a little confused, because even though it improves the user experience, we don't think it should be mandatory for using Bard. After all, it always ends up violating our privacy directly by exposing our location.

2.3.7 META models

LLaMA is a language model based on the transformer architecture and operates using auto-regressive techniques. The latest version of this model, LLaMA 2, has three available parameter sizes. Its available models have 7 billion, 13 billion, and 70 billion parameters. LLaMA 2 generates output with a system called reward modeling. It has two different reward models, the first for helpfulness and the second for safety. We assume this means that any tool using the LLaMA 2 model can produce both safe and useful output.[38]

According to the paper written by META [38], their model is an open-source model optimized for dialogue use cases. Which can be good for encouraging community development, innovation, and educational opportunities. However, being a model for dialogue, it was not developed to offer better performance in the fields we are looking for, related to code generation and/or code interpretation, for example.

2.3.8 Models Comparation

We look for something profitable and capable of carrying out the tasks we want with the best possible quality. A model that does not suffer from hallucinations, is accurate, and with a reasonable relation cost/price.

We believe the two best competitors are OpenAI and Google models, respectively, the between OpenAI GPT-3.5 and GPT-4, and Google Gemini Pro. Besides the fact each one is trained differently, we understand that GPT models provide detailed responses and work well with Azure services. In the case of GPT-4, it still doesn't have available fine-tuning, but in this case, we can make use of GPT-3.5. Google, on the other side, in addition to having some efficient models, also owns a remarkable one capable of interpreting video, unavailable in OpenAI.

Next, we created a smaller version of the tables developed in a report by the Google Gemini team [36] with each model performance, but only with the information relevant to us so we can draw some conclusions.

> **Note:** X-shots mean the model is given X examples to learn from; cannot test and use Gemini Ultra from Google because it is still unavailable in Europe.

Looking at the values in Table 2.1, we can extrapolate that Gemini Ultra is the best-performing LLM on all benchmarks. However, we can not test it ourselves for not being available in the market. From the rest of the table, we are particularly interested in the math and coding tests, where we can verify that after Gemini Ultra is GPT-4. Right behind

	Gemini Ultra	Gemini Pro	GPT-4	GPT-3.5	LLAMA-2
MMLU Multiple-choice questions in 57 subjects	90.04% CoT@8	79.13% CoT@8	87.29% CoT@32 via API	70% 5-shot	68.0% –
GSM8K Grade-school math	83.7% Maj1@32t	86.5% Maj1@32t	92.0% SFT & 5-shot CoT	57.1% 5-shot	56.8% 5-shot
MATH Math problems across 5 difficulty levels & 7 subdisciplines	94.4% 4-shot	32.6% 4-shot	52.9% 4-shot via API	34.1% 4-shot via API	13.5% 4-shot
BIG-Bench-Hard Subset of hard BIG-bench tasks written as CoT problems	83.6% 3-shot	75.0% 3-shot	83.1% 3-shot via API	66.6% 3-shot via API	51.2% 3-shot
HumanEval Python coding tasks	74.4% 0-shot	67.7% 0-shot	67.0% 0-shot	48.1% 0-shot	29.9% -shot
DROP Reading comprehension arithmetic	82.4 Variable shots	74.1 Variable shots	80.9 3-shot	64.1 3-shot	– –
HellaSwag Common-sense multiple choice questions	87.8% 10-shot	84.7% 10-shot	95.3% 10-shot	85.5% 10-shot	80.0% –

Table 2.1: LLM performance on text benchmarks.

comes the Gemini Pro, which is above GPT-3.5 and a little below GPT-4. Nevertheless, it is important to note that the benchmarks are not perfect, and the LLMs may have different strengths and weaknesses in different tasks.

Relatively to the image analysis capabilities in Table 2.2, we understand that Gemini Ultra is the overall best, with high accuracy on most tasks. GPT-4V comes right after, and Gemini Pro is not far behind. The benchmarks provide valuable insights, but like in the text benchmarks, we need to take into account that a LLM performing very well in most cases can fail to produce a good result in a few specific cases.

After the analysis of results taken from an article written by Google, to carry out some suspicions of influencing the outcomes somehow, we encountered a recent study by a third party that made a comparison between Google's Gemini Pro model and OpenAI's GPT, diverging in the fact that GPT 3.5 has slightly better accuracy than Gemini Pro, which is not what we saw in Table 2.1 comparing the text benchmarks. The article [2] is of very valuable interest because after all the analysis they conclude that:

- Gemini Pro achieves, on average, accuracy and performance somewhat inferior to GPT-3.5 Turbo and notably worse than GPT-4;
- Relatively to code generation, Gemini Pro had a higher proportion of mistakes than the other two;

	Gemini Ultra	Gemini Pro	GPT-4V
MMMU Multi-discipline college-level problems	59.4%	47.9%	56.8%
TextVQA Text reading on natural images	82.3%	74.6%	78.0%
DocVQA Document understanding	90.9%	88.1%	88.4%
ChartQA Chart understanding	80.8%	74.1%	78.5%
InfographicVQA Infographic understanding	80.3%	75.2%	75.1%
MathVista Mathematical reasoning	53.0%	45.2%	49.9%
AI2D Science diagrams	79.5%	73.9%	80.9%
VQAv2 Natural image understanding	77.8%	71.2%	77.2%

Table 2.2: LLM performance on image benchmarks.

- In mathematics, Gemini Pro has a superior performance to GPT-3.5 Turbo in the most complex examples requiring longer chains of thought but underperforms in shorter examples, indicating sensitivity to the length of reasoning chains and large digits;
- GPT-4 exhibits robustness in handling long reasoning chains in mathematics;
- Comparating general-purpose reasoning, Gemini Pro achieves slightly lower accuracy than GPT-3.5 Turbo and much lower accuracy than GPT-4 Turbo;
- Gemini Pro, in question-answering tasks may have some bias in answer ordering (favoring the final choice "D" in multiple choice) and underperform in most tasks with chain-of-thought prompting. Achieving lower accuracy than GPT-3.5 Turbo and significantly lower than GPT-4 Turbo.

So, watching this comparison and the one done by the Gemini team, we comprehend that Gemini may be using the best methods to get the best performance in the benchmarks. The conclusions we can take from merging the results of both studies are that Gemini Pro has a rather worse performance than GPT-4, and may be at the level of GPT-3.5 Turbo. However, it is difficult to make a comparison with results that perfectly indicate the best model, because the models are trained on different algorithms, and depending on the cases we can observe random unexpected results for any model.

2.3.8.1 Pricing

Relatively to prices if we analyze in English, and assume that one token is on average, equal to four chars. Google charges characters and OpenAI in tokens, so if we use these numbers we will see prices like:

	Input	Output
GPT-3.5 Turbo Model: gpt-3.5-turbo-1106	$0.0010/1K tokens	$0.0020/1K tokens
GPT-4 Model: gpt-4-0613	$0.03/1K tokens	$0.06/1K tokens
Gemini Pro Text Model: Gemini Pro	$0.001/1K tokens	$0.002/1K tokens
GPT Images Model: DALL-E 3 (Standard, 1024x1024 resolution)	$0.040/image	–
Gemini Pro Images Model: Gemini Pro (1024x1024 resolution)	$0.0025/image	$0.020/image
Gemini Pro Video Model:	$0.002/second	–
GPT Speech-To-Text Model: Whisper	$0.006/minute	–
GPT Text-To-Speech Model: TTS	–	$0.015/1K characters
GPT Fine Tuning Model: Babbage ($0.1600/100000 tokens Training)	$0.0006/1K tokens	$0.0024/1K tokens
Gemini Pro Fine Tuning Model: n1-standard-4 (europe-west2)	$0.281520/hour	–

Table 2.3: Pricing comparison of LLMs.

Note: This is an analysis where 1 token = 4 chars in English.[29]

Looking at Table 2.3 we comprehend that the GPT-3.5 Turbo is the cheapest option, but we saw in Section 2.3.8 that it normally presents the worst results. Between GPT-4 and Gemini, the prices are not very different, at least applying the conversion we used. So, it is probably more relevant to understand the quality of the individual models. It also caught our attention that OpenAI as an API costs usage of $0.03/session in the code interpreter and $0.20/GB/assistant/day in the retrieval.

2.3.9 Experiments with LLM

To examine the differences between the language models, we made some experiments to compare their responses. The analysis we did, tested some of the steps that possible features may need to be performed and see if they are possible, and how trustworthy they can be, using GPT-4, Gemini Pro, and GPT-3.5 Turbo models. For the GPT-4, we have to utilize a free option through the Bing chat. However, the input is just 4000 characters.

Then, we tested the Gemini Pro through Bard chat, and the same for GPT-3.5 using the ChatGPT.

In the first test, we gave them a JSON structure and asked them to interpret the fields that were given. With this, we were able to analyze the model's capability of comprehending content. Overall, every model doesn't provide a wrong interpretation, but sometimes they don't detail as much. GPT-4 was the only one that seemed to articulate and best understand the JSON structure and the meaning of the fields.

Then we test data analysis with the models, giving them data in the format of a JSON file with information on employees' vacations, and asking them questions about that. The responses were prevailing wrong, only banal questions got correct answers. When the models were confronted to calculate who had gone on vacation in April this year, all the models didn't answer the question correctly. Not even giving them context about the fields, the answers are not coming out correctly. It seems to us that the models were confusing the needed fields to compute the answer, seemingly using the first field they encountered in the JSON to respond. After this, we try to change the format of the JSON file to something more nested. The results were partially the same. So, finally, we try a simple approach, giving them fields of a "View" (DB projection with less information, only with the most relevant fields and a small description of each). Then we asked to generate JavaScript code to use in the JSON file that returned an array of objects of the employees who went on vacation this year in April. Now, the models produce functional responses. Only GPT-3.5 was having difficulty, not filtering the year sometimes. Conclude that it is the best approach as well to get data.

After testing the analysis of data, we try to fine-tune the GPT-3.5 model for workspace creation. Workspaces/dashboards have a predefined structure, so we first try to ask to build one using only prompt engineering and then fine-tuning to understand the difference. The difference was a little clear because fine-tuning the model to generate the dashboards will give us something that could be partially integrated into the platform, and the only thing that needs to be changed is the components plus their structuring. Probably, for the resolution of this problem, we will also have to train the model in all the existing components to build the workspaces (e.g., forms, calendars, buttons, etc) so it knows what can be used to generate the workspace.

Finally, we test the speech-to-text feature. We give the models the same inputs, and either model responds without problems. The audio was clean and could be interpreted well. Otherwise, the result may vary.

Note: We can only train the GPT-3.5 because we get 200$ free from Azure services.

2.4 Application areas of the assistants

When looking for related literature, we found some studies that integrated LLMs in the context of the company's application. OpenAI has a webpage featuring several stories from customers who use GPT models to advance their goals.[23] There were a few that caught our attention. *Stripe* [27] was one of them. They took advantage of the services offered by GPT and improved their product's resources and workflows, just like we intend to do. They led a team of engineers to explore the capabilities of GPT-4 and concluded they could use the model to scan, summarize, and analyze large-scale data very quickly, outperforming any human. Looking at this, we understand that we can make use of Large Language Models the same way, using them to analyze the huge amount of data from our platform, and more specifically from each dashboard, and respond to some user questions about it.

Another of the stories that caught our attention was *Be My Eyes*'s [28], as they used GPT to empower the blind and low-vision community leading a revolution in visual interpretation. Our first thoughts were how we do something similar, blending the real world environment with the Skills Workflow platform, capturing the real and transforming it into something in the digital one. This way, we can create features that enable, using voice commands, effortless communications but also help blind people interact with the application if necessary. Similarly, by utilizing images, we can convert real-world information to our application, like printed documents, transforming them into digital files.

From almost every example, the GPT needs to generate data, and it is what we want to do create new information and code to do something in our platform. The examples provided by OpenAI, many of them make use of the model the same way, for processing large amounts of data or creating new data through user communication.

We also discovered that *Caixa Geral de Depósitos* (Caixa Geral de Depósitos sort) has a virtual assistant [5] that can interpret natural language through voice or text and do things like make bank transfers, check balances, and account movements. They managed to create an assistant that can perform tasks based on an interpretation of human language, where the user only has to express their needs. This improved the convenience and efficiency of their customers' banking experiences, which meets what we want to achieve as well, automating processes and enabling the user to communicate with the application intuitively, to carry out tasks more quickly and efficiently.

We explored *Copilot* as well, Microsoft built an assistant that can be integrated into Visual Studio Code and is capable of generating code through natural language. It can even follow the user's progress and analyze the open document to understand how the code is being created and generate new.[9] This is a very interesting feature because the LLM may need to first analyze the code to understand how something is structured in the application and then add new code to it. Helping programmers not just to write faster, better, and more reliable code, but also providing similar answers to what they already have.

Lastly, we found an incredible application *AgentHub* [1] interested in building automation within a single line of code, which fits perfectly into what we want to develop. To build automations they use drag, drop, and linking of nodes onto a canvas, building the workflow. This way they reduce the chance of having a global assistant to do that same task and fail. They have some templates of workflows to produce things like a LinkedIn profiles processor, a tweets generation Bot, a code problem solver, an email inbox summarizer, etc. Many of these are possible because they have integration to some services like Twitter, GitHub, or Google.

There are many other possibilities that we are thinking of exploring, here we merely understand the general uses that can be done. Summarizing, we can create a table of general characteristics that were found in the examples:

	Defenition
Data analysis	As we can see in Stripe, they use GPT to analyze and summarize huge amounts of data. So, we realized that we could search and analyze data from the platform.
Code Generation	From Copilot, we understand that we can generate code for anything, and execute it most of the time correctly.
Real world to digital word	From Be My Eyes, we understand that we can communicate to the application in more than one way besides text.
Performing actions	With Caixa Geral de Depósitos sort we perceive that we can use the LLM to help communicate with the user and to orchestrate some existing actions in the application.

Table 2.4: General characteristics of LLM assistants.

It is common for platforms with AI-powered assistants to face limitations in carrying out certain user requests. It can occur due to the assistant's incapacity to produce the correct result by following an incorrect workflow, or by the LLM miss compreend what the user wants. Relatively to this last, it is difficult to attack the problem, seen to be related to the user input. However, related to workflow production, we may adopt a strategy to reduce the probability of failure, like *AgentHub* did.

2.5 Solution to build an assistant for Skills Workflow

Following the process of developing an assistant, after looking at the performance of the LLM and analyzing some of their advantages and disadvantages, when looking at the characteristics of the assistants mentioned in Table 2.4 we have to think about an architecture capable of supporting refined features in that scope.

2.5.1 Overview

When thinking about a solution to build this assistant, we think about what already exists in the application and what the assistant should do. The application offers several endpoints that allow all operations available to the user to be done. We intend to use these endpoints, but also create new ones if necessary, allowing all the necessary actions to be carried out for the assistant to function in terms of the automation that is intended to be carried out. We also want our assistant to be incremental and to be able to easily adapt to new features over time as the platform and user needs evolve. With this idea in mind, we created the following view:

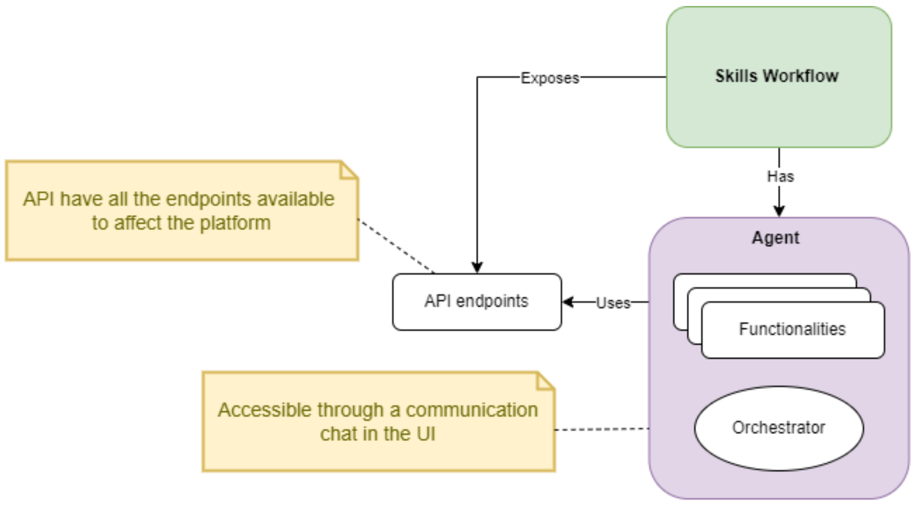

Skills Workflow
Exposes
Has
Agent
API have all the endpoints available to affect the platform
API endpoints
Uses
Functionalities
Accessible through a communication chat in the UI
Orchestrator

Figure 2.2: How an assistant can be integrated into the platform.

It is crucial to understand the steps that the orchestrator will choose are going to depend on the user input, so we need to develop carefully the component that manipulates the way the task is performed, so it correctly accomplishes the user request.

2.5.2 Building an assistant using Azure Semantic Kernel

When searching for concrete solutions to our problem, we found that Azure has a good solution that can be used to build an assistant. It is called Semantic Kernel, which is a new approach that allows us to integrate LLM into our platform. It is the same technology used by Microsoft to build Copilot, so it can power our platform as well.

An assistant should be capable of retrieving information from both the user and system, conceiving how to use that information and how to respond to the user or perform an action in the system. The Semantic Kernel, similar to operating systems, will manage resources like assistant concrete functionalities, the memory of the assistant, and the connectors (modules that enable external data access, such as Blob Storage and Cosmos DB, which is connected through Cognitive Search). It is an open-source SDK that allows us to manage our existing code with AI. This means that we will be able to give our assistant the ability to interact with the user by calling our existing services and creating new ones. [6][7]

2.5.2.1 Advantages and Limitations

The advantage of this approach is that the Semantic Kernel orchestrates and chooses the right functionalities using a single LLM call, which reduces the complexity and cost of building an AI assistant. However, there are limitations related to the writing of effective prompts, capable of producing the expected results. Although this problem will always be difficult to solve and is something present in things connected to LLMs.

We searched for similar solutions, but we did not find one that joins conventional programming languages with LLM prompts, other platforms mainly provide access to pre-trained or fine-tuned LLMs and do not allow us to write semantic functions or even integrate them with existing code, which will lead us to have to perform several LLM calls to a single user prompt request. They also don't provide a mechanism to orchestrate all the different types of functionalities that we want to implement, which is one of the biggest downsides for the other competitors of Azure.

The disadvantage we see in using the Semantic Kernel is the fact that it is still under development, and may contain some bugs and is not completely stable, in addition to there being few examples to base ourselves on.

2.5.2.2 Architecture using Semantic Kernel

We have idealized an architecture for our assistant using the Semantic Kernel.

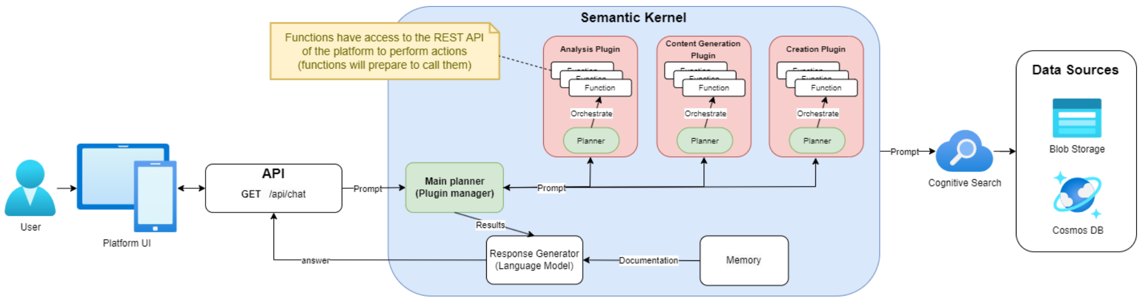


Figure 2.3: Semantic Kernel architecture.

The assistant comprises four essential components: plugins, memory, planners, and its persona.

Plugins - At a high level, a plugin is just a group of functions that can be exposed to AI applications and services. These modules give the assistant skills or characteristics, consisting of both native code and requests to AI services via prompts. For the AI to distinguish the functionalities in the plugin, each one will have a description that explains what it does so it can be chosen accordingly.[10]

Memory - Memory is the component that stores and retrieves contextual information generated from conversations and imported from external sources, such as documents, web pages, databases, etc. All the information can be accessed via semantic functions by the AI assistant and the user. These functions*, are natural language expressions that can be evaluated by the assistant to perform operations on his memory.

Note: * There are two types of functions that we can work with in plugins, semantic functions, and native functions. Semantic functions are called out from AI models, and they access or modify the memory of the assistant (do not require a fixed syntax and can be composed dynamically by the assistant or the user). The native functions, allow us to do other things, like mathematical calculations, because as we saw in Section 2.3.8 related to the performance of the benchmarks, they are not very good with math, doing things more reliably. Semantic functions can help us to perform data analysis tasks, such as data validation, transformation, visualization, and querying.

Planners - Planners are algorithms that use the AI service to decide what to do with the user's question/prompt, planning the steps to achieve the user task. There are two types of planners, Handlebars Planners and Function calling stepwise planner. Handlebars Planner can generate an entire plan with variables, conditionals, loops, and expressions, all with a single call to the AI service. On the other hand, function calling stepwise planner, can orchestrate a plan by calling functions from the AI service one by one, and evaluate the results of each function call so it can decide the next step, using the OpenAI function calling capability.[8][4]

Persona - The persona gives our assistant a personality. So we can make it like a person, more or less sarcastic, formal or friendly.

For our implementation, we can have a plugin for analysis-related use cases. Another plugin for content generation use cases where we generate data based on information from the platform. And a last one for creational use cases, this also generates data but from scratch. An organization accomplished this way would allow plugins to have some common functions where the result could be used differently by use cases. For example, in the analysis plugin, create a function that performs a search on the user request, and then the result will be used differently by the use cases.

For the Planners should exist one main planner for choosing between the plugins, and one for each of the functionalities. Inside each plugin, will be created all the necessary functions to accomplish each use case. Besides the memory needs to be configured to get data from documents and databases, it also needs to save conversation history relevant to the user's task. The response generator is the component that uses an LLM to create the final natural language response for the user.

In Figure 2.3, we can see an important part of the implementation, Azure Cognitive Search. This service enables us to perform searches on our data stored in Cosmos DB or Blob Storage through natural language queries. This is highly pertinent as it will allow us to analyze our data efficiently. To use the service is necessary to index data that we consider relevant for research to Azure Cognitive Search, CosmosDB, and Blob Storage. As we have a lot of data, we are thinking that the best solution will be the use of projections of the databases, only with the important fields and tables. After this, we need to connect to Azure Cognitive Search, Cosmos DB, and Blob Storage.

One of the functions that should be included and utilized by all the requests will notice if the user is interrogating what he is seeing in the dashboard or if the question is not about it. This function should be performed at the beginning of the pipeline and decide the data that will be used to execute the rest of the request.

2.5.3 Azure Bot

Another available option in Azure is the Azure Bot.[11] This SDK is a more robust and integrated tool that allows us to use Azure cognitive services with bots, especially Chatbots mostly focused on system analysis. However, it seemed to us that the Semantic Kernel (explained in the previous Section 2.5.2) is more flexible, allowing us to use any natural language model with our code, without being dependent solely on Azure services.

2.5.4 Google Dialogflow

Google also has its assistant-building tool, the Diagramflow. This tool similar to the Azure bot can search the web for information, create content based on user input specifications, summarize, make comparisons, and answer questions about our application's functionality or performance. Based on [12], Google Dialogflow has better natural language processing and advanced training features than Azure Bot Service. However, Azure Bot Service has more integrations and is more focused on conversational AI. Both of these bots have more restricted features than the Semantic Kernel, which allows us to generate more complex and specific functionalities and workflows.

2.5.5 Distributed Assistant

Another alternative solution would be to separate the assistant into the different functionalities that we want to implement, and instead of the user accessing the assistant all in the same place by talking to a chat, we place each functionality in its proper place, and the user can access them from the more direct way, from the specific micro-application the system has, reducing the chance of the assistant to fail in producing the exact workflow the user wants (an approach like AgentHub did, mentioned in Section 2.4). For example, if the user wants to analyze the system's data, we will use a text input in an accessible place on any page of the platform. If the user wants a summary of a message feed where several users are discussing a topic, we place a button in that feed that generates the summary. If, on the other hand, the user needs to create a systematic check of a parameter in the system, with a trigger per time, or activated by changing the parameter, we place a button on the home page that allows the user to create these situations. And so on. Following this logic, we have the advantage that the system performed what the user wanted in the first place, and the errors can only occur in the processes of the task itself where the LLM is involved.

2.6 Discussion

Taking into account the models that we deprecate, taking into account that the benchmarks are not perfect and LLMs can have different strengths and weaknesses in different tasks. We can conclude that the META language model is not the one that best meets our needs, as it has less performance and is more optimized for dialogue. Between GPT and Gemini, we can see that they are similar in terms of performance or price, so they are both options to consider. However, we are more inclined to the GPT.

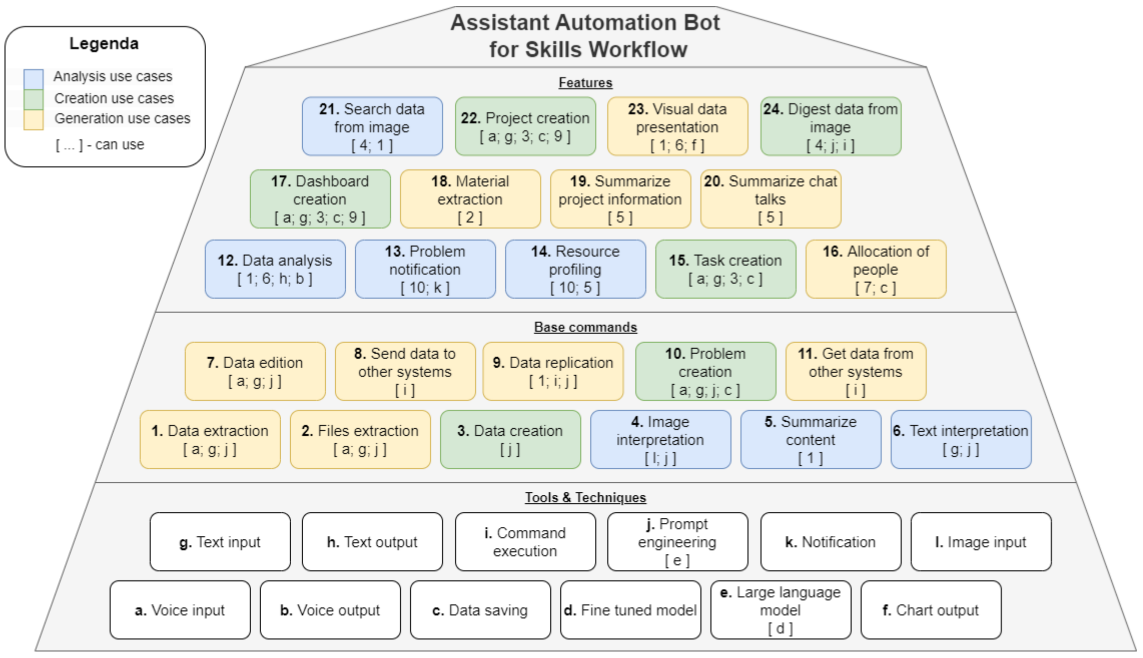
Regarding the solutions seen for the assistant architecture, the ones we consider to be the most viable are the Semantic Kernel or distributing the assistant across the platform. The first solution (2.5.2) is the most complete and allows us to set up an assistant that can be accessed in the same place, which was the solution more explored in-depth because it is a new tool and has little information yet for concrete uses. The second solution (2.5.5) reduces the chance of errors by automatically performing a requested functionality and not doing others that could lead to an undesired outcome.

3 Solution Proposal

In this section, we will address previous work that was carried out. We will reflect on the lessons learned and use that information to design the think about a solution, including features specifically for the assistant. Finally, we will provide a roadmap overview of the work to be done.

3.1 Previous work

In order to develop an assistant, we need to closely examine the problems and processes that users spend a significant amount of time on. We should analyze the functionalities of Skills Workflow (some expressed in Section 2.2) and the general possibilities for uses of the LLM discovered in Table 2.4 from Chapter 2. By doing so, we can identify processes that can be automated to improve productivity and reduce user workload.



Legenda
🟦 Analysis use cases
🟩 Creation use cases
🟨 Generation use cases
[...] - can use
Assistant Automation Bot for Skills Workflow
Features
21. Search data from image [4; 1]
22. Project creation [a; g; 3; c; 9]
23. Visual data presentation [1; 6; f]
24. Digest data from image [4; j; i]
17. Dashboard creation [a; g; 3; c; 9]
18. Material extraction [2]
19. Summarize project information [5]
20. Summarize chat talks [5]
12. Data analysis [1; 6; h; b]
13. Problem notification [10; k]
14. Resource profiling [10; 5]
15. Task creation [a; g; 3; c]
16. Allocation of people [7; c]
Base commands
7. Data edition [a; g; j]
8. Send data to other systems [i]
9. Data replication [1; i; j]
10. Problem creation [a; g; j; c]
11. Get data from other systems [i]
1. Data extraction [a; g; j]
2. Files extraction [a; g; j]
3. Data creation [j]
4. Image interpretation [l; j]
5. Summarize content [1]
6. Text interpretation [g; j]
Tools & Techniques
g. Text input
h. Text output
i. Command execution
j. Prompt engineering [e]
k. Notification
l. Image input
a. Voice input
b. Voice output
c. Data saving
d. Fine tuned model
e. Large language model [d]
f. Chart output

Figure 3.1: Use cases pyramid.

Note: The square brackets in the image represent the use cases each may use.

The pyramid shown in figure 3.1 was created to better visualize all functionalities and brainstorm about all the possible features. We break down smaller use cases (Base Features) and identified what could be used from one use case to another. This will help us to develop concrete use cases by scrutinizing existing features. In the following sections, we will delve deeper and explore some candidate use cases for implementation.

3.1.1 Framing the architecture with use cases

Looking at the model in Figure 2.2 depicts the assistant architecture and the ways of implementing it presented in Chapter 2, we decided that we would divide the features into three categories. The first is the *analysis*, where we will place all the functions that will analyze the application's data. The second is *content generation*, where we will place all the functions that will generate content based on application data. The third is the *creation* category, where we will put all the functions related to content creation based on the user's prompt. With this strategy, we can allow the reuse of some functions, for a certain feature in the same category. The orchestrator mentioned in the architecture should now be able to distinguish between the use cases described in this section and use them to answer or perform some action for the user.

3.1.2 Data Analysis use case

The Data Analysis use case aims to, when the user has all the information exposed in the workspaces and cannot interpret it or simply wants a quick search, he just asks the agent in the system about the data stored in the application's data sources.

This feature requires a consistent solution, one capable of giving correct results. We carried out a few tests in Section 2.3.9, to find the best way of getting a good result. From the tests where we give all the data to the LLM, we realized that models have difficulty interpreting large quantities of data, either in JSON, due to nested structures and because it stops as soon as it finds any pertinent data in the structure or CSV because even though it is separated by commas or semicolons, forcing it to scroll through the entire file, the model can't interpret it correctly. Besides that, this type of implementation will use a lot of tokens and, therefore expensive. The other test is to ask LLM code to apply to those files and get the answer. This last approach is more solid, providing correct results. To further improve the solution's correctness, as the application's database is large, we decided to create database projections (views) carrying only the relevant fields to perform queries without irrelevant parameters that are only there to connect functionalities on the platform, not offering relevant content. For this, we will use Azure analytics, so the projections are updated automatically as data is added to the database and the answers are up to date.

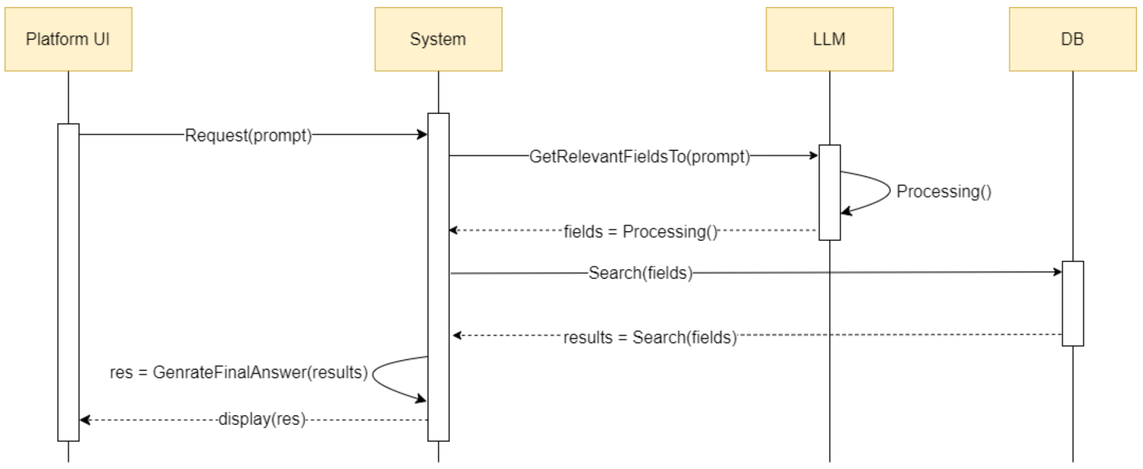


Figure 3.2: Sequence diagram for data analysis use case.

The use case should be inserted into the analysis category. It will retrieve information from our platform data, and the LLM will provide answers in natural language to the user. For this, we will use an endpoint with a query to the system database or Azure Cognitive Search. To configure Cognitive Search, first, we need to create a data source for each of the existing tables in CosmosDB, then create an Azure Cognitive Search index that defines the search fields from all the tables, and in the end, build an indexer that links the data source and index. After this, we can call the search service and send a query. The output from the function is, lastly, taken to the final step, where it will be composed a composed answer before sending the results to the user.

As an illustration, a user may inquire about ongoing projects and employees' vacations. Alternatively, they may ask for information about the employee assigned to work on project "x". Or, they may want to know which employees of project "x" are scheduled to take a vacation in April this year.

3.1.3 Dashboard Creation use case

In the Dashboard Creation use case, we intend to automate the creation of a workspace through its description in natural language, resolving three base problems:

- Significantly reducing the time the user takes to make the workspaces;
- If the user doesn't know how to do it, he just needs to express what he wants;
- The model may generate something the user doesn't know exists.

For this specific case, we thought of training a model to generate a workspace with characteristics and a structure the platform uses, giving him some examples to work over. As there is specific syntax to specify a workspace, components, and data sources, we will need a model that knows all those structures and how to orchestrate a dashboard. It is also necessary to bear in mind that the workspace will be a base from which the user may later have to work to make it to their liking, yet they have already saved a lot of work.

From the tests carried out in this regard, in Section 2.3.9, it was possible to understand that fine-tuning the model for this is possible and helps in the construction of dashboards, increasing the correctness of the solution. To do this, we have to be able to define a good model that allows LLM to know what combinations and where to place the components in the workspace structure.

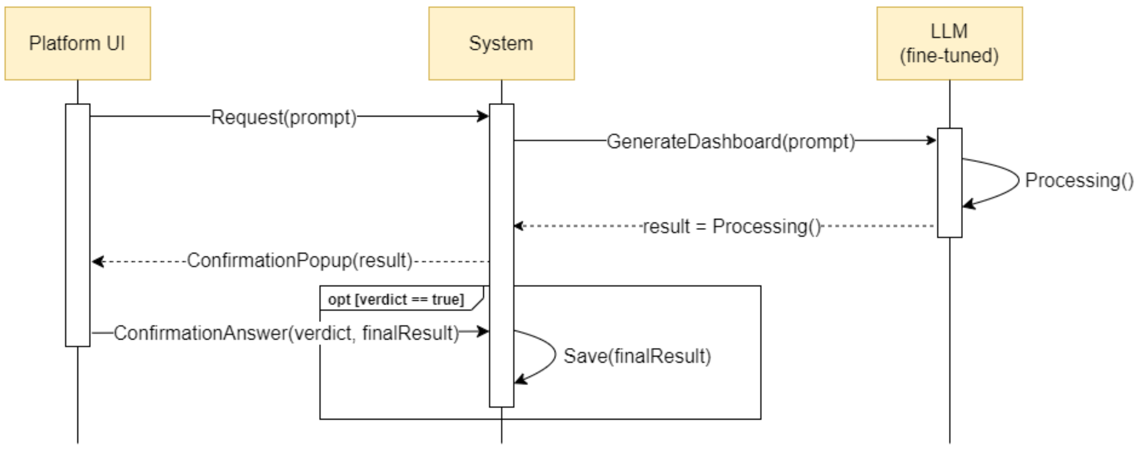


Figure 3.3: Sequence diagram for dashboard creation use case.

This use case will be inserted into the creation category. The function itself should ask the custom model to create a dashboard based on the user's request. After that, we send a confirmation request to the user showing the code for the dashboard so we can modify it if he wishes to. In the end, if the user confirms, we make a call for the endpoint that generates the dashboards.

For instance, when a user requests to create a dashboard displaying employees' availability and their respective project involvements, it must be translated into code that the system can operate. This may involve the creation of a table that lists all employees, with their respective projects displayed below their names. The result will be a dashboard that provides a clear overview of employee availability and current project assignments.

3.1.4 Problems Notification use case

Problems Notification would be new in the system. What is meant by notification of "problems" is simply that the user can define something that he wants to be notified about and is related to data in the system. For instance, whenever the number of working hours of an employee exceeds 20, the user receives a notification. The use case allows us to help the user to always be up to date with everything happening on the platform without having to constantly check manually certain data. With this, the user just needs to add a new problem.

To implement this function, we will insert it in the analysis category. When the user sends the request for the problem creation, it will first look at the saved ones, checking for similarities. If it has one, a popup appears with the code for the one saved to make a

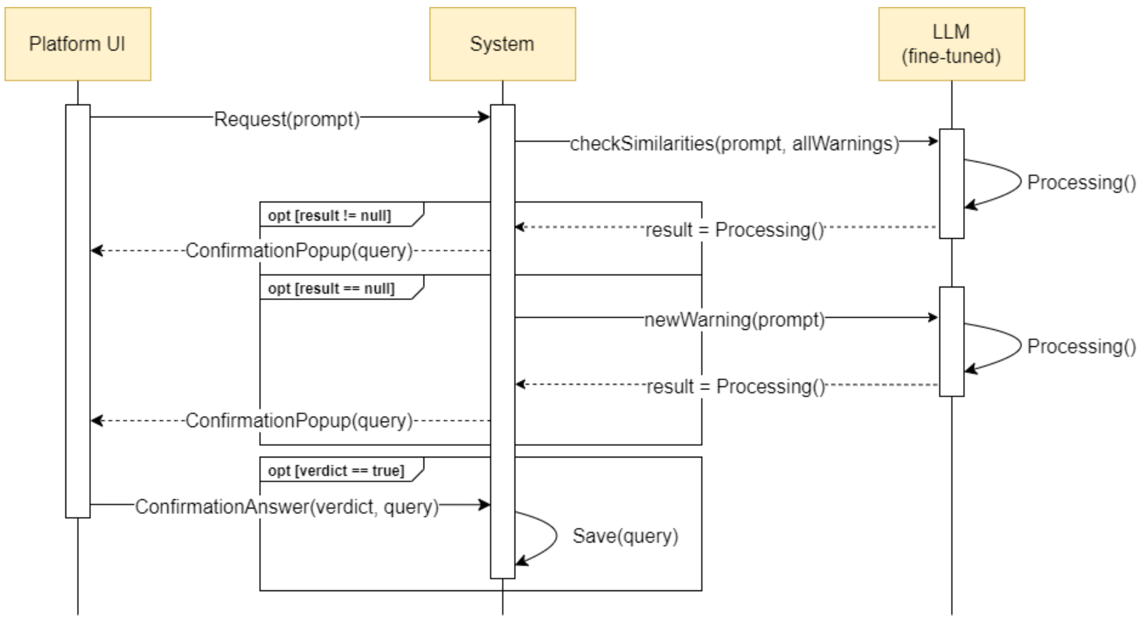


Figure 3.4: Sequence diagram for problems notification use case.

confirmation. Otherwise, it will generate one and then show the popup to confirm. These popups always have a timer to tell how often they will run. In case the user approves the warning, it will be saved, and the user will receive a notification when its execution returns positive. The saved problems will always be running periodically in system threads.

3.1.5 Summarize Project Information use case

This use case aims to extract project information such as tasks, sprints, and others. Once the information is gathered from our platform, it will be given to the LLM, which will generate a project summary, providing details about, for example, the project description, progress status, team members involved, whether it is on track to meet its deadline, etc. The summarization of a project may help the user have a quick look at the project's progress.

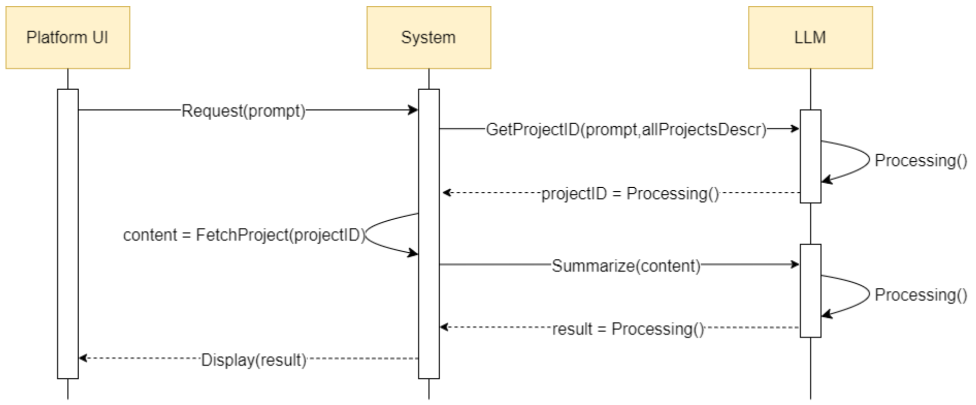


Figure 3.5: Sequence diagram for summarizing project information use case.

To perform this functionality, we will insert it in the content generation category. First, we need to ask the LLM to tell us, among all the projects, which one the user is referring to, returning its ID. Then, we use an endpoint to fetch the project information, and finally, ask the LLM to generate a project summary depending on the information the user wants to obtain. This last step can have a template to carry out a project summary for a more accurate and consistent answer.

3.1.6 Summarize Chat Talks use case

To Summarize Chat Talks will be inserted in the content generation category. Similar to Summarize Project Information, we aim to extract information, but this time from Skills Workflow chat discussions. Once the information is gathered from the chat the user intends, it will be given to the LLM, which will generate a summary, providing details about the conversation. Summarizing a conversation, we are helping the user have a quick overview of an overall conversation, not losing time scrolling in the chat.

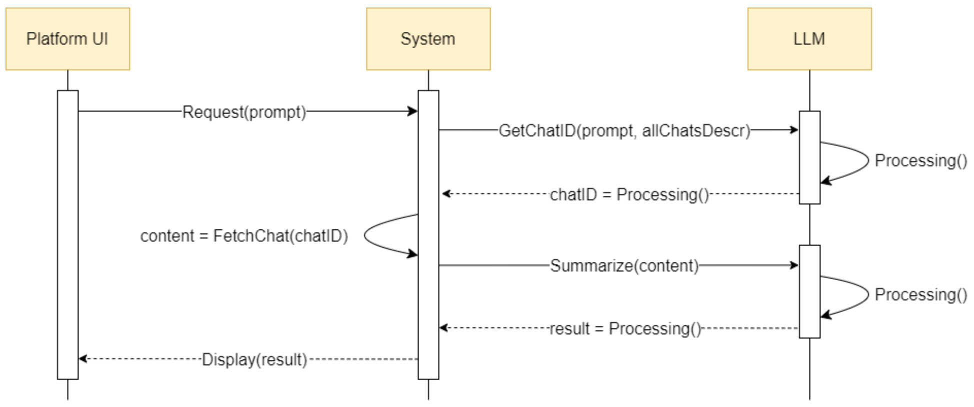


Figure 3.6: Sequence diagram for summarizing chat talks use case.

To perform this functionality, we first need to ask the LLM to tell us, among all the chat talks, which one the user is referring to, returning the ID. Then, we use an endpoint to fetch the chat information and in the end, ask the LLM to generate a chat summary.

3.1.7 Project Task Creation use case

The functionality of Project Task Creation enables the user to generate a task for a project in the micro-application. In order to create the task, the user must provide the system with relevant information about the task. Once the information is provided, the system generates a new task under the specified project. This saves the user's time as they don't have to manually create the task. Instead, the user can simply configure it by providing the necessary details to the system, by communicating in natural language.

To resolve this use case, we ask the LLM to generate the content of the specific fields for a task, needed in the endpoint that creates a new one. However, in the beginning, we need to know the project the user is referring to.

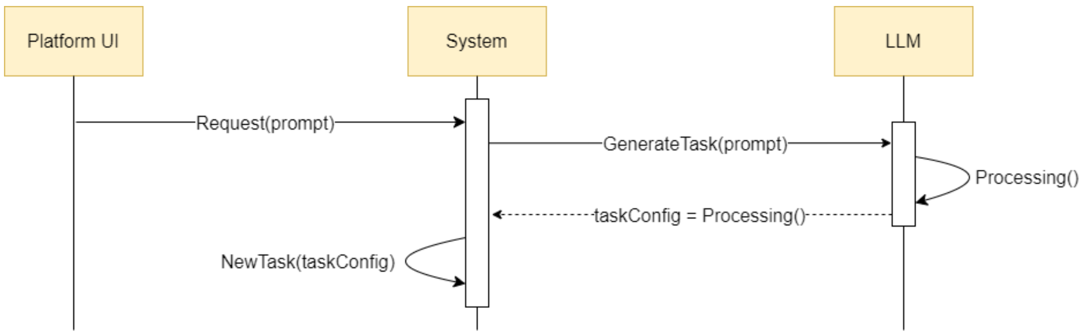


Figure 3.7: Sequence diagram for project task creation use case.

3.1.8 Visual Data Presentation use case

In this use case, we intend to transform responses given by the LLM into useful and easy-to-interpret information, in a visual data presentation, with graphics, tables, etc. Allowing the user to save pertinent information, interpret it easily, or share it with someone else. The user can experience a quick and intuitive view of the data almost instantly. For instance, imagine that the user has done a search about the employees in project "x" and the number of hours of tasks accomplished by each one, and instead of the user reading the concrete data in some text-readable format, we can find it faster in an organized table.

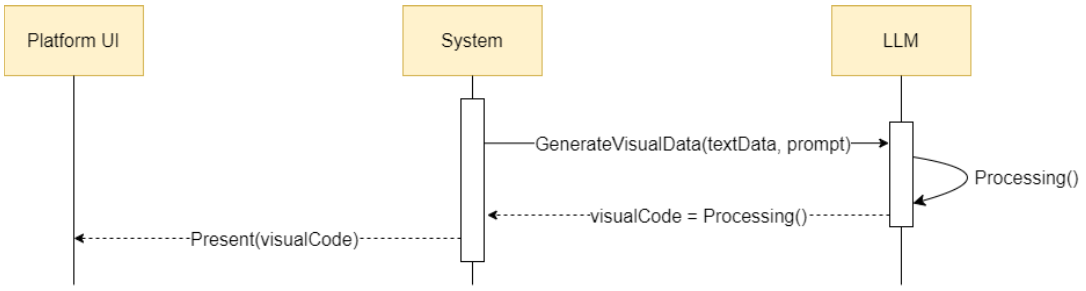

Platform UI
System
LLM
GenerateVisualData(textData, prompt)
Processing()
visualCode = Processing()
Present(visualCode)

Figure 3.8: Sequence diagram for visual data presentation use case.

This function would be framed in the analysis category and executed after a search of the platform's data related to the Data Analysis use cases approached in Section 3.1.2. The data returned will be sent joined with the prompt to the LLM, so it can generate some code using the components of Skills Workflow (e.g., graphics, tables) to create a visual representation of the data. Then, the system exposes that data to the client in a popup window.

3.1.9 Resource Profiling use case

Resource Profiling should perform the analysis of the system's resource usage, identifying performance setbacks or obstacles that are slowing a process in our system, optimizing resource utilization, and improving efficiency. The resources can be employees or project billings, where the user wants to have some information about its usage, defining some sort of profile a certain resource should sustain. One example is getting the employees inserted in a project that does not have much work and, therefore, be relocated to another project. The system should return a notification if the query created for the problem is verified.

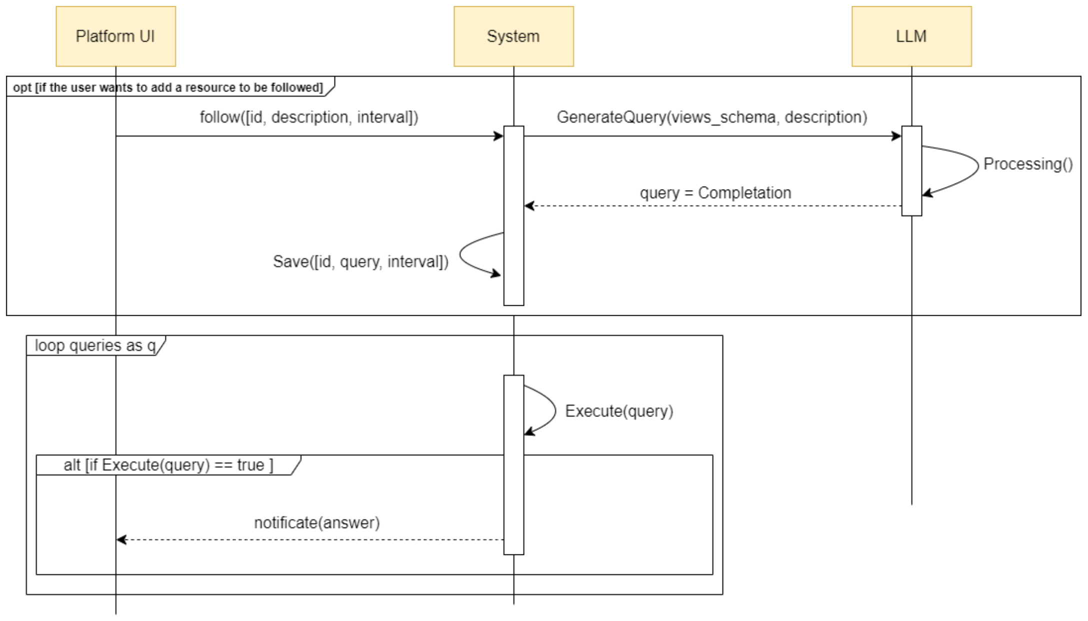

Platform UI
System
LLM
opt [if the user wants to add a resource to be followed]
follow([id, description, interval])
GenerateQuery(views_schema, description)
Processing()
query = Completetation
Save([id, query, interval])
loop queries as q
Execute(query)
alt [if Execute(query) == true]
notificate(answer)

Figure 3.9: Sequence diagram for resource profiling use case.

Resource Profiling falls under content generation because it generates notifications to the user. The user can program the system for the resources he wants to follow and add an interval to execute it. The system needs to generate a query for that using the LLM. Then, the system notifies the user any issue is detected, and the user can view the case details in the notification zone.

3.1.10 Allocation of People use case

Allocation of People will be a use case that permits us to tell the system things like, "If a person has exceeded the work time per week, please re-distribute tasks from a project so that everyone has equal work time" or simply, "I need you (system) to put Jhon Thornefield on the project x". This functionality would make life easier for the user, as they would not have to go to each project distributing and allocate members between projects. It would require the system to understand where it would move/add members from and what fields it would have to modify or create. Here, we need to instruct the LLM to analyze the definition of the dashboards and ensure that after the modification is made the system can save the changed data.

Allocation of People should be inserted in the content generation category because it manages the system state. This function will ask the LLM to allocate the person to its new duty, like a new task or project. A person can be removed from a project or added to one,

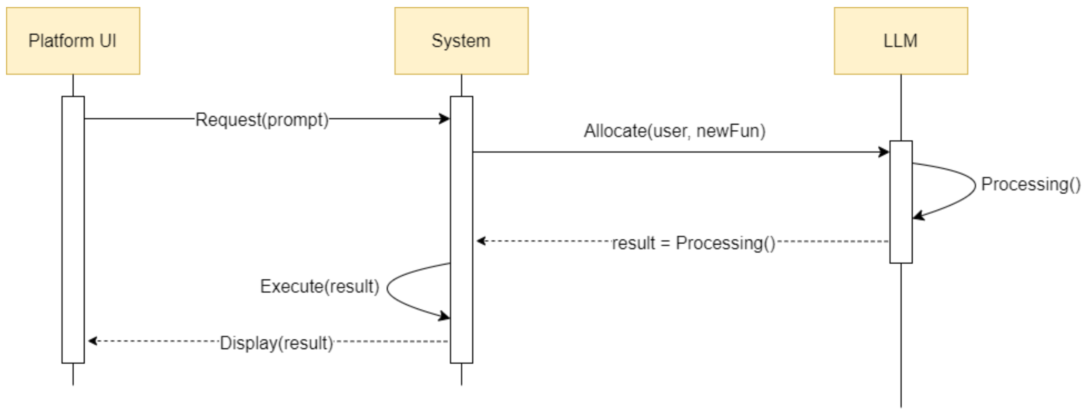


Figure 3.10: Sequence diagram for allocation of people use case.

which changes the data in the system, by calling the endpoint returned from the LLM, needed to generate the necessary information to call it also.

3.1.11 Generate Tasks use case

This use case aims to facilitate the creation of new tasks for projects. It saves the user time by automatically generating and adding the task to the correct project. The Generate Tasks use case falls under the creation category as it creates a new task based on the user's request.

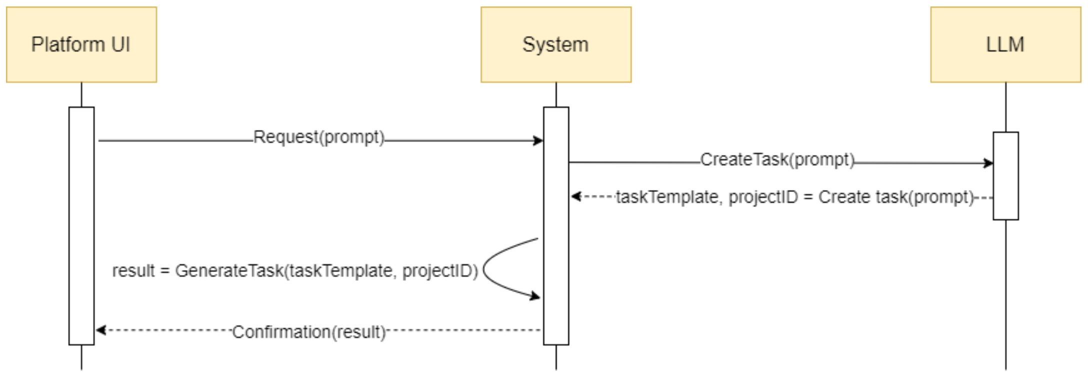

Platform UI
System
LLM
Request(prompt)
CreateTask(prompt)
taskTemplate, projectID = Create task(prompt)
result = GenerateTask(taskTemplate, projectID)
Confirmation(result)

Figure 3.11: Sequence diagram for generating tasks use case.

To develop this use case the user will make a request to the system which interacts with the LLM to generate a task JSON template code plus the ID of the project the user is referring to. After that, the system adds a new task and confirms it to the user.

3.1.12 Materials Extraction use case

In order to obtain a file of information from our platform, we use a process called Material Extraction. This involves requesting the necessary data from the system and receiving a document in the format of the user's choice. The user can choose whether they want the document in Word or Excel, and the system will automatically generate it with the searched data. This feature will be included in the analysis category, looking for material in the system. It will prove to be an extremely useful feature for users who want to save or share the information they have obtained from the platform.

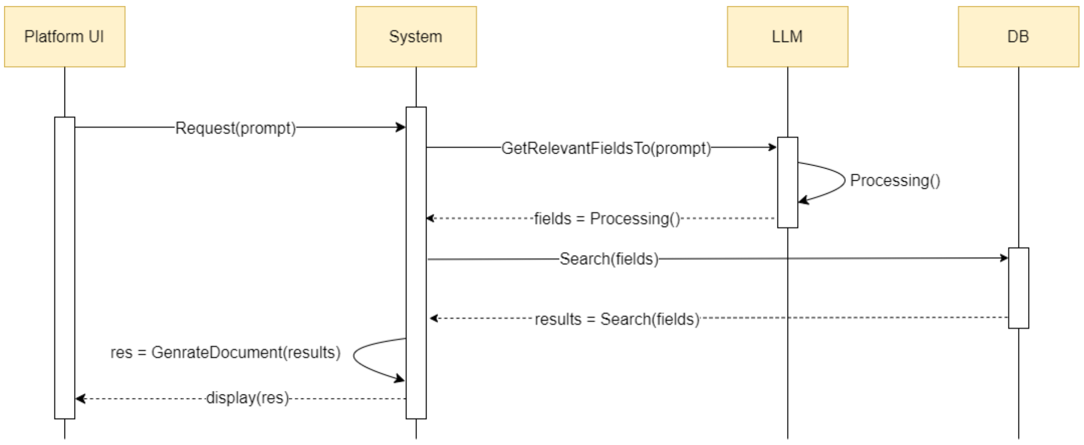


Figure 3.12: Sequence diagram for materials extraction use case.

3.1.13 Search Data from Images use case

This use case consists of using the language model that permits capturing an image and then being asked something about the image itself, where the model will get the answer from the data in the system projections of the DB. Imagine the user has an impressed document with a table, and in it is specified a user proposed vacations. He can take a picture and the system can look at that image data and respond with something related to that information. For instance, if the user's vacations are already marked or marked differently from what is in the image.

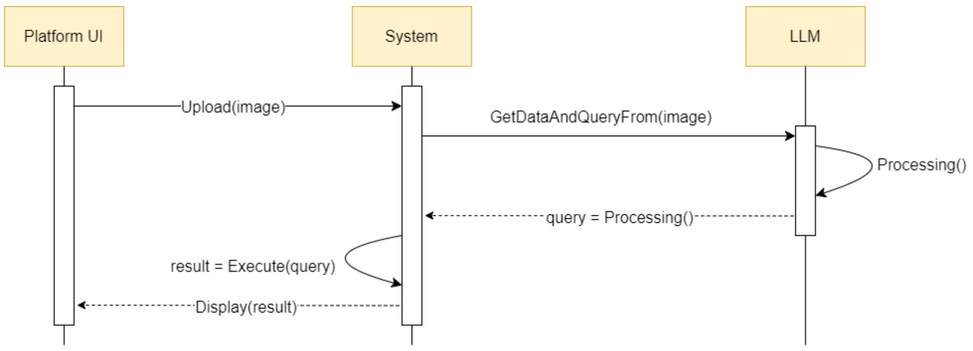


Figure 3.13: Sequence diagram for search data from the image use case.

To implement this use case, the user first needs to upload their image. Then, the image model will analyze the image and retrieve the data from it. The system will then use a query to search for relevant data from the system database projections. This use case falls under the analysis category.

3.1.14 Digest Data From Images use case

This use case involves using the language model capable of capturing images and requesting the system to perform actions based on the data obtained. It can be added, for instance, a task for a project that was a user story written by the client.

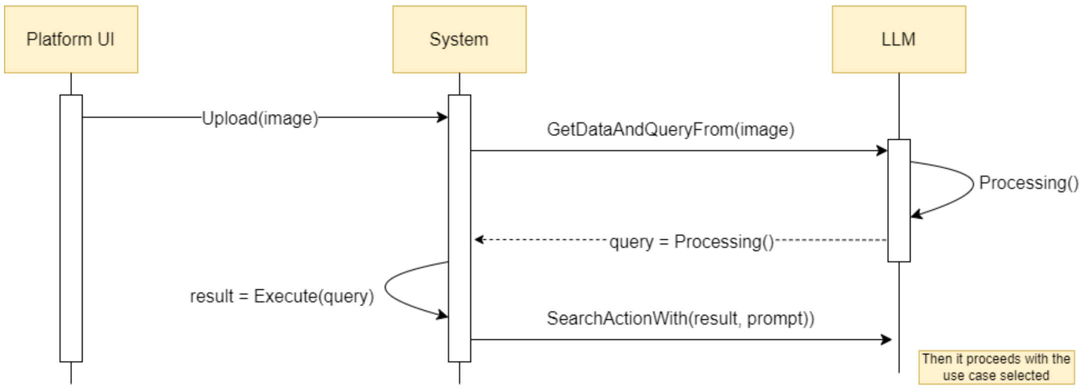

Platform UI
System
LLM
Upload(image)
GetDataAndQueryFrom(image)
Processing()
query = Processing()
result = Execute(query)
SearchActionWith(result, prompt))
Then it proceeds with the use case selected

Figure 3.14: Sequence diagram for digesting data from the image use case.

The Digest Data from Images use case falls under the creation category because it generates new data based on the user's request. This use case aims to retrieve data from an image and perform other possible functions that have already been addressed. Please consult Figure 3.1.

3.1.15 Project Creation use case

This use case scenario involves the creation of a new project on the platform. The user provides a description of the project, and the LLM uses the provided information to compose a new project. Additionally, the will LLM include any other necessary information to complete the creation process. To create a new project, we may need to train a model that can generate new project JSON code in a format that the system can understand by making it adapt a clean template of an empty project. This use case will be categorized under project creation since it involves building a new project from scratch based on user input.

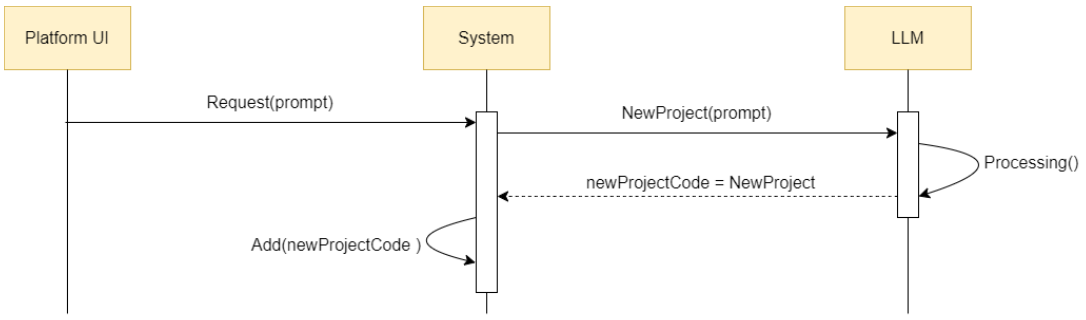


Figure 3.15: Sequence diagram for project creation use case.

3.2 Execution plan

To carry out the proposed solution, we start by analyzing a good LLM. After our searches, we understand that the differences between OpenAI and Google are not huge. In terms of quality/cost, both are good options. GPT-3.5 Turbo and Gemini Pro have similar performance and similar prices, only the GPT-4 is more efficient but also more expensive. For Skills Workflow, what we are more inclined to are the models from OpenAI, for two main reasons. First, we can alternate between GPT-4 and GPT-3.5 easily according to different domains of certain tasks, saving some costs using GPT-3.5 Turbo sometimes. Second, OpenAI is already integrated into Azure, where Skills Workflow is hosted and probably has good compatibility.

Firstly, the architecture to implement the assistant should be defined, followed by the implementation of the assistant functionalities, starting with the use cases related to summarization, and after the data analysis and problem notification (since the assistant should be incremental, we will develop as much as possible). In the end, some tests should be performed to check the assistant's performance and if it is working as expected.

To achieve the goals and build the *Skillio Assistant*, we will attempt to follow the pursuing work plan:

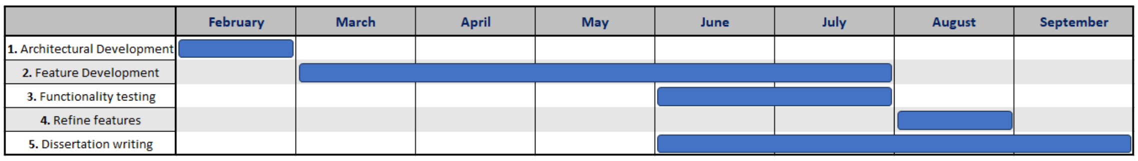


Figure 3.16: Solution Roadmap.

Bibliography

[1] *AgentHub*. Accessed February 9, 2024. URL: https://www.agenthub.dev (cit. on p. 16).

[2] S. N. Akter et al. "An In-depth Look at Gemini's Language Abilities". In: (2023-12). URL: http://arxiv.org/abs/2312.11444 (cit. on p. 12).

[3] B. Banjara. *LLMs in Conversational AI: Building Smarter Chatbots Assistants*. Accessed January 9, 2024. 2023-07. URL: https://www.analyticsvidhya.com/blog/2023/07/llms-in-conversational-ai/ (cit. on p. 6).

[4] M. Bolanos. *Migrating from the Sequential and Stepwise planners to the new Handlebars and Stepwise planner*. Accessed January 15, 2024. 2023-12. URL: https://devblogs.microsoft.com/semantic-kernel/migrating-from-the-sequential-and-stepwise-planners-to-the-new-handlebars-and-stepwise-planner/ (cit. on p. 20).

[5] S. Caixa Geral de Depósitos. *Caixadirecta*. Accessed December 6, 2023. URL: https://www.cgd.pt/Particulares/Contas/Caixadirecta/Pages/Assistente-Digital-App-Caixadirecta.aspx (cit. on p. 16).

[6] M. Corporation. *Creating AI agents*. Accessed January 4, 2024. URL: https://learn.microsoft.com/en-us/semantic-kernel/overview/ (cit. on p. 18).

[7] M. Corporation. *Creating AI agents*. Accessed January 4, 2024. URL: https://learn.microsoft.com/en-us/semantic-kernel/agents/ (cit. on p. 18).

[8] M. Corporation. *Creating AI agents*. Accessed January 5, 2024. URL: https://jordanbeandev.com/how-to-build-your-own-chatbot-using-c-semantic-kernel-azure-openai-part-1/ (cit. on p. 20).

[9] M. Corporation. *Descrição geral do Microsoft Copilot para Microsoft 365*. Accessed January 9, 2024. 2023-11. URL: https://learn.microsoft.com/pt-pt/microsoft-365-copilot/microsoft-365-copilot-overview (cit. on p. 16).

[10] M. Corporation. *Export plugins written for Semantic Kernel as an OpenAI plugin*. Accessed January 5, 2024. URL: https://learn.microsoft.com/en-us/semantic-kernel/agents/plugins/openai-plugins (cit. on p. 19).

[11] M. Corporation. *What is the Bot Framework SDK?* Accessed January 24, 2024. 2022-11. URL: https://learn.microsoft.com/en-us/azure/bot-service/bot-service-overview?view=azure-bot-service-4.0 (cit. on p. 21).

[12] M. Corporation. *What is the Bot Framework SDK?* Accessed January 24, 2024. 2022-11. URL: https://www.trustradius.com/compare-products/azure-bot-service-vs-google-cloud-dialogflow#community-pulse (cit. on p. 21).

[13] M. Corporation. *What will you do with Copilot with Bing*. Accessed on February 6, 2024. URL: https://www.microsoft.com/en-us/bing?ep=0&form=MA13LV&es=31 (cit. on p. 1).

[14] Google. *You May be Fine-Tuning Google Bard (Without You Knowing it)*. Accessed December 20, 2023. URL: https://support.google.com/bard/answer/13594961?visit_id=638386644607786805 (cit. on p. 10).

[15] time haaify. *Open Source vs Proprietary LMS: How Do I Choose?* Accessed January 20, 2024. 2019-10. URL: https://www.lambdasolutions.net/blog/open-source-vs-proprietary-lms-how-do-i-choose (cit. on p. 8).

[16] time haaify. *Qual é a diferença entre bot, chatbot e assistente virtual*. Accessed January 20, 2024. 2022-05. URL: https://haaify.com/blog/diferenca-bot-chatbot-assistente-virtual/ (cit. on p. 5).

[17] M. Kapronczay. *A Beginner's Guide to Language Models*. Accessed January 19, 2024. 2022-12. URL: https://builtin.com/data-science/beginners-guide-language-models (cit. on p. 7).

[18] Microsoft. *Azure OpenAI Service models*. Accessed December 20, 2023. URL: https://learn.microsoft.com/en-US/azure/ai-services/openai/concepts/models (cit. on p. 10).

[19] H. Naveed et al. "A Comprehensive Overview of Large Language Models". In: (2023-07). URL: http://arxiv.org/abs/2307.06435 (cit. on p. 7).

[20] I. NGUYEN. *When and How to Train Your Own Language Model*. Accessed January 19, 2024. 2022-08. URL: https://www.deepset.ai/blog/when-and-how-to-train-a-language-model (cit. on p. 7).

[21] T. Omoyeni. *Introducing Chatbots and Large Language Models*. Accessed January 9, 2024. 2023-12. URL: https://www.sitepoint.com/introducing-chatbots-and-large-language-models-llms/ (cit. on p. 6).

[22] OpenAI. *Common use cases*. Accessed December 6, 2023. URL: https://platform.openai.com/docs/guides/fine-tuning/common-use-cases (cit. on p. 8).

[23] OpenAI. *Customer stories*. Accessed December 17, 2023. URL: https://openai.com/customer-stories (cit. on p. 16).

[24] OpenAI. *Fine Tuning*. Accessed December 6, 2023. URL: https://platform.openai.com/docs/guides/fine-tuning (cit. on p. 8).

[25] OpenAI. *Moderation*. Accessed December 20, 2023. URL: https://platform.openai.com/docs/guides/moderation/overview (cit. on p. 9).

[26] OpenAI. *Prompt engineering*. Accessed December 23, 2023. URL: https://platform.openai.com/docs/guides/prompt-engineering (cit. on p. 9).

[27] OpenAI. *Stripe*. Accessed December 6, 2023. 2023-03. URL: https://openai.com/customer-stories/stripe (cit. on p. 16).

[28] OpenAI. *Viable*. Accessed December 17, 2023. 2023-03. URL: https://openai.com/customer-stories/be-my-eyes (cit. on p. 16).

[29] OpenAI. *What are tokens*. Accessed January 3, 2024. URL: https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them?q=re (cit. on p. 14).

[30] H. Sajid. *As 10 principais vulnerabilidades do LLM*. Accessed January 10, 2024. 2023-09. URL: https://www.unite.ai/pt/As-10-principais-vulnerabilidades-do-LLM/ (cit. on p. 8).

[31] V. Shevchuk. *GPT-4 Parameters Explained: Everything You Need to Know*. Accessed December 20, 2023. 2023-05. URL: https://levelup.gitconnected.com/gpt-4-parameters-explained-everything-you-need-to-know-e210c20576ca (cit. on p. 9).

[32] N. Shukla. *LLMs vs. Traditional Language Models: A Comparative Analysis*. Accessed January 10, 2024. 2023-09. URL: https://www.appypie.com/blog/llms-vs-traditional-language-models (cit. on p. 7).

[33] Statistopedia. *You May be Fine-Tuning Google Bard (Without You Knowing it)*. Accessed November 7, 2023. 2023-09. URL: https://statistopedia.com/google-bard/you-may-be-fine-tuning-google-bard-without-you-knowing-it/ (cit. on p. 10).

[34] D. H. Sundar Pichai. *Introducing Gemini: our largest and most capable AI model*. Accessed Fevereiro 7, 2023. 2023-12. URL: https://blog.google/technology/ai/google-gemini-ai/#sundar-note (cit. on p. 10).

[35] G. TARRAF. *Everything you need to evaluate open-source (vs. closed-source) LLMs*. Accessed January 20, 2024. 2023-12. URL: https://atelier.net/insights/evaluating-open-source-large-language-models (cit. on p. 8).

[36] G. Team. *Gemini: A Family of Highly Capable Multimodal Models*. 2023 (cit. on p. 11).

[37] V. S. Team and M. Murgia. *Generative AI exists because of the transformer*. Accessed November 7, 2023. 2023-09. URL: https://ig.ft.com/generative-ai/ (cit. on p. 7).

[38] H. Touvron et al. "Llama 2: Open Foundation and Fine-Tuned Chat Models". In: (2023-07). URL: http://arxiv.org/abs/2307.09288 (cit. on p. 11).

[39] *typescript-instruct*. Accessed December 23, 2023. URL: https://huggingface.co/datasets/bleugreen/typescript-instruct (cit. on p. 9).

[40] Wikipedia. *Language model*. Accessed January 19, 2024. URL: https://en.wikipedia.org/wiki/Language_model (cit. on p. 7).

[41] Wikipedia. *Office Assistant*. Accessed January 17, 2024. URL: https://en.wikipedia.org/wiki/Office_Assistant (cit. on p. 1).

[42] S. Workflow. Accessed January 9, 2024. URL: https://www.skillsworkflow.com (cit. on p. 6).