Tunisian Republic

Ministry of Higher Education and Scientific Research

Carthage University- Higher School of Statistics and Information Analysis

Graduation Project Report submitted to obtain the degree of

National Diploma of Engineering in Statistics and Data Analysis

Mahmoud LARGAT

Coding Assistant

Defended on 10/06/2024 in front of the committee composed of :

— *Mrs. Fatma Chaker, Assistant Professor and Director of the University (President)*

— *Mr. Amine Sethom, Managing Director(Company supervisor)*

— *Mrs. Aicha ElGolli, Assistant Professor (University supervisor)*

— *Mrs. Tasnim Hamdeni, Assistant Professor (Reviewer)*

End of studies internship completed at

Infinity Management

Academic year 2023-2024

Thanks

As a token of appreciation to all those who have guided, listened, and advised us, I would like to use this page to express my gratitude. At the conclusion of this internship, I wish to extend my sincere thanks and appreciation to my supervisor, "Amine Sethom", Managing Director at Infinity Management Tunis. Thank you for your valuable guidance and your undeniable interest in the success of this project over the past six months.

I also take this opportunity to express my respect and gratitude to my academic supervisor, "Aicha El Golli", Assistant Professor in Computer Science, Director of Studies and Internships At ESSAI. Thank you for the trust you have placed in me throughout this internship period, allowing me to work independently, as well as for your insightful advice, kindness, and encouragement.

I would like to extend my sincere thanks to our CEO, "Walid Bey", our CTO, "Amine Sethom", and our HR manager, "Roukaya Arfaoui", for your leadership, strategic vision, technical expertise, and human resources management, all of which are key elements contributing to project success. I am grateful for the opportunity to work under your guidance.

I also thank the members of the jury for agreeing to evaluate my graduation project. I hope to meet their expectations.

I also take this opportunity to express my sincere thanks to all the staff at Infinity Management Tunis for their kindness and willingness to answer my questions. Thank you for making this internship period an enriching experience.

Abstract

This project focuses on fine-tuning a large language model (LLM) to generate Django framework code from JSON outputs. By automating the conversion of structured JSON data into functional Django code, we aim to streamline the development process, enhance productivity, and reduce manual coding efforts.

keywords: NLP, LLM,Fine-tuning, LoRA,NN,AI,PEFT

Résumé

Ce projet se concentre sur le fine-tuning d'un grand modèle de langage (LLM) pour générer du code pour le framework Django à partir de sorties JSON. En automatisant la conversion de données JSON structurées en code Django fonctionnel, nous visons à rationaliser le processus de développement, améliorer la productivité et réduire les efforts de codage manuel.

keywords: NLP, LLM, Fine-tuning, LoRA, NN, AI, PEFT

Contents

List of Figures

General Introduction

In our capstone project, we delve into the integration of artificial intelligence (AI) to enhance business processes within Infinity Management. AI presents unparalleled opportunities for automating tasks and boosting operational efficiency, allowing organizations to streamline workflows, minimize manual labor, and enhance decision-making across diverse sectors. The core focus of our study revolves around identifying and examining the current hurdles in business process management and delving into AI-driven remedies to surmount these challenges.

The primary objectives of our project include assessing the existing state of business processes within Infinity Management, pinpointing areas for enhancement through AI applications, crafting and deploying AI-powered solutions to optimize these processes, and assessing the impact of these solutions on the organization's effectiveness and productivity. This endeavor carries both practical and theoretical significance: it seeks tangible enhancements in Infinity Management's operations while also contributing to the evolution of knowledge in the realm of AI utilization in business process management.

The structure of this report unfolds as follows: an introduction to Infinity Management and the overarching project context. The second chapter is about an exploration of fundamental AI concepts and technical decisions. The third chapter is a comprehensive analysis of the identified problem areas, subsequent design and execution of

AI-based remedies, and ultimately. Finally an assessment of the achieved outcomes accompanied by a discourse on the implications and future directions.

Chapter 1

General Project Framework

In this chapter, we will place our work within its broader context. First, we briefly introduce the internship environment by describing the host organization, Infinity Management Tunisia. Next, we discuss the scope of the project by detailing the analysis and critique of the existing system, as well as the proposed solution. Then, we list the objectives to be achieved. Finally, we explain the adopted development methodology.

1.1 Host organization

The hosting organization where I completed my internship provided me with an opportunity for learning and experience by allowing me to integrate into its environment and contribute to its activities.

1.1.1 Infinity Management

Infinity Management is a digital engineering platform that assists its clients and partners in designing and implementing powerful solutions to address the challenges of innovation and change. Its objective is to unleash the potential of technology to en-

hance customer knowledge and user experience. With offices in Tunis, Paris, and Madrid, Infinity Management specializes in highly innovative subjects, including business intelligence, robotics and artificial intelligence (RPA), blockchain, Know Your Customer (KYC), and software design and architecture (web and mobile applications, enterprise Data Management (EDM), Business Process Management(BPM), etc..

Figure 1.1: Logo Infinity Management [1]

1.1.2 The values of Infinity Management

Infinity Management collaborates with major players in various sectors such as banking, finance and microfinance, insurance, real estate, retail, services, and the public sector, with significant flexibility to also engage in other sectors. Its main mission is to assist clients and partners in their technological transformation by providing skills as well as technical and managerial expertise around the four elements illustrated in Figure 1.2.

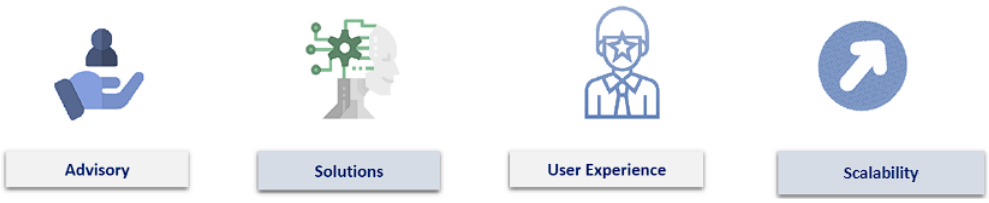

Figure 1.2: The values of Infinity Management [1]

- Advisory Consil : Addressing technological issues judiciously through their expertise ensures that they can provide their clients and partners with optimal solutions tailored to their needs.

- Scalability : Implementing scalable features and databases that can handle large volumes of requests is essential. Clients rely on their expertise to manage their data according to their size and the evolution of their business.

1.1.3 The services provided

Thanks to its innovation-driven culture, Infinity Management has developed exceptional expertise that enables it to consider all elements to offer its partners and clients the most suitable and operational solutions, particularly in the following areas :

- Solutions : Construction of integrated web and mobile platforms utilizing the latest technologies (machine learning, artificial intelligence, blockchain), and managing the entire project process: Analysis, Architecture, Implementation, Testing.

- Automatisation des processus : Robotic process automation, Electronic document management, Business process management, etc.

- Intelligence : Dashboarding, Data Warehouse, Reporting, Data Analysis, Data Mining.

- Talents : HR Consulting, HR Development, Recruitment Process Outsourcing, Manager Assessment, Headhunting, Executive Search.

1.1.4 Infinity Management's Approach

The following table 1.1 illustrates Infinity Management's approach:

1.2 Contexts and challenges

This section aims to present the problem and describe our solution.

Approach	Description
Consulting	They offer practical solutions to your technological challenges.
User Experience	They build engaging interfaces centered around user experience.
Scalability	They implement features capable of handling high volumes of requests.

Table 1.1: Infinity's Approach

1.2.1 Studies and critiques of the existing state

In the realm of this artificial intelligence end-of-studies project, our focus pivoted towards conducting a meticulous analysis of developers' workflows, particularly those entrenched in the utilization of the Django framework. Our primary endeavor was to gain a comprehensive understanding of the intricate processes involved in project development, along with a keen eye on the array of tools wielded by developers within this ecosystem. Through this analysis, we aimed to unearth key pain points and inefficiencies, thereby laying the groundwork for the subsequent exploration of opportunities for automation.

Delving deeper into our research, we sought to identify specific tasks and processes within the development life cycle that exhibited potential for automation. By scrutinizing the various stages of project development, from initial planning and design to implementation and deployment, we endeavored to pinpoint repetitive or labor-intensive tasks that could benefit from AI-powered automation. Furthermore, we took into account the unique characteristics and requirements of projects built on the Django framework, tailoring our analysis to the specific needs of developers operating within this environment.

Armed with insights garnered from our analysis, we embarked on the formulation

of AI-driven solutions designed to streamline developers' workflows and bolster productivity. Our aim was not only to alleviate the burden of manual tasks but also to empower developers with tools capable of augmenting their capabilities and efficiency. By harnessing the potential of artificial intelligence, we aspired to pave the way for a more seamless and efficient development process within the Django community, ultimately contributing to the advancement of software development practices in the age of AI.

1.2.2 Central Issue

In the realm of web development, particularly within the Django framework at Infinity Management, a central issue arises concerning the prevalence of repetitive tasks and code segments that consume significant time and resources. One of the most prominent examples of this challenge is encountered in the process of boilerplate code writing. Boilerplate code, which refers to the standardized, repetitive sections of code that are necessary for the functioning of a web application but do not contribute directly to its unique functionality, can be particularly time-consuming to write and maintain.

Developers at Infinity Management often find themselves spending considerable amounts of time crafting boilerplate code for tasks such as setting up models, views, and URL configurations in Django projects. This repetitive nature of code writing not only hampers productivity but also detracts from the time that could be better spent on more complex and intellectually stimulating aspects of web development, such as designing innovative features or optimizing performance.

Furthermore, the need for boilerplate code extends beyond just the initial development phase. As projects evolve and requirements change, developers must continually update and modify these repetitive code segments, adding to the maintenance overhead and further exacerbating the strain on resources. Recognizing the inefficiencies inherent in this process, Infinity Management is exploring the integration of

artificial intelligence (AI) to automate the generation and management of boilerplate code in Django projects. By harnessing AI-powered code generation tools, we aim to streamline the development process, reduce manual effort, and free up valuable developer time for more strategic and high-impact tasks. This approach not only enhances efficiency but also empowers our developers to focus on activities that drive innovation and value creation, ultimately positioning Infinity Management as a leader in Django web development.

How can Infinity Management effectively integrate AI-powered code generation tools into Django development workflows to alleviate the burden of repetitive boilerplate code writing and enable developers to focus on higher-value tasks?

1.2.3 AI: Innovating Workflows, Inspiring Change

The adoption of artificial intelligence (AI) in various domains stems from its transformative potential to revolutionize traditional processes and workflows. As industries grapple with challenges such as repetitive tasks, resource constraints, and the need for innovation, AI emerges as a powerful solution. The idea to implement AI often arises from recognizing the limitations of manual approaches and the immense opportunities presented by advanced technologies. Drawing inspiration from developments in AI research, particularly in areas like machine learning and natural language processing, organizations seek to leverage these capabilities to automate tasks, improve efficiency, and drive innovation. The decision to embrace AI reflects a broader trend of harnessing technology to solve complex problems, optimize resource utilization, and stay competitive in rapidly evolving markets. It underscores a strategic commitment to leveraging cutting-edge solutions to unlock new possibilities and drive sustainable growth in the digital age.

1.2.4 Solution

To address the challenge of repetitive boilerplate code writing in Django development at Infinity Management, we are exploring the implementation of an innovative AI solution. This solution entails the development of an AI-powered application, leveraging Large Language Models (LLMs), wherein developers can specify the desired task, models, views, and serializers. The application then utilizes natural language processing (NLP) algorithms to interpret these specifications and generates a complete directory containing the corresponding code files.

By utilizing LLMs, pre-trained models and fine-tuning them, the AI application can understand developer input in natural language and generate accurate code structures accordingly. Developers can simply describe their requirements in plain language, such as defining models, specifying views, and outlining serializers, without needing to manually write or modify code.

This AI-powered solution not only automates the tedious process of boilerplate code writing but also ensures consistency and accuracy in code generation. Additionally, it significantly reduces the time and effort required for setting up Django projects, allowing developers at Infinity Management to focus on more challenging and innovative tasks.

By embracing this AI-driven approach, Infinity Management aims to revolutionize Django development workflows, enhance productivity, and empower developers to deliver high-quality web applications efficiently. Moreover, the implementation of such a solution demonstrates our commitment to leveraging cutting-edge technologies to drive continuous improvement and innovation within our development processes.

1.3 Workflow: Agile Development

Agile methodology, commonly used in software development, proves highly effective when applied to Artificial Intelligence (AI) projects. Its iterative approach allows for continual improvement and adaptation, which is invaluable in the dynamic landscape of AI. When employing agile in AI, I found it particularly useful in the model review process [2].

Rather than sticking to a rigid plan, agile encourages flexibility. If initial model results were less than satisfactory, I didn't dwell on setbacks but instead focused on enhancing data processing and refining subsequent steps. This iterative cycle of review, adjustment, and reevaluation ensures that the project stays on track towards achieving optimal results [2].

The strength of agile lies in its emphasis on collaboration and responsiveness. By breaking down tasks into manageable chunks and regularly reviewing progress, I could quickly identify areas needing improvement and make necessary adjustments. This constant feedback loop not only accelerates the pace of development but also minimizes the risk of heading down the wrong path for too long [2].

Moreover, agile methodology fosters a culture of continuous learning and adaptation. As I encountered challenges and setbacks, I viewed them as opportunities for growth rather than roadblocks. By remaining open to change and embracing uncertainty, I was able to navigate through complexities and steer the project towards success.

In conclusion, agile methodology is well-suited for AI projects due to its iterative nature and focus on adaptability. By leveraging its principles, I could efficiently review and refine models until achieving satisfactory results. The ability to respond quickly to feedback and make incremental improvements is paramount in the ever-evolving field of AI, making agile an indispensable approach [2].

1.4 Conclusion

In this chapter, we focus on the various fundamental aspects of our final year project. We begin by discussing the field of artificial intelligence in general, then we delve into the second aspect: the large language models (LLM).

Chapter 2

Basic concept and technical choices

This chapter focuses on the various aspects underlying our end-of-studies project. We first address the aspect of Artificial Intelligence, providing a general overview of this field, before moving on to the second aspect: Large Language Models (LLMs).

2.1 General Introduction To AI

2.1.1 Neural Network

A neural network is a computational model inspired by the structure and function of the human brain and was initially conceptualized by Warren McCulloch and Walter Pitts in the 1940s. This network is comprised of interconnected nodes, known as neurons, that are organized into layers. The flow of information within the network starts at an input layer where data is inputted, moves through one or more hidden layers where computations take place, and ultimately arrives at an output layer where the network generates predictions or classifications. The connections between neurons have specific weights that determine the strength of the connection. Throughout the training process, the network adjusts these weights based on input data and desired

outputs, a technique referred to as backpropagation. Through repetitive training on a dataset, the neural network gains the ability to identify patterns and make accurate predictions, thereby proving to be a valuable tool for various tasks such as image recognition, natural language processing, and more. [3].

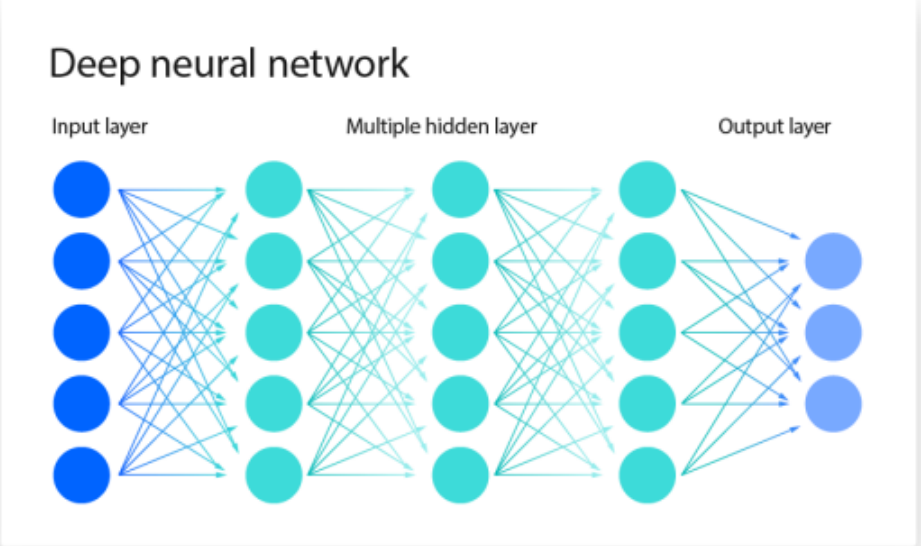

Figure 2.1: Neural Network [3]

2.1.2 AI

Artificial Intelligence (AI) involves replicating human intelligence in machines, enabling them to undertake tasks that traditionally rely on human intelligence. These tasks encompass learning, reasoning, problem-solving, perception, understanding natural language, and interacting with the environment. The field of AI encompasses a wide array of techniques and technologies, including machine learning, deep learning, natural language processing, computer vision, robotics, and other related areas. [4].

AI systems enhance their performance over time by learning from data without the need for explicit programming, continually refining their capabilities. Machine learning, a component of AI, encompasses algorithms that empower computers to discern patterns and make data-based decisions. Within machine learning, deep learning

stands as a subfield that utilizes multi-layered neural networks to represent and process intricate data structures. [4].

AI is utilized across a diverse range of fields, spanning healthcare, finance, transportation, education, entertainment, and beyond. Its applications include powering virtual assistants such as Siri and Alexa, recommendation systems akin to those employed by Netflix and Amazon, autonomous vehicles, medical diagnostic systems, and numerous other tools that bolster effectiveness, precision, and decision-making across various industries. [4].

2.1.3 Fine-Tuning

Fine-tuning in deep learning constitutes a type of transfer learning, where a pre-trained model, previously trained on a substantial dataset for a general task like image recognition or natural language understanding, undergoes slight adjustments to its internal parameters. The objective is to enhance the model's effectiveness for a new, associated task without commencing the training process anew. [5].

During the fine-tuning process, the general structure of the pre-trained model is usually preserved. The concept revolves around utilizing the beneficial features and representations that the model acquired from its original extensive training dataset and adjusting them to address a more specialized task. [5].

What Makes Fine-Tuning a Valuable Technique?

Fine-tuning presents multiple unique benefits that have propelled it to become a widely utilized method in the realm of machine learning:

Efficiency

Creating a deep learning model from scratch can be a time-consuming and resource-intensive process. Contrastingly, fine-tuning enables us to enhance a pre-trained

model, leading to considerable time and resource savings in achieving favorable outcomes. Leveraging a model that has already acquired pertinent features allows us to bypass initial training phases and concentrate on tailoring the model to the particular task at hand. [5].

Improved Performance

Pre-trained models have been previously trained on substantial amounts of data for general purposes. Consequently, these models have acquired valuable features and patterns that can prove useful for similar tasks. By fine-tuning a pre-trained model, we can take advantage of this abundance of knowledge and representations, ultimately resulting in enhanced performance for our targeted task. [5].

Data Efficiency

In numerous practical situations, acquiring labeled data for a particular task can be arduous and time-intensive. Fine-tuning provides a viable solution by enabling efficient model training, even with minimal labeled data. Utilizing a pre-trained model as a starting point and modifying it to suit our specific task can maximize the usefulness of the available labeled data and lead to beneficial outcomes with minimal effort. [5].

2.1.4 Generative AI

Generative AI empowers users to swiftly produce new content from a wide range of inputs. These models can handle both inputs and outputs such as text, images, sounds, animations, 3D models, and other types of data. [6].

How Does Generative AI Work?

Generative AI models utilize neural networks to discern patterns and structures in existing data, enabling them to create new and original content. [6].

A major breakthrough in generative AI models is their ability to utilize various learning approaches, such as unsupervised or semi-supervised learning, for training. This advancement allows organizations to efficiently and rapidly use large amounts of unlabeled data to develop foundation models. As the name implies, foundation models serve as a base for AI systems capable of performing multiple tasks. [6].

Examples of foundation models include GPT-3 and Stable Diffusion, both of which harness the power of language. GPT-3 powers applications like ChatGPT, enabling users to generate essays from short text prompts. Meanwhile, Stable Diffusion allows users to create photorealistic images from textual descriptions. [6].

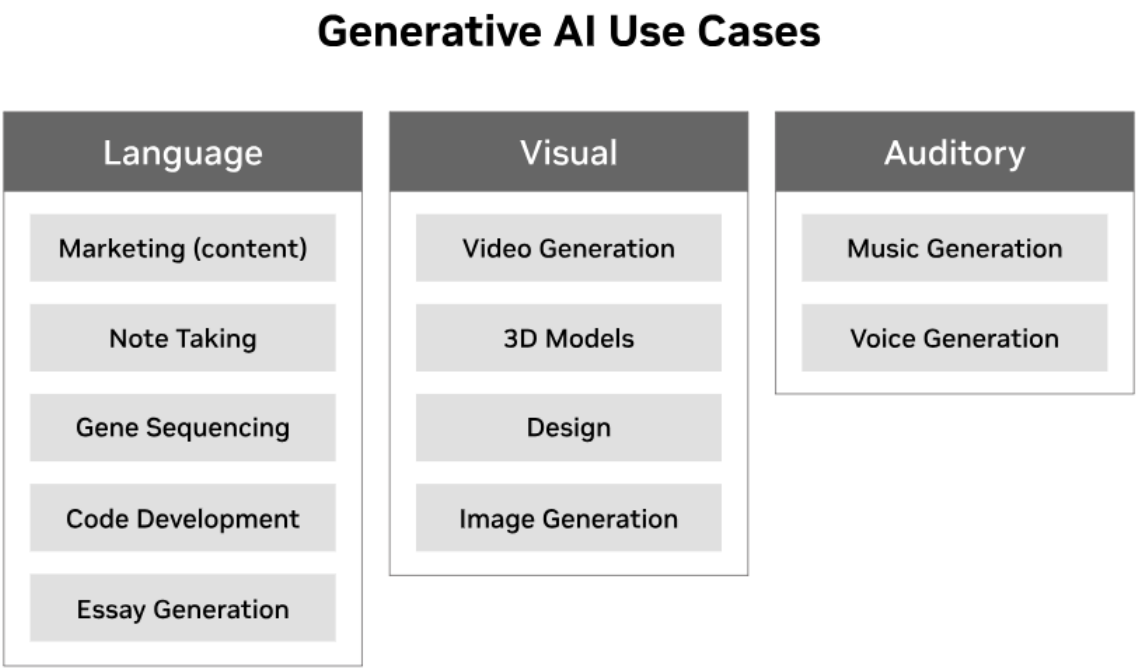

Figure 2.2: The diagram shows possible generative AI use cases within each category [6]

2.1.5 NLP

Natural language processing (NLP) merges computational linguistics, which involves rule-based modeling of human language, with statistical and machine learning models. This combination enables computers and digital devices to recognize, understand, and generate text and speech. [7]. A branch of artificial intelligence (AI), natural language processing (NLP) powers applications and devices that perform a variety of tasks, such as translating text between languages, responding to typed or spoken commands, recognizing or authenticating users by voice, summarizing large volumes of text, assessing the intent or sentiment of text or speech, and generating text, graphics, or other content on demand, often in real-time. Many people encounter NLP through voice-operated GPS systems, digital assistants, speech-to-text software, customer service chatbots, and other consumer tools. Beyond consumer use, NLP is increasingly integral to enterprise solutions, streamlining and automating business operations, boosting employee productivity, and simplifying critical business processes.[7].

2.2 Large Language Models

Large language models (LLMs) are extensive deep learning models pre-trained on vast datasets. At their core, LLMs use transformer architectures, comprising neural networks with an encoder and a decoder, both equipped with self-attention mechanisms. The encoder and decoder work together to extract meaning from text sequences and comprehend the relationships between words and phrases within the text. [8].

Transformer LLMs can undergo unsupervised training, though a more accurate term is self-learning. Through this process, transformers learn to comprehend basic grammar, languages, and knowledge. [8].

In contrast to earlier recurrent neural networks (RNN), which handle inputs sequentially, transformers process entire sequences simultaneously. This parallel processing capability enables data scientists to utilize GPUs for training transformer-based LLMs,

leading to significantly reduced training times. [8].

The Transformer neural network architecture enables the deployment of exceptionally large models, often equipped with hundreds of billions of parameters. These expansive models have the capacity to process vast volumes of data, sourced primarily from the internet, including repositories like the Common Crawl, housing over 50 billion web pages, and Wikipedia, with approximately 57 million pages.

Large language models exhibit remarkable versatility. A single model can seamlessly transition between various tasks, including answering questions, summarizing documents, translating languages, and completing sentences. As such, LLMs possess the potential to revolutionize content creation and reshape user interactions with search engines and virtual assistants.[8].

Though not flawless, Large Language Models (LLMs) exhibit a remarkable capability to generate predictions from a relatively limited number of prompts or inputs. Leveraging this capacity, LLMs are extensively utilized in generative AI applications, wherein they generate content based on input prompts expressed in human language.[8].

LLMs are colossal, boasting an immense number of parameters that can reach into the billions, rendering them incredibly versatile. Here are a few examples:

OpenAI's GPT-3 model, for instance, comprises a staggering 175 billion parameters. Its derivative, ChatGPT, showcases the capability to discern patterns within data and produce coherent and intelligible output. [8]. AI21 Labs' Jurassic-1 model boasts an impressive 178 billion parameters and a token vocabulary comprising 250,000 word parts, facilitating conversational capabilities akin to human interaction. Similarly, Cohere's Command model exhibits comparable functionalities and extends its proficiency to over 100 different languages.

Meanwhile, LightOn's Paradigm introduces foundation models with purported capa-

bilities surpassing those of GPT-3. Notably, all these Large Language Models (LLMs) provide developers with APIs, empowering them to craft innovative generative AI applications tailored to specific needs and requirements. [8].

2.2.1 Transformers

A transformer model is a type of neural network designed to grasp context and derive meaning from sequential data, such as the words within a sentence, by scrutinizing the relationships between them.[9]. A transformer model is a neural network that learns context and thus meaning by tracking relationships in sequential data like the words in this sentence [9].

Transformer models utilize a dynamic array of mathematical techniques, known as attention or self-attention mechanisms, to discern intricate connections between even distant data elements within a sequence. This enables the model to capture subtle dependencies and influences among these elements effectively. [9].

Originally detailed in a 2017 paper authored by researchers at Google, transformers represent one of the most recent and formidable classes of models developed to date. Their emergence has catalyzed a surge of progress in the field of machine learning, prompting some to refer to this momentum as the era of "transformer AI." [9].

2.2.2 Models

Within the expansive domain of language technology, a multitude of large language models flourish, each tailored to specific training data and objectives. Some are trained on broad datasets, while others specialize in codebase data, and still others are honed for tasks such as instruction or text completion. In this report, I present an overview of the premier models available at the time of writing.

DeepSeek: DeepSeek Coder comprises a suite of code language models, each meticulously trained from the ground up on a dataset containing 2 trillion tokens. This corpus

is composed of 87% code and 13% natural language, encompassing both english and Chinese. We offer a range of model sizes, from 1 billion to 33 billion versions, to accommodate diverse needs.

each model undergoes pre-training on a project-level code corpus, utilizing a window size of 16K and an additional fill-in-the-blank task. This approach facilitates project-level code completion and infilling. In terms of coding proficiency, DeepSeek Coder demonstrates state-of-the-art performance across multiple programming languages and various benchmarks among open-source code models. [10].
BERT: Introduced by Google in 2018, BeRT represents a family of Large Language Models (LLMs). Based on the transformer architecture, BERT excels in converting sequences of data into other sequences of data. Its architecture consists of a stack of transformer encoders and boasts a parameter count of 342 million.

Initially pre-trained on a substantial corpus of data, BERT is subsequently fine-tuned to excel in various tasks, including natural language inference and sentence text similarity. Notably, BERT played a pivotal role in enhancing query understanding within the 2019 iteration of Google search. [11].

Cohere: Cohere stands as an enterprise AI platform offering a range of Large Language Models (LLMs), including Command, Rerank, and embed. Distinguished by their adaptability, these LLMs can be tailored and fine-tuned to cater to the specific needs of individual companies. Notably, Cohere's LLMs are not confined to a single cloud provider, unlike OpenAI, which is closely associated with Microsoft Azure. Noteworthy is the fact that Cohere was founded by one of the authors of "Attention Is All You Need," adding to its credibility in the field.
GPT-4: GPT-4, the latest installment in OpenAI's GPT series, made its debut in 2023. Similar to its predecessors, it operates on a transformer-based architecture. However, unlike earlier versions, OpenAI has not disclosed the model's parameter count to the

public. Speculations suggest that it boasts over 170 trillion parameters.

Distinguished as a multimodal model, GPT-4 possesses the ability to process and generate not only language but also images, expanding its capabilities beyond mere linguistic tasks. A notable addition is the introduction of a system message feature, empowering users to specify the desired tone of voice and task. [11].

Upon its release, GPT-4 showcased human-level proficiency across various academic exams, sparking speculation regarding its proximity to achieving artificial general intelligence (AGI), a milestone indicating intelligence comparable to or surpassing that of humans. GPT-4 currently drives Microsoft Bing search functionality, features in ChatGPT Plus, and is slated for integration into Microsoft Office products in the future. [11].
Llama: Large Language Model Meta AI (Llama), Meta's proprietary LLM, made its debut in 2023, boasting an expansive parameter count of up to 65 billion in its largest version. Initially accessible only to approved researchers and developers, Llama has since transitioned to an open-source model.

One of Llama's notable features is its availability in various sizes, accommodating different computing power requirements. This versatility enables users to employ, test, and experiment with Llama across a spectrum of applications with varying computational resources. [11].

2.3 Conclusion

In this chapter, we delved into the existing technology of Artificial Intelligence and explored various foundations that will underpin the completion of our project. However, AI encompasses more branches beyond the scope of this project, such as computer vision, predictions, classification and clustering, as well as reinforcement learning, which are fascinating areas worth exploring.

Chapter 3

Solution Design And Project Execution

This chapter delves into the practical implementation of our project, focusing on the solution design, the development environment and execution process. It begins with the problem anaslysis then an overview of the software and tools essential to our solution, including Python, Visual Studio Code, OpenAI, Hugging Face, and GitHub. These tools are carefully selected to streamline development workflows, enhance collaboration, and ensure project success. The chapter then progresses to discuss key phases of project execution, such as data collection, processing, selection of algorithms and models, fine-tuning, validation, evaluation, model deployment. Each phase is meticulously planned and executed to achieve project objectives effectively. The chapter underscores the importance of leveraging cutting-edge technologies and methodologies to optimize project execution and drive innovation. Through strategic planning and utilization of resources, we aim to navigate complexities and challenges, ultimately realizing the full potential of our project.

3.1 Problem analysis and solution design

3.1.1 Objective Identification

To address the challenge of repetitive boilerplate code writing in Django development at Infinity Management, we are exploring the implementation of an innovative AI solution. The primary objective of this project is to develop an AI-powered application that leverages Large Language Models (LLMs) and natural language processing (NLP) algorithms to automate code generation. Developers will specify their requirements through JSON files (JavaScript Object Notation), which the AI application will interpret to generate the necessary code structures. This solution aims to streamline the development process by reducing the manual effort required for setting up models, views, and serializers in Django projects. By ensuring consistency and accuracy in code generation, the AI application will free developers to focus on more complex and innovative tasks. Ultimately, this AI-driven approach will revolutionize Django development workflows, enhance productivity, and demonstrate Infinity Management's commitment to leveraging cutting-edge technologies for continuous improvement and innovation.

3.1.2 Presenting The Data

We have repositories containing Infinity Management's previous projects. These repositories include all project files and the code developed in past projects. Within these repositories, you will find configuration files as well as application files, which encompass modules, serializers, and views. the barplot in figure 3.1 below shows the distribution of the files that we have.

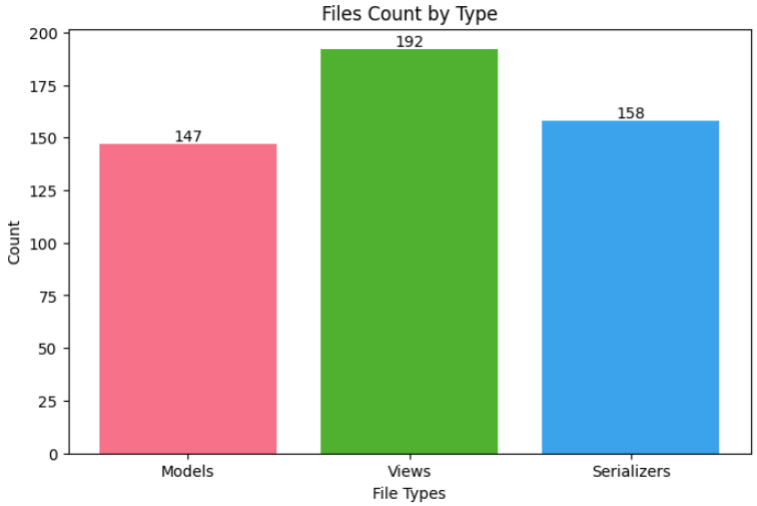

Figure 3.1: Bar Plot of the Types Of files

As you can see we have a total of 503 file they are distributed like this:

Models : 147 Views : 198 Serializers: 158

The models : these files contain the class that represent the entities of the webapplication. These models contains the attributes, the methods and the relationship with other models.

```python
class Activity(Timestampable, Authorable, Structurable):
    """Activity"""

    label_fr = models.CharField(max_length=150, blank=True)
    label_ar = models.CharField(max_length=150, blank=True)
    activity_type = models.IntegerField(blank=True, null=True)
    subject_skill = models.IntegerField(blank=True, null=True)
    animator_type = models.IntegerField(blank=True, null=True)

    # Related field
    partner = models.ForeignKey(Partner, null=True, blank=True, on_delete=models.SET_NULL)
    partner_service = models.ForeignKey(ServiceDetail, null=True, blank=True, on_delete=models.SET_NULL)

    # Related fields
    trainers = models.ManyToManyField(
        User,
        blank=True,
        related_name="trainers_set",
        related_query_name="user",
    )

    description_fr = models.TextField(null=True, blank=True)
    description_ar = models.TextField(null=True, blank=True)
```

Figure 3.2: Example of Model File

In this example represented in figure 3.2, our model name is Activity it have 7 attributes which are "label_fr","label_ar","activity_type", "subject_skill", "animator_type", "description_fr" and "description_ar", the model have 2 foreign keys which are "partner" and "partner_service".

The serializers: these files are used to convert complex data types, such as querysets and model instances, into native Python data types that can then be easily rendered into JSON, XML, or other content types. Serializers also handle deserialization, allowing parsed data to be converted back into complex types after validation.

```python
class SupportAssessmentToolSerializer(serializers.ModelSerializer):
    def __new__(cls, *args, **kwargs):
        if kwargs.get('many', False) is True:
            context = kwargs.get('context', {})
            context.update({'has_many': True})
            kwargs.update({'context': context})

        return super().__new__(cls, *args, **kwargs)

    support_files_details = serializers.SerializerMethodField()
    test_label_details = serializers.SerializerMethodField()

    class Meta: ···

    def get_test_label_details(self, obj): ···

    def get_support_files_details(self, obj): ···

    def create(self, validated_data): ···

    def update(self, instance, validated_data): ···
```

Figure 3.3: Example of Serializer File

In this example of figure 3.3, the serializer is called SupportAssesmentToolSerializer it have 4 functions which are "get_test_label_details", "get_support_files_details", "create" and "update".

The views: these files contain the logic that determines what data is presented to the user and how it is presented. Views handle requests and return responses. A folder called config were we find the configuration of the webapplication the things related to security and the settings of the application.

```python
class SupportAssessmentToolRetrieveUpdateDestroyAPIView(generics.RetrieveUpdateDestroyAPIView):
    permission_classes = [IsAuthenticated]
    serializer_class = SupportAssessmentToolSerializer
    queryset = SupportAssessementTool.objects.all()

    def patch(self, request, *args, **kwargs):...

class SupportAssessmentToolDeactivateAPIView(generics.DestroyAPIView):
    permission_classes = (IsAuthenticated,)
    serializer_class = SupportAssessmentToolSerializer
    queryset = SupportAssessementTool.objects.filter(
        deleted_at__isnull=True,
    )

    def perform_destroy(self, instance):...


class SupportAssessmentToolActivateAPIView(generics.DestroyAPIView):
    permission_classes = (IsAuthenticated,)
    serializer_class = SupportAssessmentToolSerializer
    queryset = SupportAssessementTool.objects.filter(
        deleted_at__isnull=False,
    )

    def perform_destroy(self, instance):...
```

Figure 3.4: Example of Views File

In this example of figure 3.4, the views file is named "support_assesment_views.py" and we can find 3 views:

- SupportAssesmentToolRetrieveUpdateDestroyAPIView and it have a function called "patch",
- SupportAssesmentToolDeactivateAPIView this one have a "perform_destroy" function ,
- SupportAssessmentToolActivateAPIView that have "perform_destroy" function.

3.1.3 Solution Design

Drawing from existing data and leveraging the latest advancements in AI tools, our approach entails automating the code generation process for project applications. Rather than relying solely on manual coding, developers now undertake solution modeling,

paving the way for AI-driven code generation. To streamline this process, we've structured the solution into two distinct steps. Initially, developers input the model into an Excel file, which subsequently undergoes transformation into a JSON format. The subsequent step involves an AI model comprehending the JSON data and autonomously generating the project's codebase. This methodology necessitates two integral components: a script facilitating the conversion of Excel data into JSON format, and the AI model itself, tailored to comprehend and proficiently generate the requisite data.

The project's breakdown follows a systematic approach:

Data Preprocessing: This stage involves meticulous data cleaning and preparation, fine-tuning the model for the specific task at hand - converting JSON data into a functional codebase.

Model Selection: Here, a meticulous selection process is employed to choose a pre-trained model capable of fulfilling the defined task effectively.

Model Fine-tuning: The selected model undergoes a fine-tuning process, where appropriate techniques are employed to optimize its performance in generating code from JSON input data.

Model Testing: Rigorous testing ensues to validate the model's outputs. Any discrepancies prompt a revisitation of steps 1 and 3, ensuring optimal performance.

With the model established, attention shifts towards crafting an Excel file template, primed for conversion into JSON format. Simultaneously, a script is developed to execute this conversion seamlessly.

Finally, the project culminates in a streamlined pipeline, commencing with Excel-to-JSON conversion and concluding with the automatic generation of the project repository. This methodical approach ensures efficiency and accuracy throughout the

development lifecycle.

3.2 Development environment

After describing the meticulous analysis and solution design phase, our gaze now shifts towards the Project execution. This pivotal stage marks the transition from conceptualization to action, where the groundwork laid in problem identification and solution formulation will be translated into tangible outcomes. With a strategic roadmap in hand and a clear vision guiding our endeavors, we are poised to navigate the complexities of execution with precision and professionalism. As we embark on this phase, we recognize the significance of disciplined project management, effective collaboration, and agile adaptation in achieving our objectives. With a steadfast commitment to excellence and a spirit of innovation driving our efforts, we approach Project execution with confidence, ready to overcome challenges and realize the full potential of our endeavor.

In this section, we will mention the software and tools of our solution.

3.2.1 Python

Python, a high-level and interpreted programming language, is celebrated for its simplicity, versatility, and user-friendly nature. Originally conceived by Guido van Rossum in the late 1980s, Python has evolved into one of the most widely used languages globally. Its applications span a multitude of domains, including web development and scientific computing, making it an indispensable tool in numerous industries. [12].

Figure 3.5: Python [12]

3.2.2 Visual Studio Code

Visual Studio Code (VS Code) is a free, open-source code editor from Microsoft, celebrated for its versatility and powerful features. Available on Windows, macOS, and Linux, it serves developers across all major operating systems. A key strength of VS Code is its extensibility, allowing users to enhance its functionality with a wide array of extensions from the VS Code Marketplace. It includes IntelliSense for intelligent code completions, helping developers code faster and more accurately. The integrated terminal allows for running command-line tools and scripts directly within the editor. Built-in Git support simplifies version control operations like commits and branching. Additionally, VS Code provides robust debugging tools, enabling users to set breakpoints, inspect variables, and control code execution. One of its standout features is remote development support, which allows developers to access, edit, and debug code on remote machines, virtual machines, or containers effortlessly. Its lightweight design ensures quick performance, making it ideal for a wide range of development tasks. Highly customizable with various themes, keyboard shortcuts, and layout options, VS Code is supported by a large, active community and extensive documentation, establishing it as a top choice for modern development workflows [13].

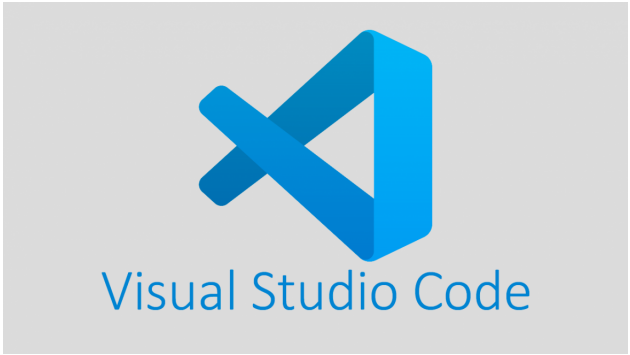

Figure 3.6: Visual Studio Code [13]

3.2.3 OpenAI

OpenAI is a research organization dedicated to advancing artificial intelligence (AI) in a safe and beneficial manner. Founded in December 2015, OpenAI's mission is to ensure that AI technology benefits all of humanity. Led by a team of researchers, engineers, and ethicists, OpenAI conducts cutting-edge research and develops innovative AI technologies [14].

Services: OpenAI offers a range of services and tools aimed at democratizing AI and making it accessible to developers and organizations worldwide. Some of their key services include:

- OpenAI API: The OpenAI API provides access to state-of-the-art AI models that can generate text, answer questions, translate languages, and more. Developers can integrate these models into their applications to add advanced AI capabilities [14].

- OpenAI Gym: OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. It provides a suite of environments for training and testing AI agents, making it easier for researchers and developers to work on reinforcement learning projects [14].

- OpenAI Scholars Program: The OpenAI Scholars Program is a research fellowship aimed at supporting talented individuals from underrepresented backgrounds in AI. Scholars receive mentorship and resources to pursue independent research projects under the guidance of OpenAI researchers [14].

[14].

3.2.4 Hugging Face

Hugging Face is a company recognized for its contributions to natural language processing (NLP). They've developed tools like Transformers, which simplify the use of advanced NLP models such as BERT and GPT. Hugging Face also offers pre-trained models for tasks like text generation and sentiment analysis. Their platform encourages collaboration among NLP developers and researchers by facilitating the sharing of models and datasets [15].

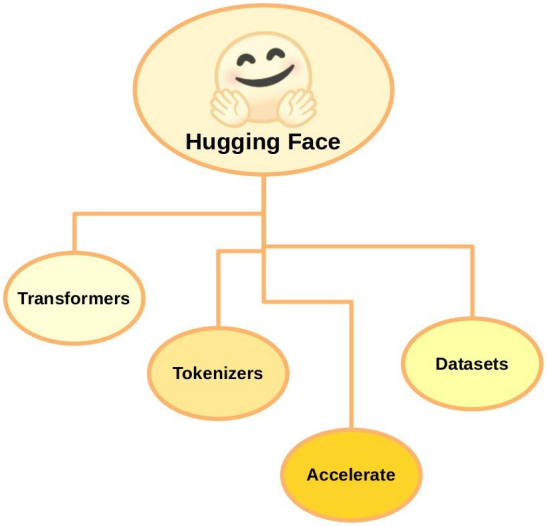

Figure 3.7: hugging face [15]

3.2.5 Git Hub

GitHub is a collaborative platform for developers, built around Git version control. It facilitates decentralized development by allowing developers to work on their copies of projects and merge changes seamlessly. GitHub offers features like repositories for organizing code, issue tracking for bug reports and feature requests, and pull requests for proposing and reviewing changes. Code review tools ensure quality control before merging changes into the main project. GitHub's flexibility and structured collaboration enable teams to work efficiently on coding projects, ensuring code quality and

fostering innovation in software development [16].

3.3 Implementation

3.3.1 Data Collection

During the data collection phase of the project, we systematically gathered the source code of Infinity Management from the GitLab repository using the "git clone" command. This command facilitated the creation of a local copy of the entire source code repository, ensuring that I had access to the latest version of the codebase and its entire history. By cloning the repository, I obtained a comprehensive dataset comprising all source code files, directories, and version control information. This approach provided me with a solid foundation for analyzing the structure, functionality, and evolution of the Infinity Management software system. Furthermore, leveraging GitLab as the source code repository allowed for efficient management and tracking of changes made to the codebase by the development team. Overall, the use of "git clone" to collect the source code data from the GitLab repository proved to be a reliable and indispensable step in preparing for the subsequent phases of the project, such as analysis, testing, and development.

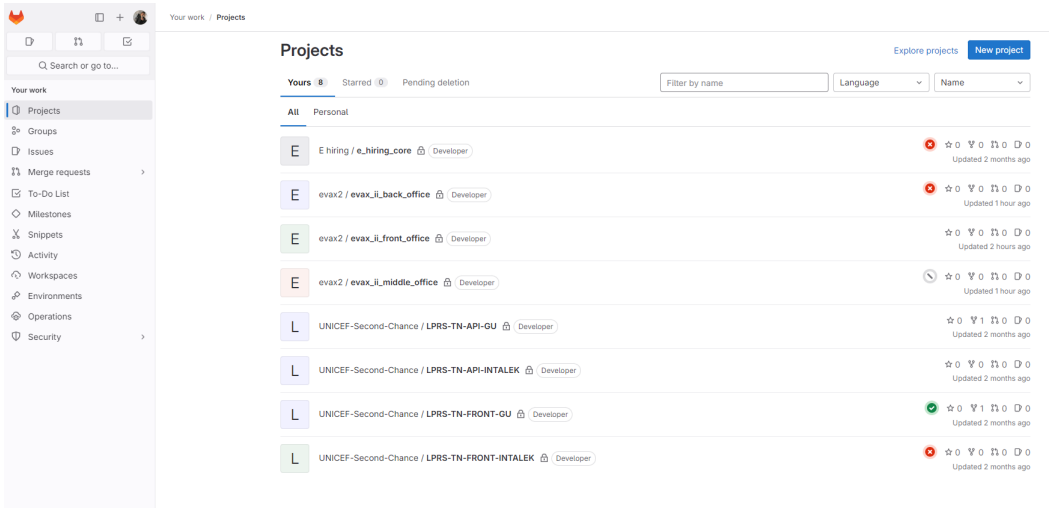

Figure 3.8: Repo Gitlab

3.3.2 Data Processing

In the initial step of data processing, I focused on filtering and organizing the collected data from the Infinity Management GitLab repository (as shown in figure 3.8). Primarily, I narrowed down the dataset to conserve only the Python files, which constitute the core source code of the project. This selective approach ensured that irrelevant files, such as configuration files or documentation, were excluded from further analysis, streamlining the dataset for subsequent processing steps. By retaining only Python files, I aimed to prioritize the extraction of actionable insights and patterns relevant to the project objectives.

In the second step of data processing, I employed a technique known as reverse prompt engineering to generate prompts corresponding to the source code obtained from the Infinity Management GitLab repository. This process involved feeding the code snippets into a language model, such as GPT (Generative Pre-trained Transformer), through a chat interface. By interacting with the model and providing it with the source code, I elicited prompts that would generate similar or related code segments. This approach allowed me to gain insights into the model's understanding of the code and its ability to generate syntactically and semantically coherent code snippets based on the provided input.

I generated prompts for each part of the code differentially for the views models and serializers. The prompts are like as shown in figures bellow.

```
# file_name = "absence_sheet.py"
prompt = f"""I will give you a python code and from that code I want you to output a json file using the template provided below if
there are multiple models in the file generate a list that contains each model :
template:
{{
    "file_name":"hello.py",
    "model_name":"Hello",
    "type":"model",
    "attributes":["name_fr","name_ar","phone","email",..],
    "functions":[],
    "m2m_attributes":[],
    "ForeignKey_attributes":[]
}}
file name : {file_name}
code : """
```

Figure 3.9: Prompt For the Models

This is the prompt I entered to GPT so I can get my json files for the model to train my AI model later. In other words I gave the template of the json file that I want to be outputed. In fact, I wanted a json file that have the file name, the model name, the type in this case it's a model the attributes of the model the methods of the model the many to many fields and the foreign keys.

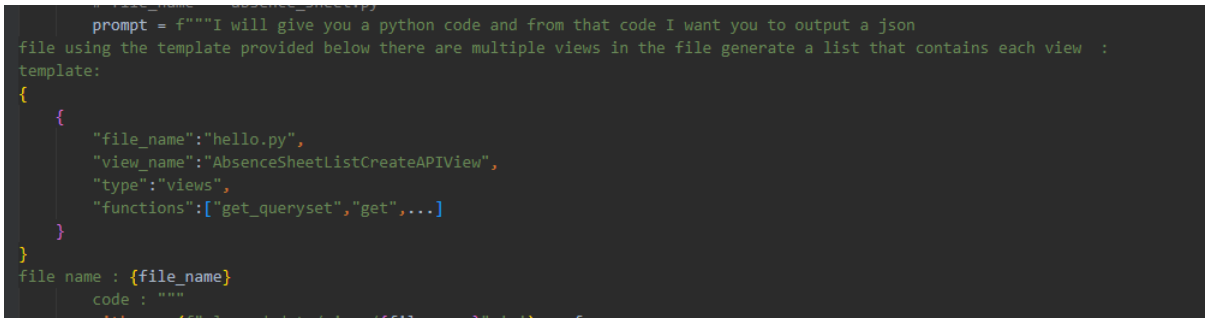

Figure 3.10: Prompt For the Views

The prompt in figure 3.10 is the one I entered to GPT so I can get my json files for the views to train my AI model later the json outputs contains the file name the view_name because one file of the views contains many views and finally the methods of that view.

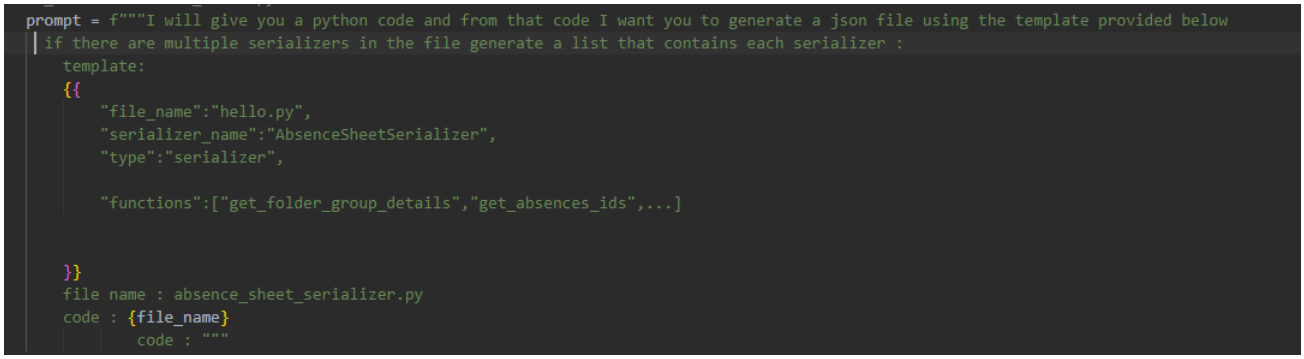

Figure 3.11: Prompt For the Serializers

The prompt in figure 3.11 is the one I entered to GPT so I can get my json files for the serializer to train my AI model later for this type of files I needed the fields of file

name, serializer name, type which will be serializer and functions of the serializer

Now after generating the json files with GPT the figures 3.12, 3.13 and 3.14 bellow are the responses that I got from GPT. I'm going to use the example of files that I showed in the previous section.

Figure 3.12: Model Output By GPT

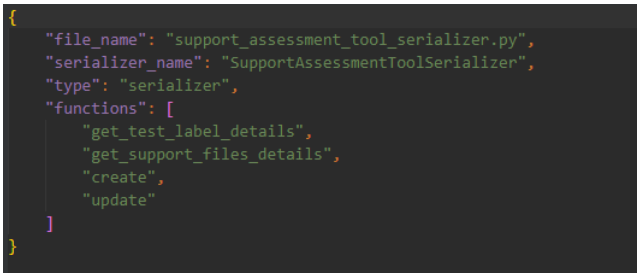

Figure 3.13: Serializer Output By GPT

```
[
    {
        "file_name": "support_assessment_tool_view.py",
        "view_name": "SupportAssessmentToolListCreateAPIView",
        "type": "views",
        "functions": [
            "get_queryset",
            "get",
            "perform_bulk_destroy",
            "create"
        ]
    },
    {
        "file_name": "support_assessment_tool_view.py",
        "view_name": "SupportAssessmentToolRetrieveUpdateDestroyAPIView",
        "type": "views",
        "functions": [
            "patch"
        ]
    },
    {
        "file_name": "support_assessment_tool_view.py",
        "view_name": "SupportAssessmentToolDeactivateAPIView",
        "type": "views",
        "functions": [
            "perform_destroy"
        ]
    },
    {
        "file_name": "support_assessment_tool_view.py",
        "view_name": "SupportAssessmentToolActivateAPIView",
        "type": "views",
        "functions": [
            "perform_destroy"
        ]
```


Figure 3.14: Views Output By GPT

3.3.3 Selection of algorithms and models

After preprocessing the Data It's time to choose an AI Model for the project. In the selection of algorithms and models for the project, I conducted extensive research into various approaches to code generation. I discovered many models such as LLAMA 2 which is an LLM developed By meta, I tested many version of that model which are the 7b and the 33b, I also tried Qwen. The Problem with those models that they are pretrained on general contexts and the more parameters mean that the model will be slower in generating the code with the given resources that we have at infinity management. I kept searching for the ideal model until I opted for DeepSeek, a specialized Large Language Model (LLM) tailored for code generation tasks. DeepSeek stood out among the models considered due to its robust architecture and its focus on

code-specific tasks, making it particularly well-suited for the objectives of the project.

One key factor that influenced the decision to choose DeepSeek was its training data composition. Approximately 80% of DeepSeek's training data consisted of code samples, indicating a strong emphasis on code-related tasks during model training. This composition suggested that DeepSeek had been extensively trained on a diverse range of code snippets, enabling it to better understand the nuances of programming languages and syntax.

Moreover, DeepSeek's architecture was specifically designed to handle code generation tasks efficiently, leveraging techniques such as self-attention mechanisms and transformer architectures to capture long-range dependencies and context within code sequences. This design feature gave DeepSeek a competitive edge in accurately generating code snippets that adhere to syntactic and semantic constraints.

By selecting DeepSeek as the primary model for code generation, I aimed to leverage its advanced capabilities to enhance the development process within the Infinity Management project. The model's proficiency in generating code snippets aligned closely with the project's objectives, facilitating tasks such as code augmentation, refactoring, and even automatic bug fixing. Overall, the decision to adopt DeepSeek represented a strategic choice to harness cutting-edge AI technologies tailored specifically for code-related tasks, thereby advancing the efficiency and effectiveness of software development at Infinity Management.

3.3.4 Fine-tuning

After the model selection, the project proceeded to the fine-tuning phase. there are so many techniques of finetuning a pretrained model. We could retrain the whole parameters again with the new data, but the problem with this technique is that the model could lose the information that was initially pretrained, this technique also need a lot of data and resources to retrain the whole model again. This is where a family

called PEFT (Parameter-Efficient Fine-Tuning) comes to help.

Parameter-efficient fine-tuning (PEFT) methods focus on fine-tuning only a small subset of additional model parameters while keeping the majority of the pretrained Large Language Models (LLMs) frozen. This strategy significantly reduces computational and storage costs. Additionally, PEFT helps mitigate the problem of catastrophic forgetting, which occurs during full fine-tuning of LLMs.

Studies have indicated that PEFT approaches outperform traditional fine-tuning methods, particularly in scenarios with limited training data, and exhibit superior generalization capabilities across different domains. Furthermore, PEFT techniques are versatile and can be applied across various modalities, including image classification and applications like the Stable Diffusion Dreambooth.[17].

PEFT methods also enhance model portability by enabling users to fine-tune models to generate compact checkpoints, mere megabytes in size, in contrast to the larger checkpoints generated through full fine-tuning. For instance, a model like bigscience/mt0-xxl, which occupies 40GB of storage, would require 40GB checkpoints for each downstream dataset under full fine-tuning. However, employing PEFT methods results in just a few megabytes for each downstream dataset, while still achieving comparable performance to full fine-tuning.

These lightweight trained weights obtained from PEFT approaches are seamlessly integrated with the pretrained LLM, allowing users to leverage the same model across multiple tasks without necessitating the replacement of the entire model. This flexibility enables efficient utilization of resources while maintaining high-performance outcomes across diverse applications. [17].

In essence, PEFT approaches empower users to achieve performance levels comparable to full fine-tuning, all while leveraging only a small subset of trainable parameters. [17].

One of the prominent PEFT methods, LoRA, played a pivotal role in fine-tuning the model for the Infinity Management project. LoRA, which stands for Low Rank Adapter, is a specialized framework designed to adapt large language models to specific tasks or domains. Its implementation was instrumental in tailoring DeepSeek to meet the unique requirements of the Infinity Management project.

During the fine-tuning process with LoRA, DeepSeek was exposed to project-specific data, including code samples, documentation, and contextual information. This allowed the model to learn domain-specific patterns and conventions, enhancing its ability to generate relevant and accurate code snippets for Infinity Management's requirements.

LoRA's low-rank adapter approach enabled efficient parameter adaptation within DeepSeek, preserving the model's generalization capabilities while fine-tuning it for the project's specific needs. By incorporating LoRA into the fine-tuning process. I aimed to strike a balance between model specialization and generalization, ensuring that DeepSeek could effectively address the challenges posed by the Infinity Management project while retaining its broader applicability. Now let's dive in with more details about the finetuning step. After getting the json files from GPT it's time to put everything in one file for each json the code correspondent to it

```python
for root,dirs, files in os.walk("cleaned data",topdown=False):
    for name in files:
        if(f".json" in os.path.join(root,name)):
            json_file = os.path.join(root,name)
            python_file = json_file.replace(".json",".py")
            with open(json_file,"r") as js:
                instruction = js.read()
            with open(python_file,"r") as py:
                output = py.read()
            if len(instruction)>20:
                data = {
                    "instruction":instruction,
                    "output":output
                }
                training_data.append(data)
file_path = "training_json_data.json"

# Open the file in write mode
with open(file_path, "w") as json_file:
    # Write the list to the file in JSON format
    json.dump(training_data, json_file, indent=4)
```

Figure 3.15: Training Data Process

In the code of the figure 3.15 above, what I did is reading each file in the directory get the json and the code file. I pass the json format in an "instruction" value and the code that will be generated in the "output" to finally get the final data to train our model in this format. The figure 3.16 below shows the data structure.

Figure 3.16: Training Data

Now After Preparing the Training data it's time to prepare the finetuning script. So first of all we need to set up the tokenizer which is the same tokenizer of the base model.

```
tokenizer = transformers.AutoTokenizer.from_pretrained(
    model_args.model_name_or_path,
    model_max_length=training_args.model_max_length,
    padding_side="right",
    use_fast=True,
    trust_remote_code=True
)
```

Figure 3.17: Loading the Tokenizer

after loading the tokenizer, I imported the base model which will be passed as an argument in the command line when we call the script to be executed (Figure 3.18).

```
model = transformers.AutoModelForCausalLM.from_pretrained(
    model_args.model_name_or_path,
    device_map='auto'
)
```

Figure 3.18: Loading the base model

After importing the base model, I set up the LoraConfig which is the configuration of the new layers of LoRA Adapter that will be trained and responsible for the model finetuning, as shown in figure 3.19.

```
config = LoraConfig(
r=16, #attention heads
lora_alpha=32, #alpha scaling

lora_dropout=0.05,
bias="none",
task_type="CAUSAL_LM" # set this for CLM or Seq2Seq
)
```

Figure 3.19: LoRA Config

After setting up the LoRA config and loaded the model, I merged the model to have the final architecture and print the trainable parameters with this line of code (Figure 3.20).

```
model = get_peft_model(model, config)
print("==========================")
print_trainable_parameters(model)
print("==================================")
```

Figure 3.20: Merging The Model

Then I loaded the training dataset, trained and saved the model (Figures 3.21, 3.22).

```
raw_train_datasets = load_dataset(
    'json',
    data_files=data_args.data_path,
    split="train",
    cache_dir=training_args.cache_dir
)
if training_args.local_rank > 0:
    torch.distributed.barrier()

train_dataset = raw_train_datasets.map(
    train_tokenize_function,
    batched=True,
    batch_size=3000,
    num_proc=32,
    remove_columns=raw_train_datasets.column_names,
    load_from_cache_file=True, # not args.overwrite_cache
    desc="Running Encoding",
    fn_kwargs={ "tokenizer": tokenizer }
)
```

Figure 3.21: Loading the training dataset Script

```
trainer = Trainer(model=model, tokenizer=tokenizer, args=training_args, **data_module)

trainer.train()
trainer.save_state()
safe_save_model_for_hf_trainer(trainer=trainer, output_dir=training_args.output_dir)
```

Figure 3.22: Training Script

After setting up the script for finetuning the model, I called it via the command line and passed the training arguments as it show in the figure 3.23 below:

```
.venv(base) infinitymgt@infinitymgts-Mac-Studio % python3
    finetune_deepseekcoder.py --model_name_or_path "deepseek-ai/deepseek-
    coder-6.7b-instruct" \
    --data_path "training_json_data.json" \
    --output_dir "output finetuning" \
    --num_train_epochs 50 \
    --evaluation_strategy "no" \
    --save_strategy "steps" \
    --save_steps 100 \
    --save_total_limit 100 \
    --learning_rate 2e-5 \
    --warmup_steps 10 \
    --logging_steps 1 \
    --lr_scheduler_type "cosine" \
    --report_to "tensorboard"
```

Figure 3.23: Training Command Line

In this command line I specify the model name which is deepseek-coder-6.7b-instruct the training data path, the output directory where the model will be saved, the learning rate, etc.

After executing the command line we the model will start the training and we can see

the trainable parameters. The total parameters are 6.7 billion parameters but with the LoRA method we are able to train only the 0.1% of those parameters and that will save time and resources.

```
========================
trainable params: 8388608 || all params: 6748901376 || trainable%: 0.12429590436498326
================================
Load model from deepseek-ai/deepseek-coder-6.7b-instruct over.
```

Figure 3.24: Number of Trainable Parameters

In the next part we will see the results and the evaluation of the finetuned model

3.3.5 Validation and evaluation

After finishing the finetuning it's time to evaluate and try the model and see the results. First of all let's see the loss function evolution during the epochs in figure 3.25 bellow:

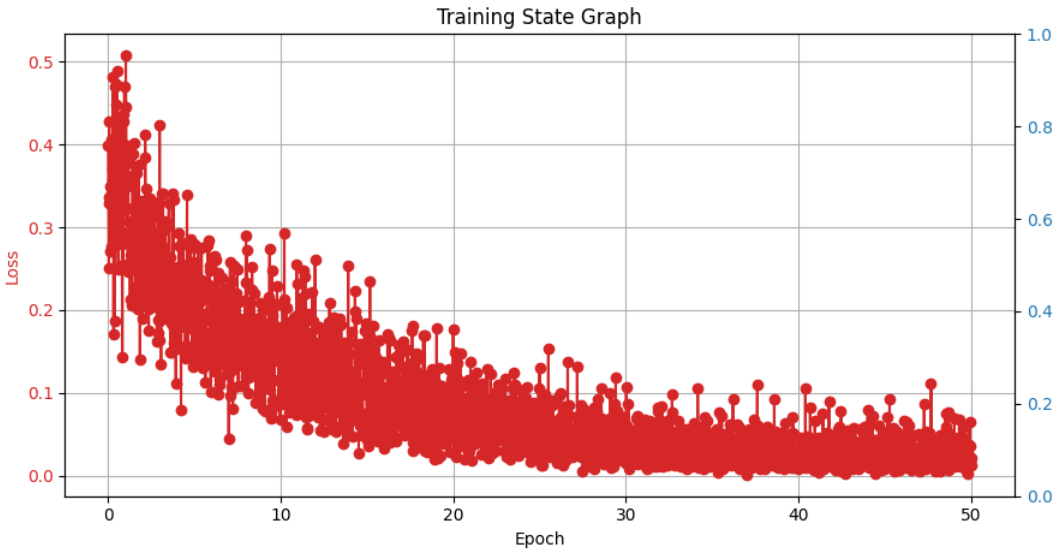

Figure 3.25: Loss Function Evolution

As we can see, the loss function is decreasing to reach the value of 0.08 at it's final

state. Now let's see the inputs/ouputs results of the finetuned model

The models:

Figure 3.26: Model Input Example

So as the training data, the model contains the attributes and it's methods for this example we took a book model that have isbn, title, author, price as attributes and two methods that are getDetails and updatePrice this model doesn't have many to many attributes nor Foreign Keys and we can see below the output of the model (in figure 3.27) when we gave this json file as an input.

```python
from django.db import models
from django.conf import settings


class Book(models.Model):
    isbn = models.CharField(max_length=20, primary_key=True)
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=200)
    price = models.FloatField(null=True, blank=True)

    class Meta:
        permissions = (
            ("view_book", "Can view book"),
        )

    def getDetails(self):
        return f"ISBN:{self.isbn}, Title:{self.title}, Author:{self.author}"

    def updatePrice(self,new_price):
        self.price = new_price
        self.save()
        return self.price

    def __str__(self):
        return f"{self.title}"
```

Figure 3.27: Model Output Example

As it's shown in the figure 3.27 above the AI model generated a code with the same structure as infinity management's previous code it have the same structure.

The Serializers:

```json
{
    "file_name": "absence_sheet_serializer.py",
    "serializer_name": "AbsenceSheetSerializer",
    "type": "serializer",
    "functions": [
        "get_folder_group_details",
        "get_absences_ids"
    ]
}
```

Figure 3.28: Serializers Input Example

```python
from ...core.serializers import GenericSerialiser, NestedGenericSerialiser
from ...academic_path_educational_stage.models import AcademicPathEducationalStage
from ...educational_stage.models import EducationalStage
from rest_framework import serializers
from ..models import AcademicPath
from ...core.mixins.serializers import NestedMixin

class AcademicPathSerializer(GenericSerialiser):
    educational_stages = NestedMixin(
        many=True
    )

    class Meta(GenericSerialiser.Meta):
        model = AcademicPath
        fields = (
            'id',
            'libelle_fr',
            'libelle_ar',
            'educational_stages',
            'is_active'
        )
```

Figure 3.29: Serializers Output Example

The Views :

```
"book_views.py": [
    {
        "file_name": "book_views.py",
        "view_name": "BookListCreateAPIView",
        "type": "views",
        "functions": [
            "get"
        ]
    },
    {
        "file_name": "book_views.py",
        "view_name": "BookRetrieveUpdateAPIView",
        "type": "views",
        "functions": [
            "put",
            "patch",
            "get"
        ]
    }
],
```

Figure 3.30: Views Input Example

```python
from rest_framework import generics
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response

from ..models import Book
from ..serializers import BookSerializer


class BookListCreateAPIView(generics.ListCreateAPIView):
    queryset = Book.objects.all()
    serializer_class = BookSerializer
    permission_classes = [IsAuthenticated]

    def get(self, request, *args, **kwargs):...


class BookRetrieveUpdateAPIView(generics.RetrieveUpdateAPIView):
    queryset = Book.objects.all()
    serializer_class = BookSerializer
    permission_classes = [IsAuthenticated]

    def put(self, request, *args, **kwargs):...

    def patch(self, request, *args, **kwargs):...

    def get(self, request, *args, **kwargs):
        return self.retrieve(request, *args, **kwargs)

#apps/core/views/__init__.py
from .user_views import UserListCreateAPIView, User
```

Figure 3.31: Views Output Example

3.3.6 Model Deployment

Now after the model is set up and ready to generate the code, the next and final step of the project is to make it easy for the developers to use the model and generate the code. First of all I designed an excel template where the users could enter the details and the conceptional design of their projects. After that the script will transform that excel data into json form and pass it through the AI model to generate the directory of their projects. In the figure 3.32 below we find an example of the excel template :

file_name	type	class	functions	attributes	m2m_attributes	ForeignKey_attributes
book1.py	model	Book		BookId,Title,ISBN,Price,PublicationDate	Author,Category,Order	
author.py	model	Auther		AuthorId,Name,Biography	Book	
category.py	model	Category		CategoryId,Name,Description	Book	
customer.py	model	Customer		CustomerId,name,email,address		Order
order.py	model	Order		OrderId,OrderDate,TotalAmount,Customer	Book	
orderitem.py	model	OrderItem		OrderItemID,Quantity,Price		OrderId,BookID
book_views.py	views	BookListCreateAPIView	get			
book_views.py	views	BookRetrieveUpdateAPIView	put,patch,get			
author_views.py	views	AllAuthorListAPIView				

Figure 3.32: Excel Template Example

in this template the user should write the file name, the type of the file if it's a model, a serializer or views and then what is the class name give the classes their methods, attributes, many to many fields and the foreign key. After the user submitting the excel file and executing the script, the data of that excel file will be transformed into a json file the script in the figure 3.33 below shows the steps of converting that excel file to json.

```python
import pandas as pd
import json

df = pd.read_excel("Excels/conception model.xlsx")
df = df.astype(str)
final_json = {}
for i in df.index:
    file_name = df.at[i,"file_name"]
    row_type = df.at[i,"type"].strip()
    row_class = df.at[i,"class"]
    functions = list(filter(lambda x: x != "nan", df.at[i,"functions"].split(",")))#df.at[i,"function


    attributes =list(filter(lambda x: x != "nan", df.at[i,"attributes"].split(",")))
    m2m_attributes = list(filter(lambda x: x != "nan", df.at[i,"m2m_attributes"].split(",")))
    foreignkeys = list(filter(lambda x: x != "nan", df.at[i,"ForeignKey_attributes"].split(",")))
    print(file_name ,
    row_type ,
    row_class ,
    functions ,
    attributes ,
    m2m_attributes ,
    foreignkeys )
    if row_type == "model":...
    elif row_type == "views":...
    else:...
    if file_name in final_json.keys():
        final_json[file_name].append(new_dict)
    else :
        final_json[file_name] = []
        final_json[file_name].append(new_dict)
filename = 'data.json'

# Open the file in write mode and write the dictionary to it
with open(filename, 'w') as file:
    json.dump(final_json, file, indent=4)
```

Figure 3.33: Excel To JSON Script

In the figure below we can see the transformed data of the excel given in example at figure 3.32.

Figure 3.34: Example Of JSON Output

Finally after converting the Excel file to json, we can call the model to generate the code. First of all I call the finetuned model.

```python
import torch
from peft import PeftModel, PeftConfig
from transformers import AutoModelForCausalLM, AutoTokenizer
from transformers import pipeline
import json
peft_model_id  = "Sloozi/deepseek_json_adapter"
config = PeftConfig.from_pretrained(peft_model_id)
pretrained  = AutoModelForCausalLM.from_pretrained(
    "deepseek-ai/deepseek-coder-6.7b-instruct",
    device_map='auto',
)
tokenizer = AutoTokenizer.from_pretrained("Sloozi/deepseek_json_adapter")
model = PeftModel.from_pretrained(pretrained, peft_model_id)
```

Figure 3.35: Calling the model Script

The files will be processed by the script below. the script is reading each file data separately generate the code corresponding to that json and then write each file to it's directory.

```python
with open(filename, 'r') as file:
    data = json.load(file)


for input_file in data.keys():
    input_text = json.dumps(data[input_file])
    inputs = tokenizer(input_text, return_tensors="pt").to(model.device)
    outputs = model.generate(**inputs, max_new_tokens=512,do_sample=True,
        temperature=0.2)
    output_text = tokenizer.decode(outputs[0],skip_special_tokens=True)
    if "_views" in input_file:
        directory = "views"
    elif "_serializer" in input_file:
        directory = "serializers"
    else:
        directory = "models"
    with open(f"python output/{directory}/{input_file}","w") as f:
        f.write(output_text[len(input_text):])
    print(f"{input_file} saved successfully")
```

Figure 3.36: Generating the code source Script

3.4 Conclusion

Chapter 3 serves as the pulsating core of our project, unveiling the intricate journey of its execution. With meticulous planning and unwavering commitment, we have translated our vision into tangible action, intertwining threads of innovation and advancement. Every aspect, from the careful selection of foundational software and tools to the strategic orchestration of each phase, reflects our relentless pursuit of excellence.

General Conclusion

This chapter encapsulates the project's comprehensive journey and notable achievements, emphasizing significant advancements in innovation and technological sophistication. Initiated by Infinity Management, the project aimed to enhance and optimize developers' workflows through the integration of AI-driven automation within the Django framework. The preliminary phases included a meticulous examination of existing workflows to identify inefficiencies and repetitive tasks amenable to automation. This scrutiny revealed the extensive burden of boilerplate code, which, despite its necessity for web application functionality, consumed substantial time and resources.

In response to these challenges, the project deployed advanced AI models to automate the generation of boilerplate code, thereby allowing developers to redirect their efforts toward more strategic and creative endeavors. The implementation phase was distinguished by rigorous development, the fine-tuning of AI models, and comprehensive validation processes to ensure that the solutions were precisely aligned with the developers' needs. The utilization of tools such as Python, Visual Studio Code, OpenAI, and Hugging Face proved instrumental in creating a robust and efficient development environment.

The project's successful realization not only met its immediate objectives but also established a framework for future technological initiatives. The insights gained and the expertise cultivated during this project serve as a robust foundation for continuous

innovation and improvement. The report underscores the project's role as a catalyst for future exploration and boundary-pushing in the realm of AI and technology.

While the project achieved its primary goals, it is essential to adopt a critical perspective on its outcomes and identify areas for future improvement. One of the challenges encountered was the initial adaptation period required for developers to effectively utilize the AI-driven tools. This highlighted the need for ongoing training and support to maximize the benefits of the implemented solutions.

Additionally, the project's reliance on specific AI models and tools like OpenAI and Hugging Face raised questions about dependency and long-term sustainability. Future projects could benefit from exploring a wider range of AI technologies to mitigate potential risks associated with dependency on particular vendors or platforms.

Looking forward, the emphasis remains on sustained innovation and the pursuit of excellence. Concrete steps for future work include expanding the AI models' capabilities to cover more complex aspects of development beyond boilerplate code and exploring integration with other popular frameworks and tools to broaden the project's applicability. Moreover, incorporating feedback mechanisms to continuously refine and adapt the AI models based on real-world usage will be crucial.

The commitment to applying the acquired knowledge and skills will facilitate further successes and adaptations to the rapidly evolving technological landscape. This forward-thinking perspective ensures that the project's impact will transcend its initial scope, fostering ongoing enhancement and innovation within the field. By addressing the identified challenges and leveraging the project's foundations, future initiatives can build upon this work to achieve even greater advancements in AI-driven development.

Bibliography

[1] Infinity Management official website. URL: https://infinitymanagement.fr/ (visited on 02/2024).

[2] What is Agile Methodology? https://www.geeksforgeeks.org/what-is-agile-methodology/. (Visited on 03/2024).

[3] What is a Neural Network? https://www.ibm.com/topics/neural-networks. (Visited on 03/2024).

[4] Stephen M. Walker II. What is artificial intelligence (AI)? https://klu.ai/glossary/artificial-intelligence. (Visited on 03/2024).

[5] Amanatulla. Fine-Tuning the Model: What, Why, and How. https://medium.com/@amanatulla1606/fine-tuning-the-model-what-why-and-how-e7fa52bc8ddf. Sept. 21, 2023. (Visited on 03/2024).

[6] NVIDIA. Generative AI. https://www.nvidia.com/en-us/glossary/generative-ai/. (Visited on 03/2024).

[7] IBM. What is natural language processing (NLP)? https://www.ibm.com/topics/natural-language-processing. Sept. 21, 2023. (Visited on 03/2024).

[8] AWS. What are Large Language Models (LLM)? https://aws.amazon.com/what-is/large-language-model/. (Visited on 03/2024).

[9] Rick Merritt. What Is a Transformer Model? https://blogs.nvidia.com/blog/what-is-a-transformer-model/. Mar. 25, 2022. (Visited on 03/2024).

[10] Daya Guo et al. DeepSeek-Coder: When the Large Language Model Meets Programming – The Rise of Code Intelligence. https://github.com/deepseek-ai/DeepSeek-Coder. Jan. 26, 2024. (Visited on 03/2024).

[11] Ben Lutkevich. 18 of the best large language models in 2024. https://www.techtarget.com/whatis/feature/12-of-the-best-large-language-models. Apr. 5, 2024. (Visited on 05/2024).

[12] Python Official Website. https://www.python.org/.

[13] Visual Studio Code Official Website. https://code.visualstudio.com/. (Visited on 03/2024).

[14] Open AI Official Website. https://openai.com/. (Visited on 03/2024).

[15] Hugging Face Official Website. https://huggingface.co/. (Visited on 03/2024).

[16] GitHub Official Website. https://github.com/. (Visited on 03/2024).

[17] Sayak Paul and Sourab Mangrulkar. PEFT: Parameter-Efficient Fine-Tuning of Billion-Scale Models on Low-Resource Hardware. https://huggingface.co/blog/peft. Feb. 10, 2023. (Visited on 03/2024).