POLITECNICO DI TORINO

Master's Degree course in LM-32 (DM270) - Computer Engineering (Software)
Department of Control and Computer Engineering (DAUIN)

Master of Science's Thesis

"Agent Code, James Code": AI-based code generation framework

Supervisor
Prof. Riccardo Coppola

Candidate
Giovanna Di Benedetto
matricola: 306136

Internship Tutor
Dott. Gianmarco Dragonetti

Academic year 2023-2024

Summary

In contemporary society, Artificial Intelligence (AI) permeates numerous facets of daily life, including healthcare, manufacturing, and education. However, the fields of computer science and computer engineering have traditionally emphasized enhancing efficiency over practical implementation in daily environments, focusing on areas such as algorithm optimization, network traffic management, and compiler optimization. This thesis addresses this gap by constructing a framework aimed at aiding developers in resolving GitHub issues through AI-driven solutions. By doing so, developers can interact with AI in a more straightforward and practical manner.

Central to this framework is an Agent, serving as the interface between human requests and AI responses. Adapted from the open-source LangChain framework, this Agent leverages AI to analyze issues, generate code solutions, develop corresponding unit tests (if required), and execute necessary code modifications post-review. By interfacing directly with GitHub via its toolkit, the framework employs GitHub Actions for code building and GitHub Webhooks to trigger Agent operations.

Notably, the framework's AI component relies on OpenAI's ChatGPT, renowned for its sophistication and versatility. Nevertheless, this framework is built with the aim of making it as customizable as possible to accommodate various AI models as they evolve. Since GitHub is the most well-known and widely used service for hosting code, the Agent's integration with GitHub ensures a seamless workflow for developers.

This thesis not only presents a practical solution for streamlining issue resolution on GitHub but also underscores the evolving landscape of AI in computer science. As the AI domain continues to burgeon, this framework represents a dynamic avenue for future exploration and refinement. The project developed through this internship and analyzed in this thesis is prone for evolution in the near future, driven by the ongoing advancement of AI

technology and the increasing incentive to study this growing branch of computer science.

Contents

List of Figures

Chapter 1

Introduction

1.1 Purpose of the Thesis

The objective of this thesis is to take advantage of artificial intelligence (AI, 2.1.1) to automate several critical steps in the software development process, such as writing unit tests or developing code to solve issues opened in a GitHub repository.

The project of this thesis was developed during an internship at Blue Reply[1], from February to July 2024.

The initial objective of this thesis was to take advantage of artificial intelligence to do in a faster way activities that may take more time than needed.

One of these activities is writing unit tests since it is needed to follow the software development best practices. However, most of the time, these tests are not done correctly or may cause a waste of time in terms of working hours.

To solve this problem, it has been conducted an analysis (3.2.3) on how the AI behaves when writing Java unit tests (from a qualitative point of view). If the performance had been satisfying, the second part of the thesis would have been doing the opposite process: making the AI generate code starting from unit tests.

Nevertheless, this initial idea was changed to a project that could be potentially used more in a working environment: an **AI-based code generation**

[1]https://www.reply.com/blue-reply/it/

framework, that uses various GitHub features (webhooks, apps, actions) to automate the code generation, the unit test generation and the bug fixing.

Since the tools used are called Agents (3.5.1), this thesis has the title **"Agent Code, James Code"**, to make wordplay with the famous movie series: as James Bond's abilities have been crucial to complete his missions, the Agents in this project are the building block of the entire framework and represent a way in which the artificial intelligence can lighten the human workload and, in general, can be used for a good purpose.

In the following sections, there will be an explanation of how everything is built:

- Chapter 2 (*Background*) answers the question *"what is this?"*: there is an explanation of all the concepts that will be used in this project, such as AI, unit testing and GitHub interaction;
- In chapter 3 (*Methodology*), the attention is on *"how it is done?"*: how the topics explained in the previous chapter take place inside the framework, how the thesis ideas evolved, the difficulties met during the various analyses and implementations and the workarounds used to reach the solution;
- Chapter 4 (*Results*) contains the whole structure and flow of the final application, the solved puzzle made by the correct joint of the elements presented in the preceding chapter, and answers to *"what is the outcome?"*;
- Lastly, chapter 5 (*Conclusions and Future Work*) contains the answer to *"is there anything to add?"*: this project is just the start of something that could be a turning point in the software development field and many areas can help this basic framework to become the next big thing in computer science.

Chapter 2

Background

2.1 Artificial Intelligence (AI)

Before analyzing in depth what has been done during this thesis project, it is crucial to explain the main organism involved, with all its benefits and the difficulties a person may encounter when interacting with it: Artificial Intelligence.

2.1.1 What is AI?

Even though it is becoming more and more common to use it, it is quite difficult to define what AI is: based on the way it is used, it can be a blessing, since it makes it possible to do slow tasks in a faster way, as well as a curse, since the outcome of the task done by the AI may be wrong.

So, *what is AI*? **Artificial intelligence** is a field of science concerned with building computers and machines that can *reason, learn, and act* in such a way that would normally require human intelligence or that involves data whose scale exceeds what humans can analyze [1, 2].

As stated by Bharadiya [3], artificial intelligence (AI) is a technique for simulating human intellect using a collection of algorithms to create a new computer that can accomplish similar tasks to humans while also performing parallel computing.

AI is a broad field that encompasses **many different disciplines**, including computer science, data analytics and statistics, hardware and software engineering, linguistics, neuroscience and even philosophy and psychology.

On an operational level for business use, AI is a *set of technologies* that are based primarily on *machine learning* and *deep learning* (2.1.4). These disciplines involve the development of AI algorithms, modeled after the decision-making processes of the human brain, that can *learn* from available data and make increasingly more accurate classifications or predictions over time.

Artificial Intelligence can be distinguished into two kinds: weak AI and strong AI [4].

Weak AI (also known as *narrow AI*) is AI trained and focused on performing specific tasks. Since “narrow” is a better term to describe how strong this AI is, it is also generally known as *artificial narrow intelligence* (ANI). It does not exhibit any creativity, nor does it have the explicit ability to independently learn in the universal sense. Its learning abilities are mostly limited to training in detection patterns (machine learning) or comparison and search operations with large quantities of data. Using weak AI, clearly defined tasks can be handled based on a defined methodology to solve more complex problems which, however, are recurrent and precisely specified. The benefits of weak AI are especially relevant in automation and control processes as well as in speech recognition and processing. To make a few examples, it is the basis of Amazon’s Alexa, Apple’s Siri and self-driving vehicles.

On the other side, **strong AI** is the basis of *artificial general intelligence* (AGI) and *artificial super intelligence* (ASI), two theoretical concepts that may have practical applications in the future. The realization of strong AI is not yet within reach: the objective underlying the idea of strong AI is to allow natural and artificial intelligence media (e.g. humans and robots) to establish a level of *mutual understanding and trust* when working in a joint field of activity. Thus, efficient human-machine collaboration could be learned and facilitated. Strong AI can independently recognize and define tasks and independently develop and expand upon knowledge in the corresponding application domain. Strong AI studies and analyses problems to find an adequate solution and the problems can also be new or creative.

- **AGI** (also known as *general AI*) is a theoretical form of AI where a machine would have an intelligence equal to humans; it would be self-aware with a consciousness that would have the ability to solve problems, learn and plan for the future;

- **ASI** (also known as *superintelligence*) aims to surpass the intelligence and ability of the human brain. To be classed as an ASI, the technology would have to be more capable than a human in every single way possible. This means that not only these AI tools can carry out tasks, but they would even be capable of having emotions and relationships [5].

2.1.2 Artificial Intelligence applications

There are numerous real-world applications for AI systems and it is an environment that will evolve rapidly, covering many aspects of daily life. These applications are becoming increasingly common in a wide variety of industries (healthcare, finance, retail, manufacturing, etc.). Generally, artificial intelligence can be applied in the following ways:

- **Natural Language Processing** (NLP): it allows computers to understand human language. This technology is used in a variety of applications (machine translation, spam filtering, etc.) (2.1.3);
- **Computer vision**: it allows computers to identify and interpret visual content. This technology is used in a variety of applications, such as self-driving cars, facial recognition and object detection;
- **Machine Learning**: it allows computers to learn from data and improve their performance over time. This technology is used in applications, such as predictive analytics and recommendation systems (2.1.4);
- **Robotics**: this is the branch of AI that deals with the design, construction and operation of robots. Robots are used in a variety of applications, such as manufacturing, healthcare and space exploration.

Business Intelligence

Going into detail, AI is becoming increasingly important in **business intelligence**: AI-powered tools can help companies collect, analyze and see data more efficiently. As a direct consequence, there is an increment in productivity, improvement in decision making and reduction of costs.

AI can be included in this field through *data collection* (including structured data, such as databases, and unstructured data, such as text documents); *data analysis* (to identify patterns and relationships), *data visualization* (to understand data more easily) and in *decision-making* (thanks to insights and recommendations generated by AI models).

As stated by Bharadiya [3], **business intelligence** (BI) is a broad category of information technology (IT) solutions that include tools for **acquiring, analyzing, and reporting data** to users about an organization's performance and its surroundings. These IT solutions are among the highest-priority investment opportunities.

It is a strategy and method of **boosting corporate performance** by giving strong support to executive decision-makers, allowing them to have actionable information at their fingertips. BI tools are considered a technology that improves the effectiveness of company operations by increasing the value of corporate data and, as a result, the way that data is used.

BI technology attempts to assist individuals in **making better business choices** by providing accurate, current, and relevant data when needed. Competitive firms gather BI to analyze the environment to achieve a lasting competitive edge, and in some cases, such intelligence may be regarded as a valued core capability.

Analysts and managers may use it to determine which modifications are most likely to respond to shifting patterns. It's evolved into a paradigm for analyzing data and assisting decision-making units.

Kaushik [6] affirms that using Artificial Intelligence in marketing would **benefit entrepreneurs** with higher returns and responses from customers and make them achieve strong competition in the market. Other than marketing it can **renovate the business** with invention and attractive ideas.

Artificial Intelligence is considered essential for business houses as it predicts choices of the customer and helps sales expansion through data analysis and automation.

Lastly, Pallathadka et al. [7] state that in the industry of finance and e-commerce, Artificial Intelligence has been used for the achievement of better customer experience, effective management of supply change, improved efficiency of operation of business, reduction in mate size having the main aim of designing standards, consistent and dependable methods of product quality control, and searching new methods of serving and reaching customers at low maintenance costs.

Healthcare

Another field in which AI help is becoming dominant is **healthcare**: it can help doctors in many ways, such as *disease diagnosis* (by analyzing patient data and identifying patterns), *treatment development* (identifying patterns given by large amounts of patient data, to develop new therapies) and *personalized care* (the patient's specific needs can be satisfied through a deep analysis of the data and a resulting treatment plan).

With deep learning algorithms meeting, and in some cases surpassing, the performance of clinicians, AI is positioned to have a major role in a range of healthcare delivery areas, including diagnostics, prognosis, and patient management [8, 9].

Pharmaceutical firms have benefited from AI in healthcare by speeding up their drug discovery process and automating target identification. Artificial Intelligence (AI) can also help to eliminate time-consuming data monitoring methods. Shaheen's study [10] also indicates that AI-assisted clinical trials are capable of handling massive volumes of data and producing highly accurate results. Medical AI companies develop systems that assist patients at every level. Patients' medical data is also analyzed by **clinical intelligence**, which provides insights to assist them improve their quality of life.

Sharing pre-trained and approved AI models could help solutions adapt faster to different situations. Models used to diagnose illness from pictures, forecast patient results, filter misinformation and disinformation depending on propagating patterns through social media, and distill knowledge graphs from massive collections of scholarly papers are instances of algorithms that could be broadly useful [11, 12, 13].

Through his study, Woo [14] noted that **applying predictive AI models** and advanced analytics to unlock real-world data (RWD) can help researchers **better understand diseases**, find relevant patients and important investigators, and enable revolutionary clinical study designs.

In combination with an efficient digital infrastructure, clinical trial data might be cleansed, aggregated, coded, preserved, and maintained using AI algorithms [15].

AI-aided technologies and their applications can provide lifestyle interventions and daily reminders based on an individual's vital signs through digital devices. Within healthcare organizations, AI-based technologies are set to significantly **transform how healthcare systems operate**, optimize, and interact with patients, and provide care services to increase the overall efficiency of patient outcomes [16].

For example, Dawes et al. [17] reported how patients with high blood pressure and lung disease can be treated with more accurate data based on an AI-supported magnetic resonance imaging (MRI)-based algorithm of cardiac motion.

AI-enabled devices, such as **personal assistants** could significantly influence the monitoring and support of patients at times when medical staff are unavailable. AI-supported smart robots can also perform operations and augment physicians' work with certain diagnoses, treatment methods, cost and time reduction, and improved response time to patients' needs [18].

From a pharmaceutical point of view, Díaz et al. [19] noted that AI technology in healthcare has helped companies **speed up their drug discovery process**. It, on the other hand, **automates the identification of targets**. In addition, by analyzing off-target compounds, AI in healthcare 2021 aids in drug repurposing.

AI can overcome limitations with traditional rules-based clinical decision support systems and enable better diagnostic and decision support. Opportunities to automate triage and screen and administer treatment are also becoming a reality. AI embedded in smart devices, supported by the Internet of Things and fast Wi-Fi, could bring AI-enabled health services into patient's homes, thus democratizing health care [8, 20, 21].

As a result, **clinical trial automation** has proven to be a benefit for AI and the healthcare business. Furthermore, Artificial Intelligence and healthcare assist in the elimination of time-consuming data monitoring procedures. Additionally, AI-assisted clinical trials handle large amounts of data and produce very accurate outcomes [10].

Artificial intelligence has the potential to help healthcare providers in a variety of ways, including patient treatment and administrative tasks: healthcare staff is often inundated with much paperwork in the care process. This workload has prompted the industry to transition to electronic systems that integrate and **digitize medical records**, which are aided by AI-based technology. In addition, the use of **chatbots** has been identified as a potentially effective tool for engaging in conversation with patients and family members in hospitals [22].

The continuous research in the use of AI systems will greatly augment the work of medical staff as they can alert some areas that humans often miss or help minimize medical errors during the patient treatment.

If proponents of these strategies are correct, AI and machine learning will bring in a **new era of drug development** that is faster, cheaper, and more effective.

The expanded use of AI is undoubtedly creating a major change in the healthcare service market and the spread rate should increase as advances in AI accelerate in the future [16, 23].

Education

The third main field in which AI can be used in an efficient way is **education**: it can *improve the students' engagement* (by providing interactive and engaging learning experiences), *automate administrative tasks* (such as grading papers or scheduling classes) and *personalized learning* (identifying areas where the student needs additional support by tracking their progress).

With the thriving of AI technology, its applications in education have been increasing, with promising potential to provide customized learning, offer dynamic assessments and facilitate meaningful interactions in online, mobile or blended learning experiences [24].

Hwang et al. [25] investigated the effects of a fuzzy expert system on elementary students' math learning outcomes in Taiwan. In this study, students in the experimental group outperformed those in the other two groups in mathematics learning achievement. In addition, the adaptive learning model with affective and cognitive performance analysis was found effective in reducing math anxiety among fifth graders in Taiwan.

One of the main contributions of AI in the world of education is the usage of intelligent tutors. **Intelligent tutors** or **agents** provide customized, timely, and appropriate materials, guidance, and feedback to learners. With great potential, research indicates mixed implications regarding its effects on learning.

A study with high school students in the USA found that connecting math to students' personal interests that were not school-related would increase learning in an intelligent tutoring system, and thus highly customized personalization could promote learning and thus may lead to student success [26].

Artificial Intelligence in Education (AIEd) research carried out by Dias et al. [27] suggested that dynamic, holistic expert systems can help with pedagogical planning and fully unleash the potential of **learning management systems** (LMS) for teaching and learning. Their study proved that the structural characteristics of an expert system can model how LMS users interact with it, and thus facilitate and improve the teaching and learning experiences on the LMS.

A study with over 1300 participants in Hong Kong investigated an AI-enhanced e-learning system called *SmartTutor* [28]. It was found that customized learning materials and resources were well received and both students and faculty confirmed that they were helpful in the teaching/learning process.

Hwang et al. also demonstrated the potential of AI in addressing learners' affective or emotional needs, which in turn may improve learning. It also suggests the need for more inclusive designs of AIEd technologies to address students' varied needs and preferences.

A similar concept of the intelligent tutor is the **Teaching Assistant** (TA). Gulz et al. [29] indicated that young children perceived the TA as an independent entity, and researchers thus suggested that TA was promising in facilitating metacognitive scaffolding.

Research also suggested that TA with a similar level of self-efficacy with target students may help improve learners' performance in math [30].

Another useful feature of AIEd is the development of **intelligent feedback systems**, which can also measure how people learn, in addition to what is learned. Machine learning, for example, can predict at-risk or marginal college students as well as gifted students with high accuracy, which then empowers educators to intervene accordingly for student success [31, 32, 33].

Regarding this, Kose [34] found that personalized mobile learning, via AI and Augmented Reality (AR), improved learning experiences as well as learning outcomes in open computer education.

In another study, machine learning algorithms were used to **predict undergraduate students' attitudes** toward educational applications of cloud-based mobile computing services by their information management behaviors with 74% accuracy [35].

Lastly, a study on a smart glass system also confirmed that AI technology with visualizations helped both children and adults with autism, by serving as a social communication aid [36].

To sum up, for learners, AIEd may facilitate varied interactions, increase learner engagement, generate adaptive learning materials, offer meta-cognitive prompts, provide enriched learning environments, and improve learning outcomes. For educators and administrators, AIEd may provide predictive models, identify gifted or at-risk students, monitor the learning progress, create personalized learning materials, assessments and feedback, and analyze scaled data instantly for evaluation or administrative purposes. AI-enhanced learning environments may improve the LMS for both instructors and students through expert systems, generate visual feedback, and enrich the learning experience with visualization and immersive technologies.

Agriculture

Additionally, it can be used in **agriculture**. Here it can *improve crop yield* (by analyzing data on soil conditions or weather patterns), *reduce labor costs* (by automating tasks such as field irrigation) and *protect the environment* (by monitoring and managing natural resources, such as water and soil).

Agriculture entails a variety of processes and phases, the majority of which are performed manually. AI can help with the most complex and routine jobs by **supplementing existing technology**. When integrated with other technology, it can gather and evaluate massive data on a digital platform, determine the best course of action, and even initiate that action [37].

Farmers can create more output with less input, improve the quality of their product, and ensure a faster time to market for their harvested crops owing to AI-based technological solutions [38].

By analyzing soil management data sources such as temperature, weather, soil analysis, moisture, and historic crop performance, AI systems are able to provide predictive insights into which crop to plant in a given year and when the optimal dates to sow and harvest are in a specific area, thus improving crop yields and decrease the use of water, fertilizers, and pesticides.

Via the application of AI technologies, the impact on natural ecosystems can be reduced, and worker safety may increase which in turn will keep food prices down and ensure that the food production will keep pace with the increasing population [39].

The first way to adopt AI in agriculture is **soil management**, since it is one of the most important parts of the agricultural process. To do so, there are various techniques:

- **Management-oriented modeling** (MOM) minimizes nitrate leaching as it consists of a set of generated plausible management alternatives, a simulator that evaluates each alternative, and an evaluator that determines which alternative meets the user-weighted multiple criteria. MOM uses "hill climbing" as a strategic search method that uses "best-first" as a tactical search method to find the shortest path from start nodes to goals [40];
- The second way is to classify soil according to associated risks: knowledge of engineering for constructing the **Soil Risk Characterization Decision Support System** (SRC DSS) involves knowledge acquisition, conceptual design and system implementation [41];
- Lastly, an **artificial neural network** (ANN) model can be used to predict soil texture (sand, clay and silt contents) based on attributes obtained from existing coarse-resolution soil maps combined with hydrographic parameters derived from a digital elevation model (DEM) [42].

Crop management is the second field in which AI can be used in agriculture. **Precision crop management** (PCM) is an agricultural management system designed to target crop and soil inputs according to field requirements to optimize profitability and protect the environment. PCM has been hampered by the lack of timely, distributed information on crop and soil conditions [43].

PROLOG is a technique that utilizes weather data, machinery capacities, labor availability, and information on permissible and prioritized operators, tractors, and implements for evaluating the operational behavior of a farm system. It also estimates crop production, gross revenue, and net profit for individual fields and for the whole farm. In short, it removes less-used farm tools from the farm [44].

Other than crop management, another related way to use AI is **crop prediction** methodology. This is used to predict the suitable crop by sensing various soil parameters and parameters related to the atmosphere. Parameters like soil type, PH, nitrogen, phosphate, potassium, organic carbon, calcium, magnesium, sulfur, manganese, copper, iron, depth, temperature, rainfall and humidity [45].

About **harvesting**, a study has been conducted to analyze how cucumber growth changes. The use of AI in harvesting cucumber comprises the individual hardware and software components of the robot including the autonomous vehicle, the manipulator, the end-effector, the two computer vision systems for detection and 3D imaging of the fruit and the environment and, finally, a control scheme that generates collision-free motions for the manipulator during harvesting [46].

The last aspect related directly to the field is **weed management**. A system can utilize an unmanned aerial vehicle (UAV)-imagery to divide images, compute and convert to binary the vegetation indexes, detect crop rows, optimize parameters and learn a classification model. Since crops are usually organized in rows, the use of a crop row detection algorithm helps to separate properly weed and crop pixels, which is a common handicap given the spectral similitude of both [47].

Accuracy agriculture uses AI to help identify plant ailments, irritations, and helpless plant nutrition on ranches. Weeds may be detected and targeted using simulated intelligence sensors, which can then be used to identify which herbicides to administer in the appropriate buffer zone. Pesticides and other potentially harmful substances are kept out of our food by prohibiting their misuse [48].

From an analytical point of view, **data analytics** in agriculture can result in large production gains and significant cost reductions. Farmers can acquire recommendations based on well-sorted real-time information on crop needs by merging AI and big data. As a result, guesswork will be eliminated, allowing for more exact farming methods such as irrigation, fertilization, crop protection, and harvesting [49].

Predictive analytics has the potential to be a game changer. Farmers can collect and process substantially more data using AI than they could without it, and they can do so much faster. Farmers may use AI to handle critical difficulties including analyzing market demand, projecting pricing, and deciding the best time to sow and harvest [50].

Farmers can save money by using driverless tractors, smart irrigation and fertilization systems, smart spraying, vertical farming software, and AI-based harvesting robots. AI-driven farm tools are faster, tougher, and more accurate than any human farm worker [51].

Although AI has made some remarkable improvements in the agricultural sector, it still has a below-average impact on agricultural activities when compared to its potential and impacts in other sectors. More still needs to be done to improve agricultural activities using AI as there are many limitations to its implementation.

Manufacturing

In the case of **manufacturing**, AI is directly linked to *improvement*, in *efficiency* (by automating tasks), *productivity* (with production processes optimization) and *quality* (by detecting defects).

Recent developments in artificial intelligence (AI) and the extensive amount of generated manufacturing data, known as big data, are allowing the integration of new kinds of analytics tools in the supply chain, which are optimizing the way goods are produced.

With such systems, the quality stays constant, which benefits businesses by increasing their customer satisfaction, while the production time and costs involved for a product or a service decrease. Several approaches in machine learning are available for such methods. Supervised learning can be used to differentiate between certain characteristics for products that have only a limited number of features. With sufficient data available, it is therefore possible to perform a classification task and thus find quality defects more quickly. These classifications can be optimized by neural networks and trained to near perfection.

One of the main reasons why preventative maintenance could be useful is the capability to predict when a mechanical part may require replacing. Combined with historical evidence, machine learning produces an algorithm that detects possible problems when they emerge, helping organizational specialists take the steps required to eliminate problems that can delay or even interrupt development [52].

In **supply chain management** (SCM), for example, AI is used to plan the routing of products optimally and dynamically. With such applications, AI can be used to achieve many benefits in both manufacturing and SCM. By optimizing processes and minimizing errors, costs and manpower can be saved. However, in order to exploit the full potential of AI, a certain level of commitment is required. The integration of such concepts is a step that requires investment of resources and time [53, 54].

Machine learning is in general important to **optimize the decision-making process** in the flow of goods and services alongside supply chain management. Properly applied, these methods can lead to time and resource savings. In particular, the planning process can benefit from well-known statistical methods that have long been used and extended by ML. Especially for non-linear problems, ML has a fundamental advantage over more traditional methods. Despite the clear benefits, a study shows that ML with one or more supply chain functions, was applied in only 15% of companies. The lack of data or ignorance about the subject could be reasons why such methods are not yet much more widespread. These technological developments influence the purchasing and supply management function and the personnel deployment and will improve those systems in the future [55, 56, 57].

In **sales forecasting** for short shelf-life and highly perishable goods, the advantages of applied AI methods exceed the precision level of conventional statistical techniques and as a consequence, boost inventory balance across the chain, minimize stock-out rates, enhance supply and increase profitability [58].

In areas such as the **stock market** or **crypto exchanges**, trading bots that use deep reinforcement learning have been successful. Reinforcement learning algorithms, in contrast to the other machine learning methods mentioned earlier, do not need as much data to achieve promising results. This approach is much more oriented towards human learning behavior of learning optimal ways through trial and error. The agent needs an environment that the learner can influence with their own actions. The environment sends him rewards for his actions. The goal of the agent is to maximize these rewards to find the best way for the environment. Adapted to the market, the required parts for a reinforcement learning approach would be clear. The agent would be the bot, the environment would be the market, and the reward would be the profit or loss generated [59].

One of the most valuable advantages of AI in manufacturing is **real-time monitoring**, as it gives a more accurate description of where and if any inefficiencies exist in the production chain and what causes the bottleneck.

The potential to identify the exact process that needs adjusting, helps organizations to solve the problem rapidly, resulting in time and cost savings. The benefits illustrated in the work of Kumar et al. show that cloud manufacturing, a real-time monitoring method, may lead to increased efficiency of resources by recognizing the current machine state, minimizing system downtime with the help of condition-based real-time tracking through analysis of the obtained sensor data. This information can then be reused by machine-to-machine communication protocols and cloud service data retrieval methods. Moreover, this concept can help small and medium-sized companies (SMEs) registered in the network, which can benefit from this cooperation and provide cost-effective production services with short lead times [60].

One of the most widely used methods for incorporating AI in purchasing departments is to **automate and optimize processes**. These improvements can be achieved with similar techniques, which are used in supply chain management. Globalization has also increased the number of markets. The large range of products makes it difficult for people to keep an overview. This is where neural networks, which classify the offers according to certain features, can be used again to make the purchase decision autonomously or to make it immensely easier. Through this application, companies could develop a non-negligible advantage over competitors who do not use such methods [56].

The continuous evolution of smart cities, intelligent medical care, intelligent transportation, intelligent logistics, intelligent robots, self-driving vehicles, smart-phones, intelligent toys, smart communities, and smart economies, to name but a few, provide a broad market demand and driving force in terms of new development of both AI technologies and applications [61].

In **Customized Manufacturing** (CM) factories, automated devices can potentially make decisions with reduced human interventions. Technologies such as ML and computer vision are enablers of cognitive capabilities, learning, and reasoning (e.g., analysis of order quantities, lead time, faults, errors, and downtime). Product defects and process anomalies can be identified using computer vision and foreign object detection.

Scheduled maintenance ensures that the equipment is in the best state. Sensors installed on a production line collect data for analysis with ML algorithms, including convolutional neural networks.

AI technologies still have their limitations when they are formally adopted in real-world manufacturing scenarios. On the one hand, AI and ML algorithms often have stringent requirements for computing facilities [62].

Today, information and communication technologies are the base of **smart manufacturing and intelligent systems** driven by AI are the core of CM. With the development of AI technologies, new theories, models, algorithms, and applications - towards simulating, extending, and enhancing human intelligence - are continuously developed [63, 64].

The AI-assisted CM is to construct smart manufacturing systems supported by cognitive computing, machine status sensing, real-time data analysis, and autonomous decision making. AI permeates through every link of CM value chains, such as design, production, management, and service [65, 66, 67, 68].

Finance

The powered technologies of AI are useful to predict cash flows, detect frauds, adjust credit scores, make investment decisions and manage risks.

Starting from risk management, fraud detection and prevention to credit decisions and financial advisory, everywhere the application of AI is undeniable. Artificial intelligence can be used to **analyze** the **customers' spending patterns** and their **regular financial activities** based upon which loan borrowing behavior can be predicted. The technologies are also helpful in the automation of work by which the financial firms can do the **stock market** prediction along with the sales forecasting [69].

The use of AI-powered technologies is very helpful for a finance firm to **manage** its **daily records** and **transactions**. The continuous increment of the data and transaction history from a huge population is quite difficult to manage by using the typical manual process [70].

AI brought automated computerized **digital transaction** processes where individuals can get financial services at any time. Moreover, in case any issues arise related to financial transactions or relevant to them, AI also offers chatbots and virtual assistants to provide the customer support at any time [71].

AI also helps to **reduce the repetitive mundane activities** that usually need to be done in the financial sectors. In addition to this, the combination of machine learning and artificial intelligence effectively makes time-consuming work faster by using the same data to fill similar blank boxes [72].

AI-based powered technology offers a **cybersecurity-based encrypted environment** through which the user can easily transact with full safety. To turn this concept into reality, the AI-based algorithm boosts company security by analyzing and determining the pattern of the normal data and trends that effectively alter companies on an immediate basis when discrepancies and unusual activities are detected [73].

Government

As people become dissatisfied with digital government products, artificial intelligence (AI) might help **close the interaction and service delivery gap** while also **enhancing public participation**. However, it should not be employed in governance just because it is a fresh and exciting technological advancement. To be effective in their jobs, government officials need to be prepared to deal with challenges that arise. Artificial intelligence (AI) should be one tool in a toolbox that may be used to deal with a particular situation [74].

Adding the advantages of artificial intelligence to government entities should not necessitate the creation of new systems. Although most of the advancement in artificial intelligence has come from early government-funded research, governments may also benefit from the gains being made by corporations and entrepreneurs in the field.

Chatbots and other types of AI solutions (e.g. machine learning algorithms, process automation and image recognition software) can significantly **reduce the administrative burden** of public organizations and advance the communication between government and citizens within the provision of public services, which has been problematic for a long time [75].

Artificial intelligence is being used by government entities all around the globe, from small towns in the United States to whole countries like Japan, to **improve the quality of citizen services**. As a result, although future uses of artificial intelligence in governance are constrained by several factors, including government resources and the inventiveness and trust of individuals in their governments, the most obvious and directly favorable possibilities are those where AI can reduce administrative burdens, assist with the allocation of resources challenges, and perform extremely difficult tasks [76].

New York Municipal intends to collaborate with IBM's artificial intelligence platform, Watson, to develop a **new customer-management system** that will reduce the time and effort required to respond to queries and complaints about city services sent via the city's 311 portal. Approximately 65 percent of queries are answered on municipal websites, and the app is utilized to handle those issues [77].

AI is increasingly being applied to **citizen inquiries** and **information initiatives**. They may not fully use AI and machine learning's powers, but this shows how AI is transforming this sort of employment for the future. To better comprehend citizen feedback and queries, AI may be employed in the future for sentiment analysis of demands and conversations.

In certain circumstances, artificial intelligence (AI) might also be used to **assist with in-person public interaction** and service delivery. Because of AI's ability to expand citizen access to real-time replies, it may even be utilized in **ordinary occupations** to create and fill out paperwork [78].

Retail

Consumer shopping behavior data is used to create a database, which is then trained repeatedly to **improve recognition accuracy** and **predict what consumers want** so that an enterprise can deliver a more enjoyable shopping experience [79].

New retail could **create value-added products and services** using big data analysis and cloud computing. Innovative technology applications can help retailers predict consumer needs and recommend products based on their preferences. It also can help suppliers optimize inventory management, logistics, and transportation [80].

By applying AI-based technologies and services to retail shopping, consumers can have a **smoother shopping experience**. In most cases, services and technology applications are still in the pre-purchase phase, concentrating on collecting consumer shopping data. The purchase phase is less diverse, but electronic payment and self-checkout are increasingly popular, with over 60 billion dollars in transactions per year [81, 82].

The use of AI in **supplier management** enables suppliers to adapt production plans to orders from the retail and to identify suitable suppliers for the retail. This makes supplier selection more objective and subject to both qualitative and quantitative performance indicators with AI-based decision models that take various factors into account and go beyond the traditional role of pure price reduction [83, 84, 85].

In today's society, both stationary retail and e-commerce are highly dynamic and as the market is changing rapidly, prices are too. Complex analyses and decisions in price management can be carried out with intelligent and self-learning solutions. **Dynamic pricing**, as a new development, is a pricing strategy in which companies adjust prices for products or services in real-time based on the current market demand. This is a model that calculates prices using automatic algorithms, as human decisions would not cope with the required speed and amount of data to be considered [72].

With improvements at the inventory level, it is also possible to **reduce unsold goods** and enhance the cash flow (as inventory costs cannot be invested in new products). Another field of application here is the **optimization of shelf space** within the stores and the determination of an optimal base-stock level so that the shelf always looks filled up and replenishment is not needed too often. AI algorithms can also optimize the order and delivery to individual customers [86, 87, 88].

AI systems can interactively **monitor and optimize all logistics processes** and even the product's characteristics (size, shape and weight) and order demand factors can be considered during logistics strategy formulation [89].

The retail chain uses AI during replenishment tasks to reduce waste. On the customer side, the **optimization of the fulfillment processes** within the supply chain can be implemented with AI technologies [90].

A major application of AI within this task set is the replacement or **automation of activities at the point of sales** (POS). AI applications related to serving customers have particularly been developed for POS digitization, automation and advertising. Here, solutions are based on AI in connection with displays, language assistants and robots [91].

In terms of interactive displays and digital signage, AI is used to **advertise products** from a store in a targeted manner, related to age, gender, emotions or objects detected in real-time on the body of a customer [92].

Lastly, another major area of implementation for **fraud detection** systems based on AI is credit card and payment fraud. AI-based **video surveillance systems** with image processing and recognition algorithms register at real-time speed when goods remain in the trolley and signal the checkout staff or the supervisor [93, 94].

Transportation

To efficiently create and manage a sustainable transport system, technology could be of immense support. With urban areas struggling with traffic congestion, AI solutions have emerged in accessing real-time information from vehicles for traffic management, and utilizing mobility on demand in trip planning through a single user interface [95].

AI in its current form can solve problems in real time transport thus managing design, operation, schedule and administration of logistical systems and freight transport. AI techniques allow the utilization of these **applications for the entire transportation management** – vehicle, driver, infrastructure and how these components dynamically offer transport services [96].

Since the past couple of decades, due to the emergence of smart technologies, various information systems for logistics, routing, mapping and planning have been developed. These systems have provided increased data processing capabilities to better plan the transportation process leading to **Intelligent Transportation Systems** (ITS) [97].

Building ITS into the transport systems has ensured increased performance due to information acquisition, exchange and integration across vehicles, city infrastructure and other related activities. It is observed that ITS supports the decision-making process for the city authorities and vehicle users.

To build an effective and efficient mobility ecosystem in a city, a holistic approach to mobility management is required. **Connected vehicles** send data in real time, thus generating immense amounts of data. With transportation demands continuing to increase, data growth through devices also grows; thus, creating a need for smarter management of road traffic [95].

Connected cars are capable of accessing the Internet through smart devices and are also capable of communicating with other cars and infrastructure. They draw real-time data from multiple sources supporting drivers through stressful operations during driving. These cars ensure safety and reliability [98].

Safe integration of AI-based decision-making, traffic management, routing, transportation network services and other mobility optimization tools are other possibilities for efficient traffic management.

Pattern recognition is used with image processing for **automatic incident detection** and identifying cracks in pavements or bridge structures. The clustering technique is used for identifying specific classes of drivers based on driver behavior [99].

2.1.3 Natural Language Processing

As a branch of artificial intelligence, **Natural Language Processing** (NLP), uses machine learning to process and interpret text and data. *Natural language recognition* and *natural language generation* are types of NLP [100].

Natural language processing applications are used to derive insights from unstructured text-based data and give the user access to extracted information to generate a new understanding of that data. Some examples of the usage of NLP in real life applications are the following:

- **Content classification**: it can be used to classify documents by common entities or domain-specific customized entities;
- **Customer sentiment**: entity analysis can be used to find and label fields within documents to understand customer opinions better;
- **Document analysis**: here, custom entity extraction is used to identify domain-specific entities without having to spend time on manual analysis;
- **Healthcare**: it helps to improve clinical documentation and automate registry reporting, to accelerate clinical trials;
- **Receipt and invoice understanding**: even here the technique of extracting entities to identify common entries in receipts is useful, since it can help understand better the relationships between requests and payments;
- **Trend-spotting**: aggregating news with text helps marketers extract relevant content about their brands from online news or articles.

2.1.4 Machine Learning and Deep Learning

Machine learning is a sub-discipline of AI, while **deep learning** is a sub-discipline of machine learning.

Both algorithms use **neural networks** to *learn* from huge amounts of data. These neural networks are programmatic structures modeled after the decision-making processes of the human brain. They consist of layers of interconnected nodes that extract features from the data and make predictions about what the data represents (the interconnections and the distribution of nodes inside each neural network resemble the human neural system, hence the name).

The main difference is in the type of neural network used by each discipline:

- Machine learning enables *supervised learning*: the networks used here have a specific layout, with an input layer, one or two "hidden" layers and an output layer. In these cases, the data needs to be structured or labeled by human experts, so that the algorithm can efficiently extract features from the data;

- Vice versa, deep learning enables *unsupervised learning*: the networks used are *deep neural networks*, composed of an input layer, three or more (usually hundreds) hidden layers and an output layout. The "unsupervised" part is given by the fact that the extraction of features is done automatically, without requiring human intervention. This essentially enables *machine learning at scale*.

2.2 OpenAI and ChatGPT

The world of AI is growing more and more and many developers are taking advantage of its capabilities to automate operations, such as writing a script in a particular programming language or checking if the code written by a user presents some flaws or not.

Even the less skilled users from a technological point of view have a general knowledge of what ChatGPT is, but many need to be aware of the details of this tool. In the following sections, there will be a basic explanation of what ChatGPT is and who is behind this powerful tool: OpenAI.

2.2.1 OpenAI

One of the most famous artificial intelligence research and deployment companies is **OpenAI** [101], founded in the United States in 2015, with the mission to ensure that artificial general intelligence benefits all of humanity.

Figure 2.1. OpenAI logo

It offers many **models** that can be used to generate text (ChatGPT, 2.2.2), turn audio into text (Whisper, [102]) and vice versa (TTS, [103]), generate images (DALL-E, [104]) or understand their content (Vision, [105]) and many more functions.

Along with the models, the newly developed **Assistant** is a tool that can be thought of as an *evolution* of the generative text interaction with the user: it can analyze the content of files uploaded to the Assistant UI and let the user build their own AI assistant.
In this project, this tool will constitute a turning point in the static analysis of unit tests, since its usage accelerated incredibly the analysis itself (3.2.3).

Since 2022 this company is gaining more and more attention thanks to the release of what will be their most famous and used product: ChatGPT.

2.2.2 ChatGPT

OpenAI's text generation models (often called **generative pre-trained transformers** or **large language models**) have been trained to understand natural language, code and images. The models provide text outputs in response to their inputs. The inputs to these models are also referred to as **prompts**. Designing a prompt is essentially how the user *programs* an LLM model, usually by providing instructions or some examples of how to complete a task [106].

Developed by OpenAI, **ChatGPT** [107] is a chatbot based on a large language model (LLM) that enables users to refine and steer a conversation towards a desired length, format, style, level of detail and language. This model interacts with the user conversationally: the dialogue format makes it possible for ChatGPT to answer follow-up questions, admit its mistakes, challenge incorrect premises and reject inappropriate requests.

By itself, ChatGPT offers many models to choose from, each one with its characteristics; these can be grouped into two categories (Figure 2.3): **GPT 3.5** and **GPT 4**.

The former [108] can be used by everyone freely, through the provided UI on a web browser, and will answer pretty rapidly to the user's request; it is also supported better for specific features, such as fine-tuning (2.3.1) or chat completions.

The major downside of this model is that the retrieved results may not be accurate and may be easily influenced by the way the user writes the request.

Another problem is given by the *hallucinations*, the typical behaviour of an LLM in which it makes up information when the model does not have enough data to solve the user's problem.

On the other hand, as analyzed in its technical report [109], the latter offers a more accurate result and is more capable of carefully following complex instructions (while gpt-3.5 models are more likely to follow just one part of a complex multi-part instruction), thanks to its wider knowledge and improved reasoning abilities; generally, performs better on a wide range of evaluations.

It also outperforms the previous large language models and most state-of-the-art systems; it also outperforms existing models both in cases in which the request is in English or other languages.

The downside is that it returns outputs with higher latency (due to its larger context window) and costs much more per token.

There are various models in these two categories, each one differing for context window, training data aging and other specific characteristics. For example, gpt-4-vision-preview is the model able to understand the content of images, while gpt-4-0125-preview's main characteristic is the reduction of cases of "laziness" where the model doesn't complete a task[1]. Overall,

[1]as of April 2024

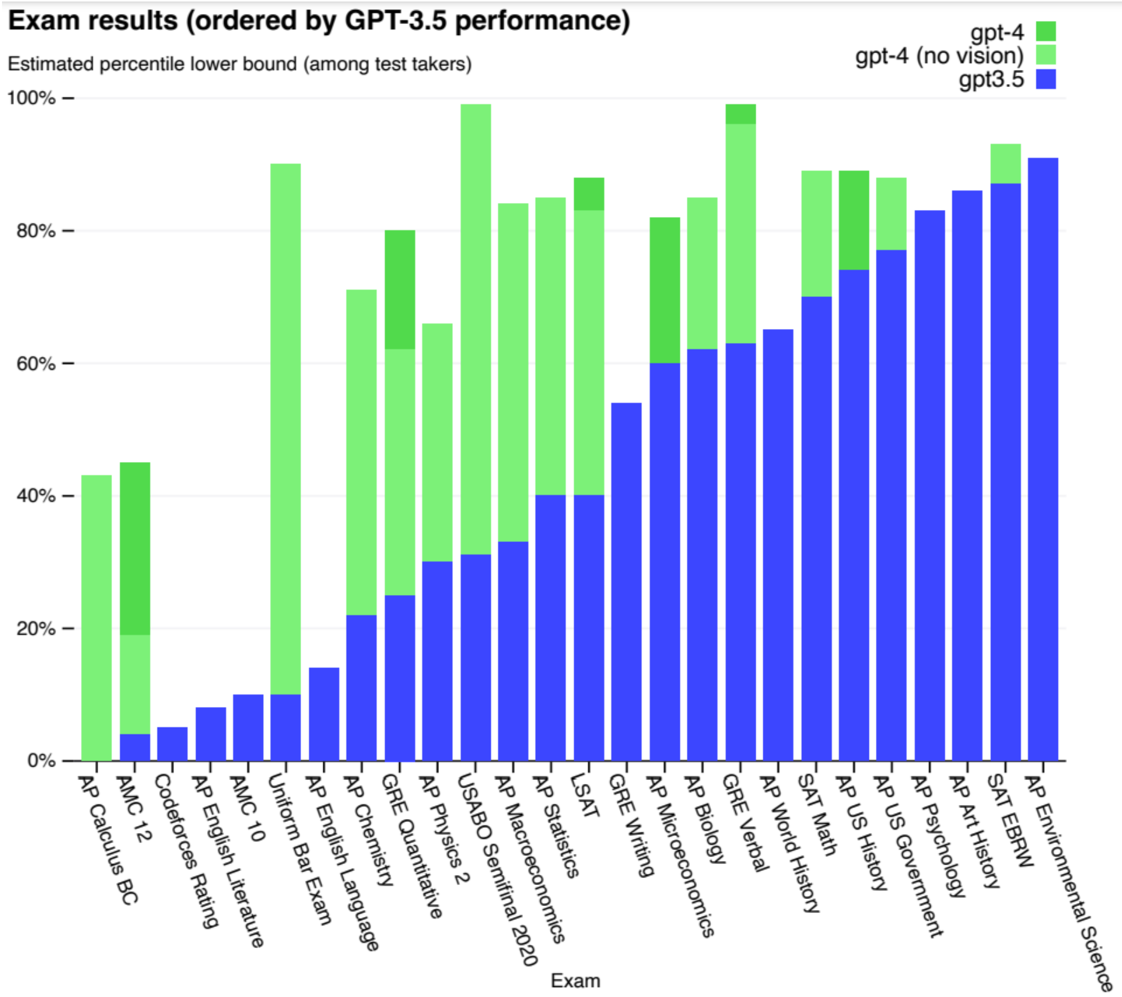


Figure 2.2. GPT performance on academic and professional exams [109]

regarding text generation models, OpenAI recommends choosing the model that performs the best depending on the complexity of the tasks it is being used for. To do so, the user can perform some tests in the **OpenAI Playground**, to investigate which models provide the best price-performance trade-off for the needed usage.

2.3 Fine-Tuning and Few-Shots

There are many strategies to manage a conversation with ChatGPT in order to obtain the desired result, but they have a common factor: the more specific

Features	GPT-3.5	GPT-4
Parameters	175 billion	> 1 trillion
Modality	Text only	Text and images
Performance	Poor on complex tasks	Human-level on various benchmarks
Accuracy	Prone hallucinations and errors	More reliable and factual (still hallucinates, but not as much)
Type of learning methods	Unsupervised LLM	Supervised LLM
Context window (latest models)	16,385 tokens (gpt-3.5-turbo-0125)	128,000 tokens (gpt-4-turbo)
Training data (latest model)	Up to September 2021 (gpt-3.5-turbo-0125)	Up to December 2023 (gpt-4-turbo)
# words processed at once	Up to 3,000	Up to 24,000

Figure 2.3. ChatGPT 3.5 and 4 - overview of main differences

is the user request, the better is the result provided by ChatGPT. However, most of the times the users may be too lazy to write all the specifications, or there is a really high probability that something may be missed: the user may consider useless some information that the model doesn't know and, if provided, it would have changed the result significantly.

Another problem is given by the **context window** of ChatGPT (that can be thought of as its *memory*): the information written in the prompt is collected into "tokens", that may be represented by some group of words or just one word. Most of the time, the information that the user has to give to the model is too much and cannot be contained into the context window of the specific model efficiently: by summarizing it, some important details might be missed and the result will not be the one desired by the user.

To solve this problem, two strategies can be used: fine-tuning and few-shots.

2.3.1 Fine-Tuning

With the technique of **fine-tuning** [110], the model is trained with some data examples, to have results specific to what is desired. Fine-tuning can be applied to the desired model only if an API Key provided by the specific AI development company (in this case, OpenAI) is available.

In this case, the user must create some **training data** and **validation data**. The first is used to train the model, while the second is used to see a second benchmark produced during fine-tuning: not just how much the learning on the training set has progressed, but how well similar questions are inferred.

It's important to note that OpenAI models only accept the JSONL format for both the training file and validation file: this means that the data have to follow the JSON line format.

Based on the type of models, the messages can be either a conversation between *system-user-assistant* entities or a *prompt-completion* pair format. Since for GPT models the first kind of messages is the supported one (while the other is for models like *babbage* or *davinci*, which are not analyzed in this thesis), it will be analyzed in detail (Figure 2.4).

As previously mentioned, in the fine-tuning conversation there are three entities:

- **system**: it explains what the model embodies and gives a general context to the model of how should behave;
- **user**: this is the request done by the user;
- **assistant**: this is how the model should answer according to the request done by the user.

For each message "triplet", it is important that the amount of tokens doesn't go over the maximum possible; since this number differs among the models, it is important to check which model has the right amount of tokens available for the training.

```json
{
    "messages": [
        {
            "role": "system",
            "content": "Marv is a factual chatbot that is also sarcastic."
        },
        {
            "role": "user",
            "content": "What's the capital of France?"
        },
        {
            "role": "assistant",
            "content": "Paris, as if everyone doesn't know that already."
        }
    ]
}
```

Figure 2.4. Conversational chat format example

In case none of them satisfies this request, most probably the training data is too long and needs to be shortened, to follow the OpenAI indications; however, at the same time, the amount of information that is kept should be enough to make sure that the context of the model can be considered acceptable to train it.

Once the model is trained, the output model is retrieved, with all its data, and it can be used for the specific goal (Figures 2.5 and 2.6).

As stated by Latif et al. [111] in their study on the ChatGPT behavior with automatic scoring, fine-tuned GPT models are more suited to tasks like text completion, response evaluation, or open-ended queries because of their **autoregressive nature**, which excels in sequence formation. GPT's already strong generating capacities will be strengthened by fine-tuning it for specific tasks or domains.

2.3.2 Few-Shots

As opposed to the previous strategy, the **few-shots** methodology aims to give context to the model "on the fly": the model is not fine-tuned through the OpenAI UI, instead it receives some example messages called *shots* (each pair of user-assistant messages is one shot). Using the OpenAI Playground,

Figure 2.5. OpenAI Fine-tuning UI (1)

the user can insert the system message to give a minimum context to the model (if necessary) and a little interaction between the user and the assistant, so that the model understands how the answer should be built based on the format of the input.

As stated by Cao et al. [112], few-shot classification is the task of predicting the category of an example from a few labeled examples. The number of labeled examples per category is called the number of shots (or shot number). Recent works tackle this task through **meta-learning**, where a meta-learner extracts information from observed tasks during meta-training to quickly adapt to new tasks during meta-testing.

In the following example, the model is asked to tell if the affirmation of the user is true or false. First, the user writes two examples, to show how

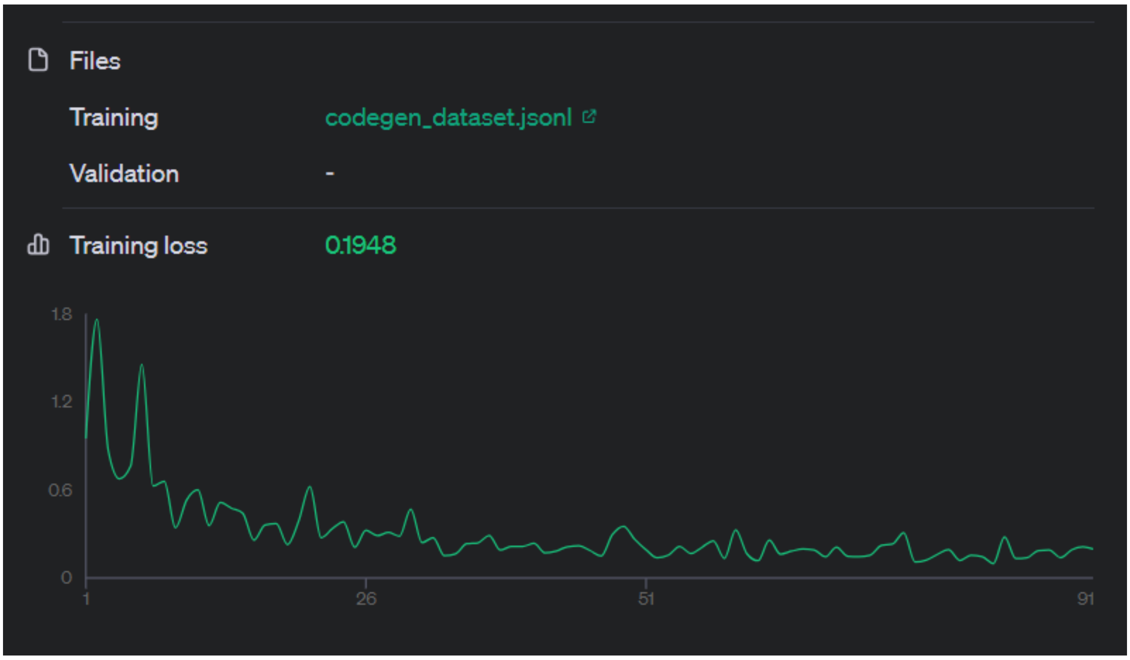


Figure 2.6. OpenAI Fine-tuning UI (2)

the model should answer the request; in this example, the model shouldn't write all the calculations done and a detailed explanation of what it did to come to the result (Figure 2.7). Then, the user writes the actual request, which is the last "user" message.

Once the user clicks the "Submit" button, the model retrieves the answer; this answer is written in the same way as desired by the user (Figure 2.8).

This method is useful in case there are not enough examples to fine-tune a model (a fine-tuned model can be achieved if it receives at least 10 examples) and if the examples need to be edited (in case of fine-tuning, the model needs to be trained again in case the examples are modified).

2.4 Software testing

One of the most important activities to do to make sure that the code written by someone is correct is to perform some tests on it. As stated by Hellmann et al. [113], beyond implications for the quality of the code and the effort that will be required to fix bugs post-release, the effectiveness of testing in an agile context can determine the outcome of an agile software development effort.

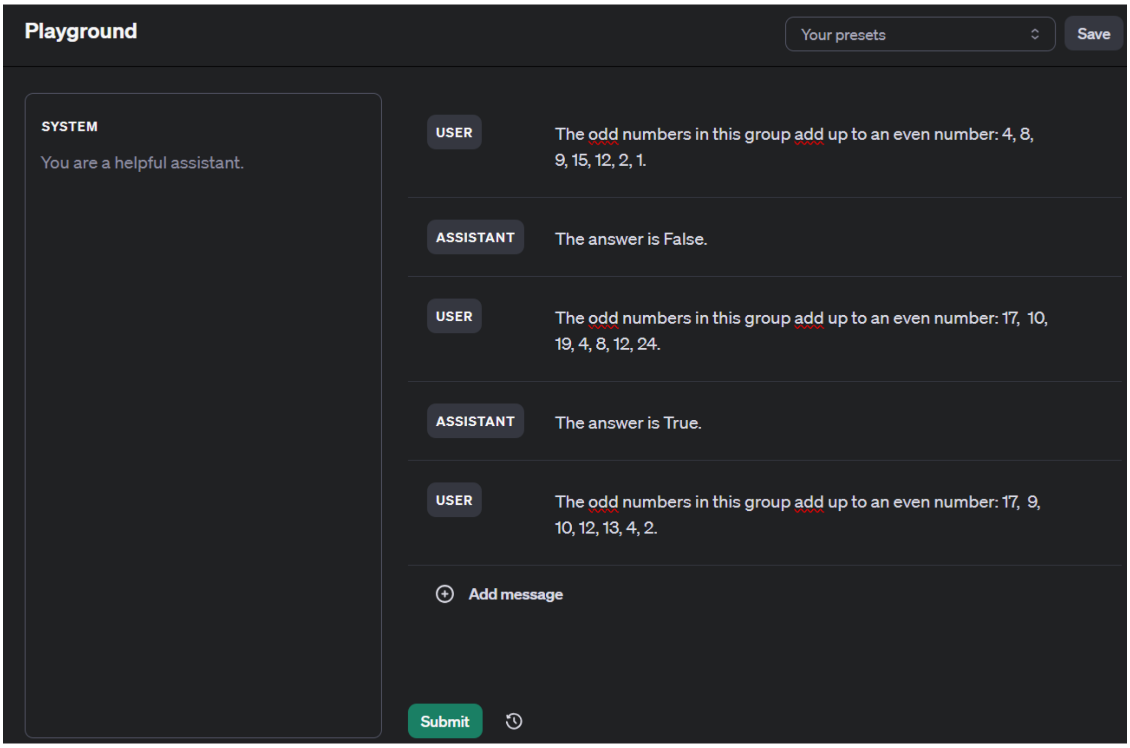


Figure 2.7. OpenAI Playground: 2-shots example inserted by the user, UI before the model's answer

Every stage of the software life cycle involves testing, but the testing done at each level of software development is different and has different objectives [114]:

- **Unit Testing** is done at the lowest level. It tests the basic unit of software, which is the smallest testable piece of software, and is often called "unit", "module", or "component" interchangeably;
- **Integration Testing** is performed when two or more tested units are combined into a larger structure. The test is often done on both the interfaces between the components and the larger structure being constructed, if its quality property cannot be assessed from its components;
- **System Testing** tends to affirm the end-to-end quality of the entire system. System test is often based on the functional/requirement specification of the system. Non-functional quality attributes, such as reliability, security, and maintainability, are also checked;

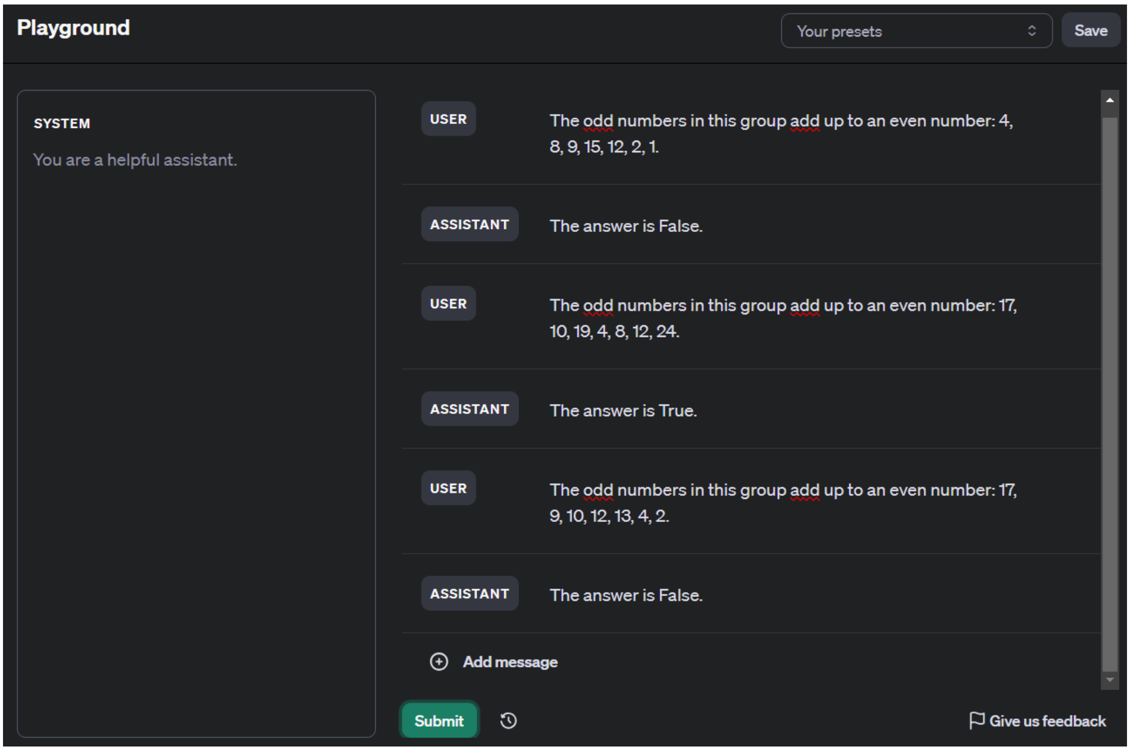


Figure 2.8. OpenAI Playground: 2-shots example inserted by the user, UI after the model's answer

- **Acceptance Testing** is done when the completed system is handed over from the developers to the customers or users. The purpose of acceptance testing is rather to give confidence that the system is working than to find errors;

In this thesis, the focus is on unit tests.

2.4.1 Unit testing

Unit testing is the process where the developer tests the smallest functional parts of code, called *units*, that can be logically isolated in a system (a function, a subroutine, a method or a property) [115, 116]. As stated by Michael Feathers [117], such tests are not tests if they rely on external systems: "If it talks to the database, it talks across the network, it touches the file system, it requires system configuration, or it can't be run at the same time as any other test.".

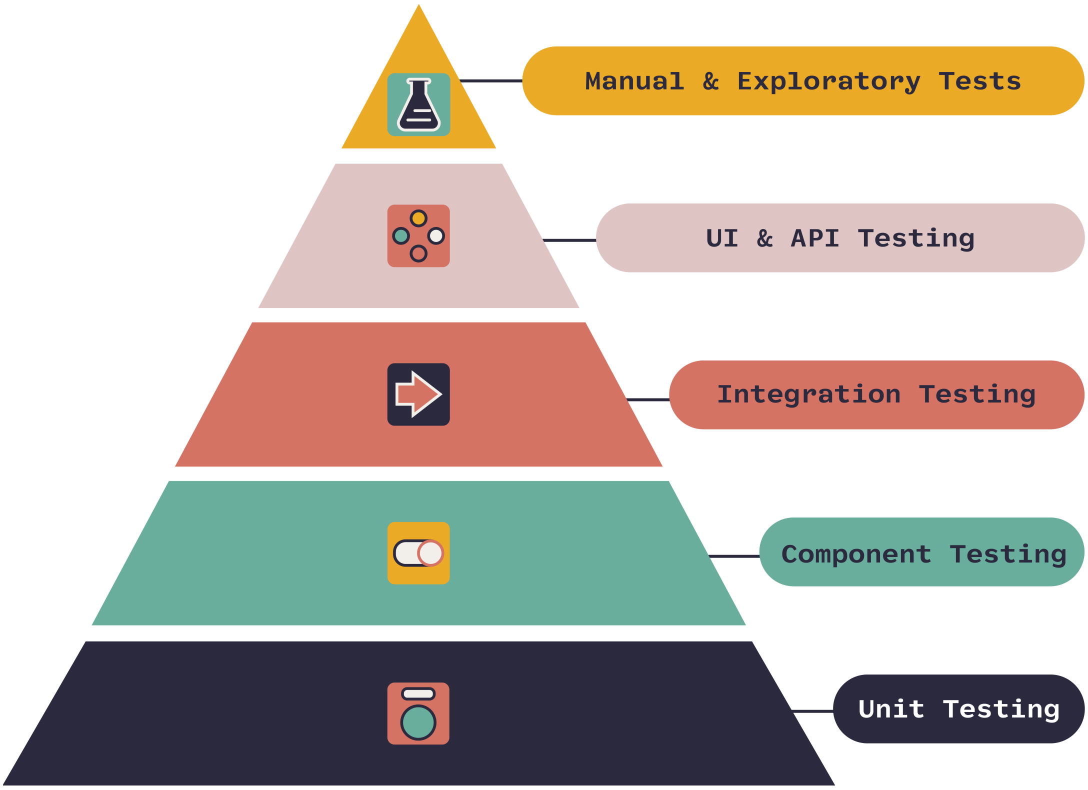


Figure 2.9. Agile testing pyramid

The growth of object-oriented programming has influenced the way programmers approach software testing. Being predominantly bottom-up, it is natural that object-oriented programming favors a similar testing methodology that focuses on classes. A unit test exercises a “unit” of code in isolation and compares actual with expected results. In Java, the unit is usually a class. Unit tests invoke one or more methods from a class to produce observable results that are verified automatically [118].

Software testing helps ensure code quality, and it’s an integral part of software development. The best practice is to write software as small, functional units and then write a test for each code unit. At first, each test can be written as code; then, every time there is a change in the software code, the tests run automatically. This way, if a test fails, the developer can quickly isolate the area of the code that has the bug or error. Unit testing enforces modular thinking paradigms and improves test coverage and quality.

Different from integration tests, in which the code is being tested in its entirety to make sure that the various parts of the software system interact with each other correctly, unit tests are focused on a specific portion, to make sure it is written correctly. It is also a good practice to write unit tests for error cases (for example, if some parameter is of the wrong type or it is empty), to make sure that also these cases are considered.

These multiple tests related to a single block of code are called **test cases**: they should cover the full expected behaviour of the code block (most of the times, it is not necessary to define the full set of test cases, but it is a good practice to do so).

Unit testing benefits software development in many ways: it isolates bugs efficiently (smaller tests give a more granular view to the developer on how the code performs) and can be considered as a form of documentation. For the latter question, the tests can be analyzed to understand how the related code should behave when it runs.

They are also used to do **test-driven development**, a pragmatic methodology that takes a meticulous approach to build a product using continual testing and revision: to do so, developers build tests to check the functional requirement of a piece of software before building the full code itself. Another strategy is **continuous integration and continuous development** (CI/CD), one of the core activities in the application of DevOps to software development: any change to the code is automatically integrated into the wider codebase, run through automated testing and deployed in case of successful tests.

Some best practices with unit tests consist of the usage of a unit test framework (for Java, the most common and supported one is **JUnit**, recently upgraded to the fifth version [119]), automate unit testing and implement unit testing thoroughly (it should be as important as code development).

2.4.2 Other types of software testing

After focusing specifically on unit tests, a brief analysis [120] of other tests must be done, to understand how deep can an application be controlled to solve its possible bugs:

- **Integration tests**: these tests verify that different modules or services used by the application work well together. Most of the time, the interaction with the database is tested or they are used to make sure that microservices work together as expected. These types of tests are more expensive to run as they require multiple parts of the application to be up and running;

- **Functional tests**: these focus on the business requirements of an application. They only verify the output of an action and do not check the intermediate states of the system when performing that action. The difference with integration tests is that an integration test may simply verify that the user can query the database while a functional test would expect to get a specific value from the database as defined by the product requirements;

- **End-to-end tests**: End-to-end testing replicates a user behavior with the software in a complete application environment. It verifies that various user flows work as expected and can be as simple as loading a web page or much more complex scenarios (verifying email notifications, online payments, etc.). End-to-end tests are very useful, but they're expensive to perform and can be hard to maintain when they're automated. It is recommended to have a few key end-to-end tests and rely more on lower-level types of testing to be able to quickly identify breaking changes;

- **Acceptance testing**: these tests are formal tests that verify if a system satisfies business requirements. They require the entire application to be running while testing and focus on replicating user behaviors;

- **Performance testing**: performance tests evaluate how a system performs under a particular workload. These tests help to measure the reliability, speed, scalability and responsiveness of an application. For instance, a performance test can observe response times when executing a high number of requests, or determine how a system behaves with a significant amount of data. It is used to determine if an application meets performance requirements, locate bottlenecks, measure stability during peak traffic and more;

- **Smoke testing**: these are simple tests that check the basic functionality of an application. They are meant to be quick to execute and their goal is to assure the developer that the major features of your system are working as expected. Smoke tests can be useful right after a new build is made to decide whether or not you can run more expensive tests, or right after a deployment to make sure that the application is running properly in the newly deployed environment.

2.5 Spring Boot

One of the most known frameworks to create a stand-alone application in Java is **Spring Boot** [121, 122]. Spring Boot is a *convention-over-configuration* extension for the Spring Java platform intended to help minimize configuration concerns while creating Spring-based applications. Most of the applications can be preconfigured using the Spring team's "opinionated view" of the best configuration and use of the Spring platform and third-party libraries.

Figure 2.10. Spring Boot logo

It has many useful features:

- Create stand-alone Spring applications;
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files);
- Provide a radically faster and widely accessible getting-started experience for all Spring development;
- Be opinionated out of the box but get out of the way quickly as requirements start to diverge from the defaults;
- Automatically configure Spring and 3rd party libraries whenever possible;
- Provide a range of non-functional features that are common to large classes of projects (such as embedded servers, security, metrics, health checks, and externalized configuration);

- Absolutely no code generation (when not targeting native image) and no requirement for XML configuration.

Spring Boot does not require any special tools integration, so the user can use any IDE or text editor. Also, there is nothing special about a Spring Boot application, so the developer can run and debug a Spring Boot application as they would do with any other Java program. It explicitly supports both Maven and Gradle as building tools to support dependency management.

One of the many features of Spring Boot is the support of the **Model-View-Controller** (MVC) pattern (Figure 2.11), an architectural design pattern that separates an application into three interconnected components:

- **Model**: represents the application's data and business logic. It is responsible for managing the data and the rules that govern it;
- **View**: represents the presentation layer and is responsible for displaying the data to the user. It handles user interface elements and rendering;
- **Controller**: acts as an intermediary between the Model and View. It receives user input, processes it, and manages the flow of data between the Model and View.

This pattern has many advantages, such as separation of concerns, reusability, scalability and testability.

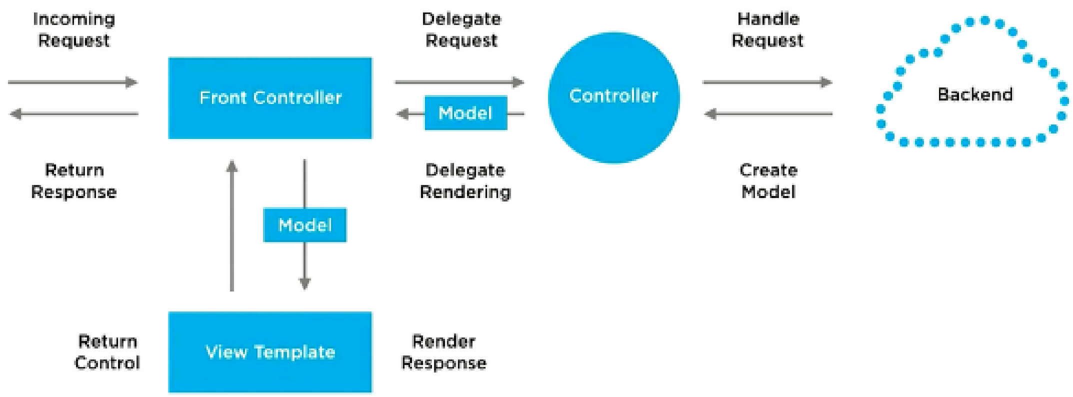


Figure 2.11. Spring Boot - MVC pattern

These are translated in Spring Boot in the following way:

- The Model is represented by Java objects, often known as **beans** or **entities**. These objects **encapsulate data and business logic**;
- In Spring Boot, the View corresponds to the **user interface**, the part the user employs to interact directly with the application;
- Spring Boot controllers handle user requests, **manage the interaction** between the Model and View and control the application's flow. These controllers are annotated with *@Controller* and handle the requests coming from the View, translate them in a format comprehensible by the Model and transfer them to the backend; then the responses are given back to the UI.

In this thesis, the developed Spring Boot application is really basic: it doesn't contain a View (since it was not needed), but it contains some model, service and controller classes, as explained further below (3.3).

2.6 GitHub

One of the most used platforms to store, share and work together with other users to write code is **GitHub** [123]. Using a *repository* and storing the code inside it lets the user track their activity efficiently, share their work, let others review the code and suggest how to improve it and, most importantly, securely collaborate on a project. **Collaborative working**, one of GitHub's fundamental features, is made possible by the open-source software, Git, upon which GitHub is built.

Figure 2.12. GitHub logo

Git is a version control system that tracks changes in files. It is particularly useful when a group of people make changes to the same files at the same time.

To do so, the user would create a branch off from the main copy of files that all the collaborators are working on, edit files independently and safely on their branch. Then Git intelligently merges the user changes back into the main copy of files and keeps track of changes made by all the users.

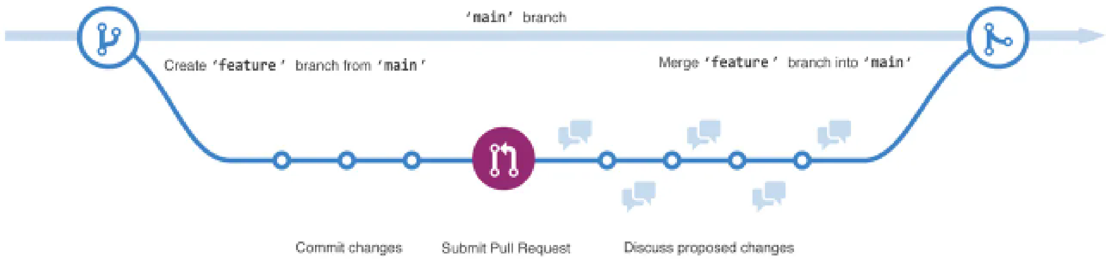


Figure 2.13. GitHub - basic workflow

GitHub and Git are **strictly related**: when the files are uploaded to GitHub, they are stored on a Git repository; when the user makes changes (or *commits*) to the files on GitHub, Git will automatically track and manage these changes.

Most of the users work on the files locally and then sync these local changes with the central remote repository on GitHub (Figures 2.13 and 2.14).

To sum up, Git figures out how to merge this flow of changes, while GitHub helps the developer manage the flows through various features (such as pull requests).

GitHub contains many features that will be used inside the project of this thesis and will be analyzed further in the next chapter: GitHub App, GitHub Webhooks (3.6) and GitHub Actions (3.7).

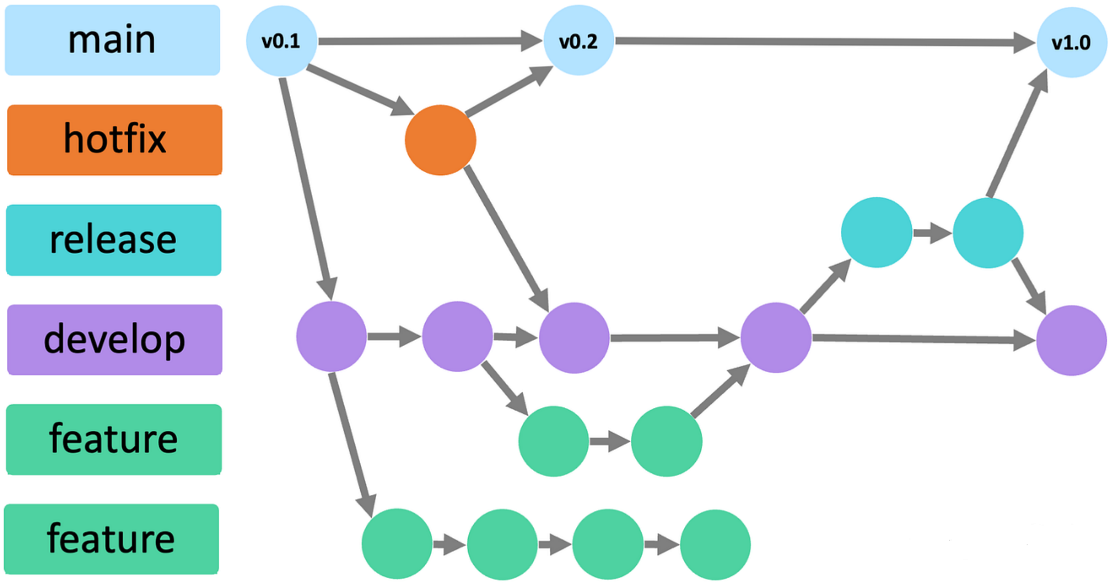


Figure 2.14. GitHub - articulate workflow

Chapter 3

Methodology

3.1 A brief explanation

This project embodies many complex concepts and its objective is quite ambitious: **make the AI do what a developer does daily**. At first, it might seem quite easy, since it is known that LLM models can answer correctly to requests about writing some code scripts.

What is not always considered is what is *behind* the job done by the developer:

- The written code must be **understandable** by everyone: for example, a new employee has to comprehend in the least amount of time possible what is going on in the script they are currently watching, and if it is not written understandably, this activity may take too much time;

- The written code should always follow the **clean code best practices**: coherence in variable and function names, keeping method short, avoiding useless repetitions, etc.;

- Every piece of written code should be **tested** before the merge in a collaborative environment, to discover bugs as soon as possible and fix them before provoking a waterfall effect on the whole project;

- It may be obvious, but the developer needs to **have the whole context** before approaching the environment, to avoid writing useless code scripts.

With this brief list, everyone can understand that this project is way more difficult than expected.

Despite this, if this challenging idea comes true, this would be the start of a new era for the AI world.

At first, the idea was to use AI in a double way: first, given the code, the artificial intelligence would have to write the correspondent unit tests; in the second phase, given the unit tests, the AI would have to generate the code for which these tests were made.

To do so, the first thing that has been done is an **analysis of the various GPT models** (3.2), to understand in practice the capabilities of each one of them: how quickly they generate the tests, how deep in detail the system prompts should be to give them enough context, how much code is covered by the generated test classes and so on.

To have an objective result, the static analysis method is the one used to examine in detail these tests, as it will be explained in the following section.

Since this analysis took some time and the internship had a specific time frame, after discussing with the tutor, the conclusion was that the second part of the thesis (the generation of the code starting from human-written unit tests) was more useful from a statistical point of view instead of a practical one, so it was *substituted* with an objective probably more challenging but also more fulfilling: the whole code generation **starting from a GitHub issue**.

Actually, the idea was to consider Jira issues [124], since it is the ticket and project monitoring software used in the company, but there were some problems with the integrations between this software and the AI, so this idea was archived.

Once the project had changed its course, the next dilemma to solve was understanding **how could the LLM model obtain** the same **knowledge of an entire code base** as a developer would do (3.4): as seen in the previous chapter, different models have different context windows and remembering every content in a GitHub repository, understanding the various interactions and the reasons behind them is not doable only with the basic models.

To do so, there is a feature that includes the interaction with AI and a database containing the needed information to give context to a model: RAGs.

The **Retrieval Augmented Generation** can be described as a technique for enhancing the accuracy and reliability of generative AI models with facts fetched from external resources. The obtained models combine pre-trained parametric and non-parametric memory for language generation [125].

Even with these, there was a problem: the world of RAGs is quite distributed and there were some doubts regarding what would be the best provider to use for this project, without spending too much money. One example was Azure AI Search, given that the company was associated with OpenAI and an OpenAI API Key was available to be used for this project, but this aspect was not examined deeply.

By doing some research, it was discovered what will be the turning point of this application: the **LangChain** framework (3.5). This framework has direct integration with GitHub (which is where the code would have been uploaded) and with LLMs, providing an efficient way to link these two far concepts.

In some way, the RAG is integrated into LangChain: the bridge built to link the GitHub repository and the LLM is given by a GitHub Application.

Given this, the focus has now changed to *where* the LLM will operate. The test subject of this project is a **Spring Boot application** written in Java 17 and using Gradle 8.6, in which there is a fictional university, with students, professors and courses as models (3.3). A bit of code has been previously written, so that the LLM model has a basic context on how it should be formatted and how the various classes interact with each other.

Once the code is written and inserted in a GitHub repository, the automation of the application is enhanced with the usage of **GitHub Webhooks** (3.6) and **GitHub Actions** (3.7), two features useful to interact with a repository: the first will receive notifications when something happens on the repository and will trigger some actions, the second will automate operations such as building and deployment.

Lastly, to simulate a real life environment, a **GitHub branch protection rule** (3.8) has been added, to prevent irresponsible merges of code into the main branch.

3.2 Analysis of different GPT models

As written before (2.2.2), there are many models to choose from, each one with different characteristics (context window length, training data, etc).

Before jumping into action, it is important to decide which one of the various models offered by OpenAI is the best to use to do the unit test in Java in a good way. To do so, the **static analysis** comes to help: it is the best strategy to see how the unit test performs given the Java class and some parameters.

3.2.1 Analyzed models

The analysis involved the following 8 models from OpenAI [126]:

- **gpt-3.5-turbo**: The latest GPT-3.5 Turbo model with higher accuracy at responding in requested formats. Returns a maximum of 4,096 output tokens. Its context window is 16,385 tokens long and the training data is up to September 2021;

- **gpt-3.5-turbo-16k**: (*legacy*) Snapshot of gpt-3.5-16k-turbo from June 13th 2023. Its context window is 16,385 tokens long and the training data is up to September 2021;

- **gpt-3.5-turbo-0125**: The latest GPT-3.5 Turbo model with higher accuracy at responding in requested formats. Returns a maximum of 4,096 output tokens. Its context window is 16,385 tokens long and the training data is up to September 2021;

- **gpt-3.5-turbo-0613** (*fine-tuned*): (*legacy*) Snapshot of gpt-3.5-turbo from June 13th 2023. Its context window is 4,096 tokens long and the training data is up to September 2021. This model has been trained with 10 examples;

- **gpt-3.5-turbo-1106_10** (*fine-tuned*): GPT-3.5 Turbo model with improved instruction following, JSON mode, reproducible outputs, parallel function calling, and more. Returns a maximum of 4,096 output tokens. Its context window is 16,385 tokens long and the training data is up to September 2021. This model has been trained with 10 examples;

- **gpt-3.5-turbo-1106_30** (*fine-tuned*): Same as above, but it has been trained with 30 examples;

- **gpt-4**: Snapshot of gpt-4 from June 13th 2023 with improved function calling support. Its context window is 8,192 tokens long and the training data is up to September 2021;

- **gpt-4-turbo-preview**: GPT-4 Turbo preview model intended to reduce cases of "laziness" where the model doesn't complete a task. Returns a maximum of 4,096 output tokens. Its context window is 128,000 tokens long and the training data is up to December 2023.

At the time of training[1], most of the models now marked as *legacy* were considered up to date, so they were the only ones available (for both fine-tuning and few-shots).

3.2.2 Static analysis parameters

The following parameters are the ones used in the static analysis. Each one of them has a different **impact** on the unit test (in terms of relevance) that can be *low*, *medium* or *high*.

- **Cyclomatic complexity** (*medium*): it measures the complexity of a test method based on the number of linearly independent paths through the code. Higher values indicate more complex and potentially harder-to-maintain tests;
- **Cognitive complexity** (*high*): it measures how difficult it is for humans to read and understand a test method, taking into account factors like nesting, recursion, and the readability of control structures;
- **Lines of code** (*medium*): the total number of lines in a test class or method. Can indicate the potential for refactoring if the test is excessively long;
- **Number of methods** (*medium*): high counts can indicate a test class is testing too many functionalities or could benefit from being split into multiple classes;
- **Assertion density** (*high*): the ratio of assertion statements to non-assertion statements. It helps evaluate the focus of the test on actual testing;
- **Arrange-Act-Assert (AAA) Sequence Compliance** (*high*): it measures adherence to the AAA pattern, which helps maintain test structure clarity;

[1] February-March 2024

- **Test smells** (*high*): useful to identify patterns in test code that suggest potential issues, such as Flaky Test, Eager Test, Lazy Test, and Mystery Guest. Tools can detect these smells, indicating areas for improvement;
- **Duplicated code** (*high*): it identifies repeated code blocks within test classes. Duplicates can make maintenance harder and might suggest a need for parameterized tests or utility methods;
- **Magic numbers** (*medium*): the use of hard-coded values in tests without clear explanation. Using named constants can make tests more readable and maintainable;
- **Code coverage** (*high*): it measures the extent to which the source code is executed by the tests. Commonly includes Line Coverage, Branch Coverage, and Path Coverage;
- **Mutation coverage** (*high*): it assesses the quality of the tests by modifying (mutating) the production code in small ways and checking if the tests detect the changes. High mutation coverage suggests effective tests;
- **Comment-to-code ratio** (*low*): the proportion of comments within the test code, which can indicate the maintainability and understandability of tests;
- **Test code consistency** (*high*): the uniformity of coding style, naming conventions, and structure across all test cases, improving readability and maintainability.

3.2.3 Static analysis methodology

The analysis has been divided into 5 cases:

- **Case 1**: pure interaction with the model with a small Java class;
- **Case 2**: pure interaction with a bigger Java class, that has more than one constructor, many methods and many exceptions considered;
- **Case 3**: few-shots method, with 1 shot (one *user-assistant* interaction before the actual request of the user);
- **Case 4**: few-shots method, with 3 shots;
- **Case 5**: few-shots method, with 5 shots.

All these interactions have been done on the OpenAI Playground, which allows to manage the interactions in an easy and user-friendly way and allows changing the model in a faster way.

The various classes have been saved in different Java files and, once all of them have been generated, the OpenAI Assistant performed the analysis.

Since the Assistant can check the content of files, the Java class containing the code to be tested and the related generated unit test by the model are uploaded, along with a PDF file in which there is a description of the parameters to be considered to perform the static analysis.

Once everything had been analyzed, the Assistant gave a percentage rating for each metric and an overall score on how good the test generated by the model was. The only two parameters that have been checked by hand are Code Coverage and Mutation Coverage.

For the Code Coverage, the software that has been used is JaCoCo [127], while for the Mutation Coverage, it has been used PITest [128]. Both these different types of coverage can easily be analyzed thanks to the user interface provided once the correspondent command is run (Figure 3.1 and Figure 3.2).

JaCoCo's mission is to provide the standard technology for code coverage in Java VM-based environments: the focus is providing a lightweight, flexible and well-documented library for integration for various build and development tools. It was developed to have a code coverage technology that could be widely used in various contexts, since most of the preexisting technologies were fit for a specific Java tool and did not offer a documented API that may have allowed embedding in different contexts.

On the other hand, **PITest** (also known as simply *PIT*) is a state-of-the-art mutation testing system, providing gold standard test coverage for Java and the JVM. It's fast, scalable and integrates with modern test and build tooling.

PIT runs the unit tests against automatically modified versions of the application code. When the application code changes, it should produce different results and cause the unit tests to fail. If a unit test does not fail in this situation, it may indicate an issue with the test suite.

Traditional test coverage measures only which code is *executed* by the tests. It does *not* check that these tests can *detect faults* in the executed code.

demo > com.coverage

com.coverage

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Contact		76%		50%	11	27	13	82	0	16	0	1
Guess		39%		33%	5	9	10	13	3	6	0	1
StringUtils		95%		93%	2	13	1	14	1	5	0	1
AuthService		100%		85%	3	15	0	18	0	5	0	1
Calculator		100%		n/a	0	5	0	5	0	5	0	1
Total	83 of 464	82%	19 of 64	70%	21	69	24	132	4	37	0	5

Figure 3.1. JaCoCo output example

Pit Test Coverage Report

Package Summary

com.coverage

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
5	82%	108/132	86%	73/85	95%	73/77

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
AuthService.java	100%	18/18	100%	18/18	100%	18/18
Calculator.java	100%	5/5	100%	8/8	100%	8/8
Contact.java	84%	69/82	91%	32/35	91%	32/35
Guess.java	23%	3/13	10%	1/10	50%	1/2
StringUtils.java	93%	13/14	100%	14/14	100%	14/14

Report generated by PIT 1.15.8

Figure 3.2. PITest output example

3.2.4 Static analysis results

At the end of the analysis, all the results have been gathered in different tables, one for each case (Figure 3.3). Once all the results are gathered, the different performances have been analyzed, to see which model builds the

	gpt-3.5-turbo	gpt-3.5-turbo-16k	gpt-3.5-turbo-0125	gpt-3.5-turbo-0613	gpt-3.5-turbo-1106_10	gpt-3.5-turbo-1106_30	gpt-4	gpt-4-turbo-preview
Cyclomatic complexity	100,0%	80,0%	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%
Cognitive complexity	100,0%	85,0%	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%
Lines of code (LOC)	90,0%	75,0%	90,0%	90,0%	90,0%	95,0%	100,0%	100,0%
Number of methods	95,0%	70,0%	95,0%	100,0%	95,0%	100,0%	100,0%	100,0%
Assertion density	100,0%	90,0%	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%
Arrange-Act-Assert (AAA) Sequence Compliance	100,0%	95,0%	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%
Test smells	100,0%	85,0%	95,0%	95,0%	95,0%	95,0%	100,0%	100,0%
Duplicated code	100,0%	80,0%	85,0%	100,0%	100,0%	85,0%	100,0%	100,0%
Magic numbers	90,0%	90,0%	90,0%	85,0%	90,0%	90,0%	80,0%	80,0%
Code coverage	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%
Mutation coverage	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%
Comment-to-code ratio	80,0%	80,0%	80,0%	80,0%	85,0%	80,0%	50,0%	90,0%
Test code consistency	100,0%	90,0%	95,0%	90,0%	100,0%	90,0%	100,0%	100,0%
Overall evaluation	97,9%	87,4%	95,6%	96,5%	97,6%	95,8%	97,3%	98,5%

Figure 3.3. Static analysis - Case 1

most accurate unit tests. To make sure that the parameters have different importance, each one of them has been multiplied by a number, that represents its impact on the test quality (1 for *low*, 2 for *medium*, 3 for *high*). Without considering the coverage parameters (Figure 3.4), the model that

NO COVERAGE	PM - C1	PM - C2	FS - 1S	FS - 3S	FS - 5S	AVG
gpt-3.5-turbo	95,0%	94,0%	86,0%	95,0%	92,0%	92,4%
gpt-3.5-turbo-16k	85,0%	92,0%	96,0%	94,0%	88,0%	91,0%
gpt-3.5-turbo-0125	94,0%	91,0%	96,0%	94,0%	90,0%	93,0%
gpt-3.5-turbo-0613	94,0%	94,0%	82,0%	92,0%	89,0%	90,2%
gpt-3.5-turbo-1106_10	96,0%	93,0%	85,0%	92,0%	89,0%	91,0%
gpt-3.5-turbo-1106_30	94,0%	98,0%	88,0%	93,0%	91,0%	92,8%
gpt-4	95,0%	92,0%	97,0%	97,0%	92,0%	94,6%
gpt-4-turbo-preview	97,0%	93,0%	95,0%	97,0%	90,0%	94,4%

Figure 3.4. Static analysis results (coverage not considered)

performs the best is gpt-4, while the one with the worst performance on average is gpt-3.5-0613. Most probably this is caused by the low width of the context window of the model. Considering the coverage (Figure 3.5), the

WITH COVERAGE	PM - C1	PM - C2	FS - 1S	FS - 3S	FS - 5S	AVG
gpt-3.5-turbo	98,0%	88,0%	90,0%	96,0%	87,0%	91,8%
gpt-3.5-turbo-16k	87,0%	87,0%	96,0%	95,0%	83,0%	89,6%
gpt-3.5-turbo-0125	96,0%	86,0%	98,0%	93,0%	86,0%	91,8%
gpt-3.5-turbo-0613	97,0%	88,0%	88,0%	95,0%	90,0%	91,6%
gpt-3.5-turbo-1106_10	98,0%	80,0%	92,0%	95,0%	81,0%	89,2%
gpt-3.5-turbo-1106_30	96,0%	96,0%	92,0%	95,0%	82,0%	92,2%
gpt-4	97,0%	90,0%	99,0%	97,0%	92,0%	95,0%
gpt-4-turbo-preview	98,0%	89,0%	97,0%	98,0%	85,0%	93,4%

Figure 3.5. Static analysis results (coverage considered)

best-performing model is still gpt-4, while the worst is another one of the fine-tuned models (gpt-3.5-1106_10).

Overall the best-performing model is **gpt-4**, and it will be the one used in the application to generate the unit tests.

3.3 Creation of the Spring Boot application

The initial idea was to create a little Spring Boot application of a fictional university, with students, courses and professors as models, and to write some issues about the creation of endpoints (GET, POST, PUT, DELETE) related to these models. There was an equal distribution between easy issues (such as the creation of a single student) and more difficult ones (such as the deletion of a professor with cascade deletion logic on courses).

The aim of these issues was to **use the differences** that can be retrieved in the opened pull request **to fine-tune a GPT-3.5 model**. In this way, the model would have provided the code in the same way and with the same strategy chosen by the developer (for example, error checks on the controller files and usage of MongoTemplate on service files).

After solving 13 issues (since at least 10 are required to fine-tune a model [129]), the gpt-3.5-1106 model (chosen since it is the one with the most satisfying characteristics, both in context window length and in training data aging) has been fine-tuned with the issue title and issue description as the user message, while the diffs related to that issue were used as the assistant message (Figure 3.6). Even though the data had been written in the best way possible to create an efficient fine-tuned model, the results retrieved by using this fine-tuned model were not satisfying, since it was difficult to explain that the diffs were not the desired output of the issue, but they should have been used **as a blueprint** to understand how the code should be written. Most probably this information had to be written somewhere in the fine-tuning conversation, but it has been considered a dangerous move, since many messages were slightly below the token limit and the conversation must be the same for all the messages (otherwise the dataset would have not been considered valid).

Since the cost of fine-tuning is advantageous only if it is going to be used for a long time and it is quite difficult to provide a satisfactory dataset, the preferred model used for the code generation point of view is still gpt-4.

```
{
    "messages":[
        {
            "role":"system",
            "content":"You are a great Java coder, able to efficiently solve issues about a Spring
            Boot application that uses Java 17 and Gradle. Once the user opens an issue, analyze the
            request and provide a proper solution, creating new files or updating the existing ones
            in a coherent way."
        },
        {
            "role":"user",
            "content":"Issue title: Create Professor data model . Issue description: Create the
            Professor data model. Professor should have the following properties: id: String, name:
            String, surname: String, title: String, email: String, phone: String, department:
            String, office: String"
        },
        {
            "role":"assistant",
            "content":"diff --git a/CHANGELOG.md b/CHANGELOG.md index c2fa740..5fe78c1 100644 --- a/
            CHANGELOG.md +++ b/CHANGELOG.md @@ -12,7 +12,7 @@ on [Google Changelog](https://raw.
            githubusercontent.com/googleapis/release-pleas ### Bug Fixes ### Features - +- Add
            Professor data model under com.ai.university.model package. ([#4](https://github.com/
            gdb-99/spring_university/pull/4)) closes ([#1](https://github.com/gdb-99/
            spring_university/issues/1)) ### Code Refactoring ### Configurations \\ No newline at
            end of file diff --git a/src/main/java/com/ai/university/model/Professor.java b/src/main/
            java/com/ai/university/model/Professor.java new file mode 100644 index 0000000..ad78a02
            --- /dev/null +++ b/src/main/java/com/ai/university/model/Professor.java @@ -0,0 +1,16
            @@ +package com.ai.university.model; + +import lombok.Data; + +@Data +public class
            Professor { + String id; + String name; + String surname; + String title; + String
            email; + String phone; + String department; + String office; + +}"
        }
    ]
}
```

Figure 3.6. Fine-tuning dataset - single message overview

3.4 Giving context to ChatGPT

The most difficult task was to **give enough context to ChatGPT**, since even though the gpt-4 model is highly performing, it has a restricted context window; also, finding a good strategy to retrieve the main information of a repository and give them to the model without going over the token limit is pretty tricky, since there will be some information that may be useless for the user, but could represent a turning point for the model.

3.4.1 Extraction of main information

Since in this project everything is done on a Spring Boot application, the code will be written in **Java**. Knowing that a Java class has a pretty straightforward structure, the **position** of the information could be used to **extract the main data** of each class, such as **imports**, **class names**, **attributes** and **method prototypes**.

The initial strategy consisted of giving the model these four elements, accompanied by a **brief user explanation** and seeing if it was able to understand what was the general content of the repository.

After this, another test was provided, which consisted of **giving a basic task**, such as writing a specific method, to see if the format would be coherent with the received data, or to **extend the specific class** with more methods, to see how well the content of the repository was understood and to be able to fix the request in the moment (by extending the user explanation, for example).

Nevertheless, if the project is big (or it contains many well-structured classes) or the user explanation is not good enough, the model would not receive the desired context and this extraction operation would be completely worthless.

Overall, this process provided good results at first, but it would take too long and the amount of time wasted on giving the information could be used on solving the tasks manually.

Since the objective of this thesis is to automate various operations, this strategy won't work efficiently and it is necessary to find another way to solve this problem.

3.5 LangChain

The turning point of this whole project was given by the usage of **LangChain** [130], an **open-source framework** for developing applications that connect external sources of data and computation to large language models (LLMs). In this way, the developed applications are **powered by large language models** (Figure 3.8).

Concretely, this framework consists of some open-source libraries, such as *langchain-core* (that contains the base abstractions and LangChain Expression Language), *langchain-community* (third-party integrations, such as

Figure 3.7. LangChain logo

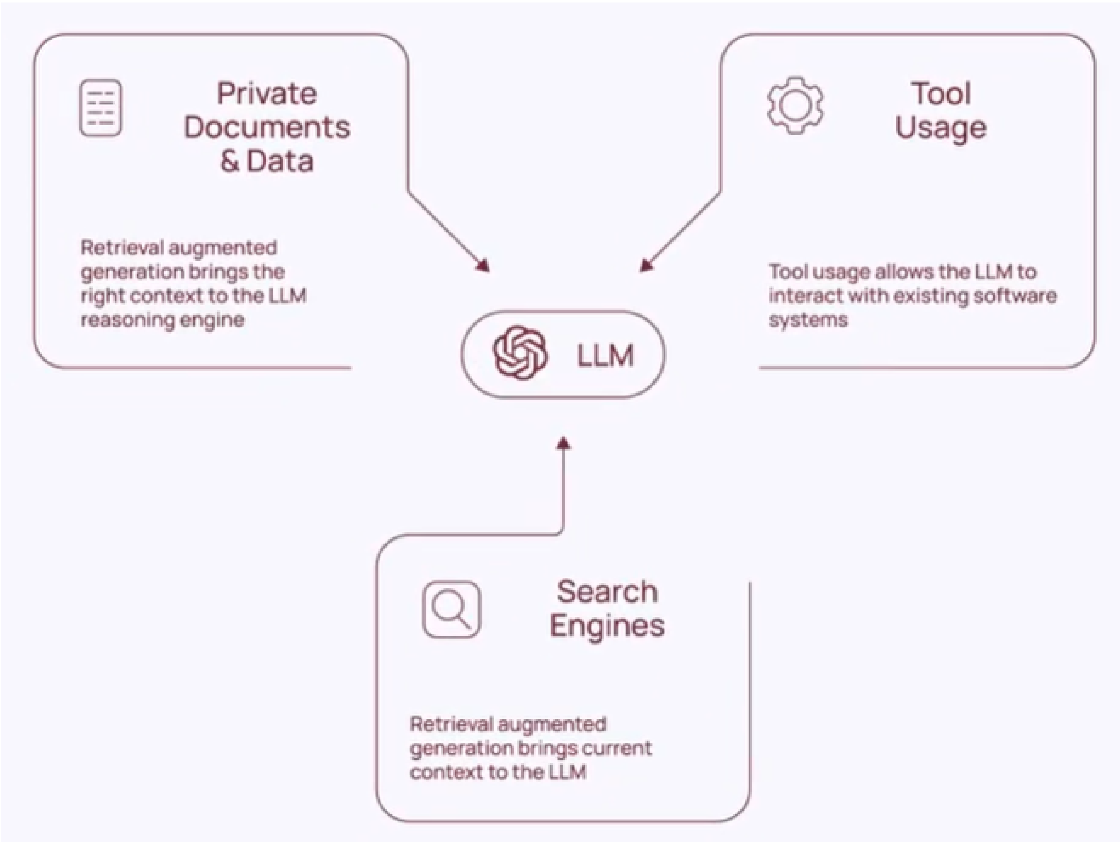


Figure 3.8. LangChain - main overview

langchain-openai) and *langchain* (that contains chains, agents and retrieval strategies that constitute an application's cognitive nature).

LangChain provides standard, extendable interfaces and external integrations for the following main components [131]:

- **Model I/O**: formatting and managing language model input and output;
  - **Prompts**: formatting for LLM inputs that guide generation;

  - **Chat models**: interfaces for language models that use chat messages as inputs and returns chat messages as outputs (as opposed to using plain text);
  - **LLMs**: interfaces for language models that use plain text as input and output;

- **Retrieval**: interface with application-specific data (e.g. RAG);
  - **Document loaders**: load data from a source as *Documents* for later processing;
  - **Text splitters**: transform source documents to suit the application better;
  - **Embedding models**: create vector representations of a piece of text, allowing for natural language search;
  - **Vector stores**: interfaces for specialized databases that can search over unstructured data with natural language;
  - **Retrievers**: more generic interfaces that return documents given an unstructured query;

- **Composition**: higher-level components that combine other arbitrary systems and/or LangChain primitives;
  - **Tools**: interfaces that allow an LLM to interact with external systems;
  - **Agents**: constructs that choose which tools to use given high-level directives;
  - **Chains**: building block-style compositions of other runnables;

- **Memory**: persist application state between runs of a chain;

- **Callbacks**: log and stream intermediate steps of any chain;

This constitutes a complete set of **interoperable** and **interchangeable** building blocks (Figure 3.9).

Common end-to-end use tasks with LangChain include chatbots, query analysis, extraction of structured output, usage of RAGs, combined usage of SQL and CSV, etc.

Since the main problem was in the amount of information given to the LLM, LangChain solves this using the *retrieval* logic, to provide additional

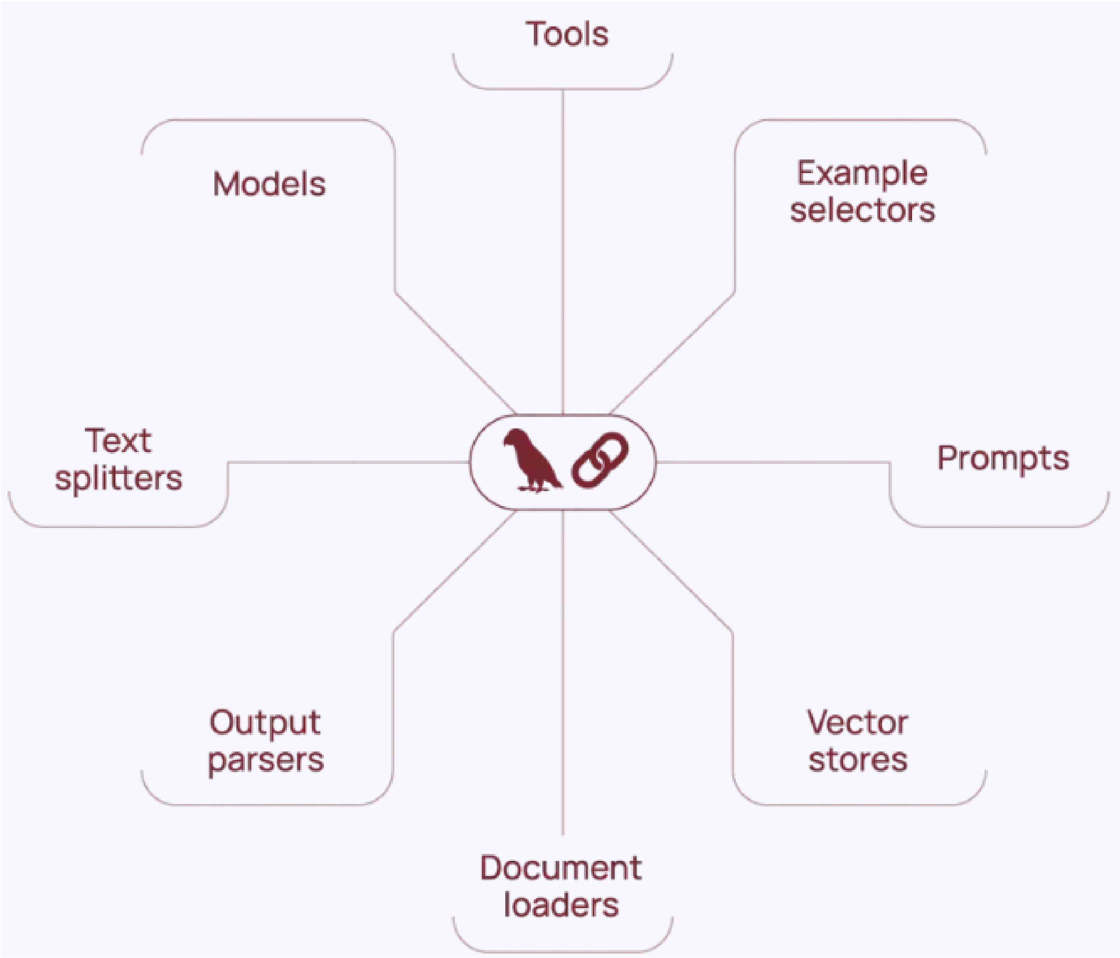


Figure 3.9. LangChain - tools overview

context to the user message. Then, a *retriever* can be used to fetch only the most relevant pieces of information and pass those in.

3.5.1 LangChain Agent

The most useful LangChain feature in this project is the **Agents** [132]: they can be used to turn LLMs into **reasoning engines that take action, responsibly**. These agents can be considered as *copilots*, able to write first drafts for reviews, wait for approval before execution or act on behalf of the developer.

The core idea of agents is to use a language model to choose a **sequence**

of actions to take. In chains, a sequence of actions is hardcoded; in agents, a language model is used as a reasoning engine to determine which actions to take and in which order. This is usually powered by a prompt, a language model and an output parser. (Figure 3.10). The great advantage over chains

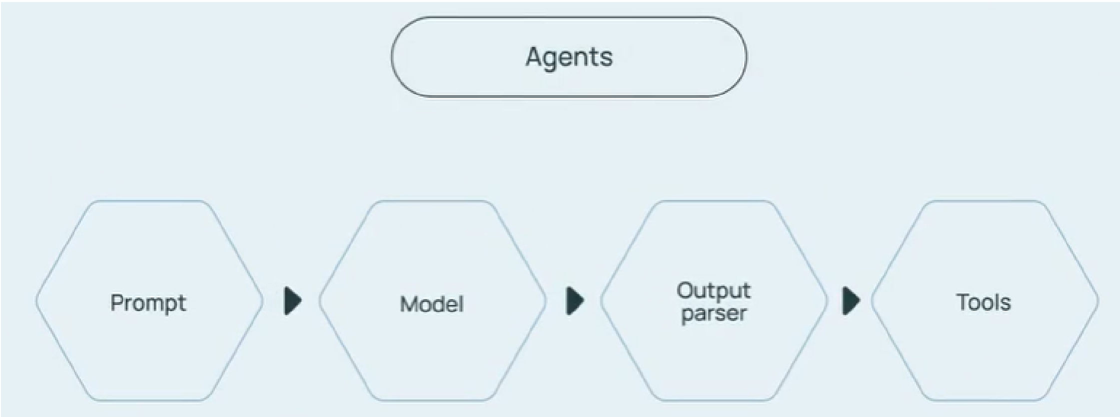


Figure 3.10. LangChain Agent - flow overview

is that is the model itself that decides how many times the tools must be used and when. Most of the time (and also in this project), the agent type used is the **tool-calling agent**, which is the most reliable kind and the recommended one for most use cases (Figure 3.11).

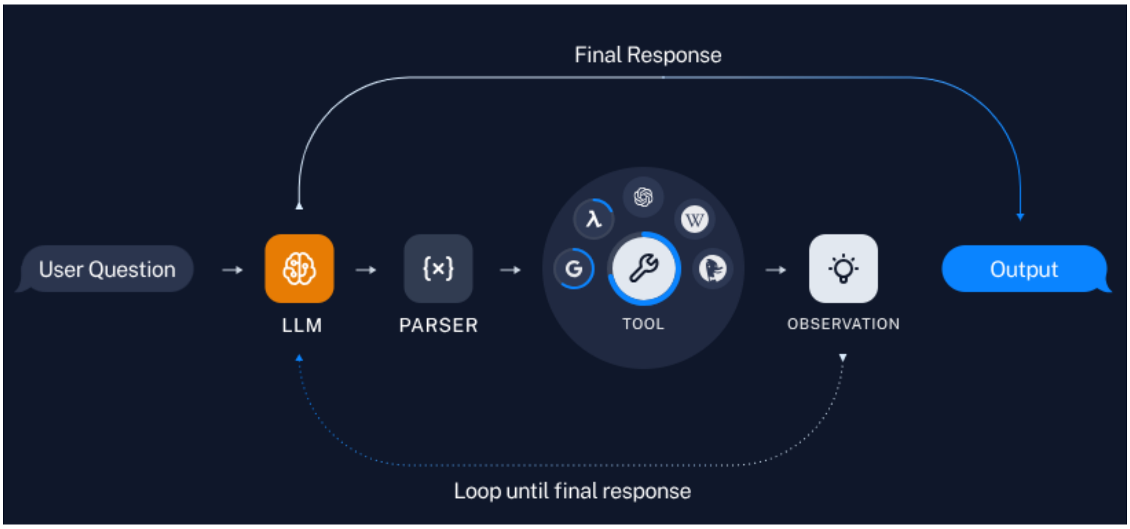


Figure 3.11. LangChain Agent - details on flow

3.5.2 GitHub Integration

Among the various available integrations in LangChain, one of the most important and supported ones is the **GitHub integration**. It enables the communication between a large language model and a GitHub repository in an efficient way through a LangChain Agent.

This communication is conveyed through a GitHub Application (Figure 3.12): **GitHub Apps** are tools that *extend* the basic GitHub functionality [133]; these can do both things on GitHub (open issues, comment on pull requests, etc.) or do things outside GitHub but related to it in some way (such as post on Slack when an issue is opened on GitHub).

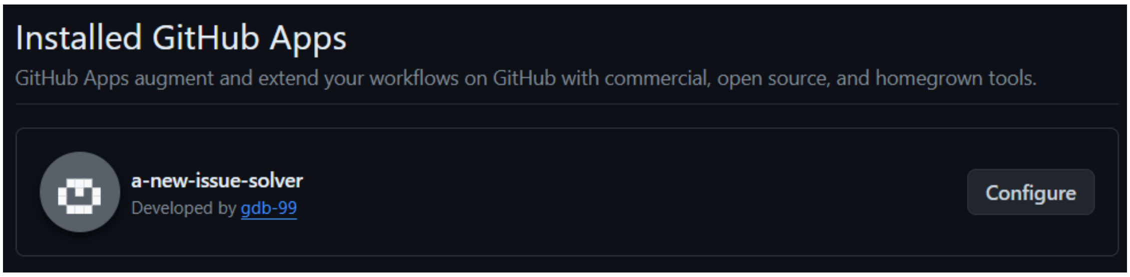


Figure 3.12. GitHub App - overview

To use an App (whether it was created or downloaded through the GitHub Marketplace), this must be **installed** on the user or organization account: in this way, the user grants the app permissions to read or modify a repository or organization data (the specific permissions can be chosen in the settings menu of the app). Also, the GitHub App can only do things that **both the user and the app have permission to do**. For example, if the user has write access to a repository but the GitHub App only has read access, then the app can only read the contents of the repository even when it is acting on the user's behalf.

When the app is installed, the user **specifies the repositories** on which this app can access and operate (Figure 3.13). As seen in the figure, there are five repositories permissions (two of type *read-only* and three of type *read and write*), explicitly required by LangChain to execute operations in a GitHub repository.

It must also be noted that, if the app is created by the developer, the source code has to be in a repository **different** from the one in which the application is operating and **cannot be a local source code**, since a URL

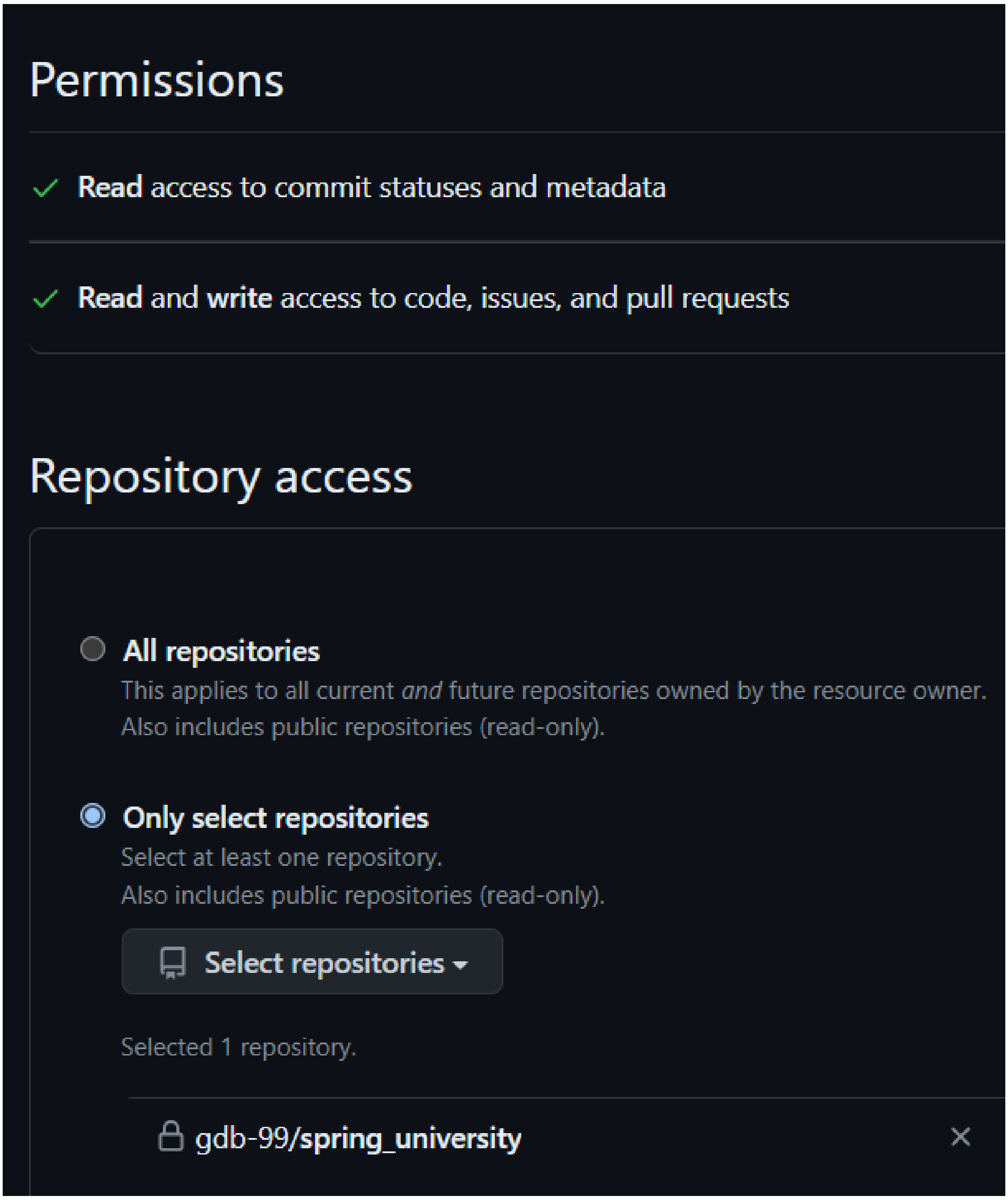


Figure 3.13. GitHub App - permission overview

must be inserted in the app settings. In this case, the source code is stored on GitLab, since it is the platform preferred by the company, even though the app operates on a GitHub repository. This apparent incoherence doesn't cause problems, since there are no particular restrictions on the GitHub App settings.

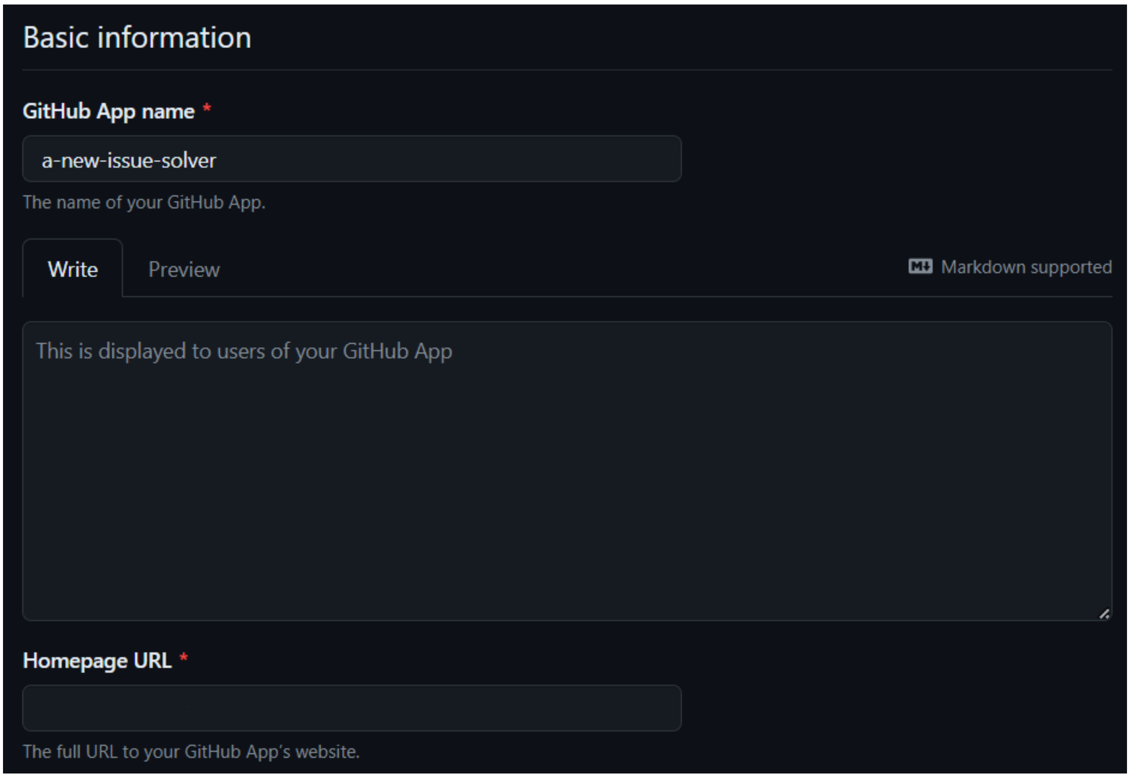


Figure 3.14. GitHub App - settings overview

Unlike GitLab, which has just a personal access token that doesn't allow to go in depth in the repositories operations like the app, this GitHub App is exactly what was needed in this project: it constitutes the perfect bridge between the OpenAI model and the repository.

3.5.3 GitHub Agent - a coding perspective

From a coding point of view, the agent is composed of three main parts: the preparation, the input and the invocation.

In the **preparation**, there is the selection of the preferred large language model (as stated before (3.3), gpt-4 is the model that will be used in this project) and the preparation of the needed toolkit (in this case, the GitHub toolkit), to gather all the possible operation that the agent can do in the

GitHub repository (Figure 3.15).

```
llm = ChatOpenAI(model="gpt-4-1106-preview", temperature=0, verbose=True)
github = GitHubAPIWrapper()
toolkit = GitHubToolkit.from_github_api_wrapper(github_api_wrapper=github)
tools = toolkit.get_tools()
```

Figure 3.15. LangChain Agent - preparation overview

The following are the available tools in the GitHub toolkit (Figure 3.16), which indicate the actions that the agent can perform inside a specific GitHub repository.

After the agent preparation, the next step is to give the **input** to the model (Figure 3.17). First of all, the developer must create the system message, which is the one that gives the model the general overview of the situation (the more detailed it is, the better); it is also possible to keep the *chat history*, to add context to the model and make sure it remembers the actions done previously. It is not used in this case, since it was redundant and the *agent_scratchpad* functionality was sufficient (it is where the tools descriptions are loaded for the agent to understand and use them properly in the intermediate steps).

Lastly, the agent is **invoked**, with the AgentExecutor creation (Figure 3.18). The **AgentExecutor** is the actual tool that performs all the *thinking operations*: calls the agent, executes the actions it chooses, passes the action outputs back to the agent and repeats (Figure 3.19).

3.5.4 Note on LangChain usage

A thing to be noted before proceeding further is that, since this library is open source and is growing rapidly, there were some errors in the version used in this project (0.1.11) that have been manually fixed and the documentation is not always up to date.

For example, the invocation of the Agent in the documentation is shown with a deprecated method, or the GitHub toolkit taken with the new agent invocation method is not retrieved correctly, since this new function follows a specific regular expression in which characters such as blank spaces are not allowed (Figure 3.20 and 3.21). Luckily, these errors were not that serious to put the whole application construction at risk and have been solved correctly,

```
Available tools:
    Get Issues
    Get Issue
    Comment on Issue
    List open pull requests (PRs)
    Get Pull Request
    Overview of files included in PR
    Create Pull Request
    List Pull Requests' Files
    Create File
    Read File
    Update File
    Delete File
    Overview of existing files in Main branch
    Overview of files in current working branch
    List branches in this repository
    Set active branch
    Create a new branch
    Get files from a directory
    Search issues and pull requests
    Search code
    Create review request
```

Figure 3.16. GitHub toolkit - overview

thanks to the active community that supports this really useful framework.

To help potential future developers, a Markdown file summing up the

```
prompt = ChatPromptTemplate.from_messages([
("system", "You have the software engineering capabilities of a Google Principle
engineer. You have access to a repository, with its issues and pull requests. This
repository contains a Spring Boot application, with Spring Boot 3.2.4, Java 17 and Gradle
8.6. Note that MongoTemplate is being used in this repository (and not MongoRepository),
so use the same strategy. Note that the package is called 'com.ai.university': do all of
your operations on this package and its subpackages."),
("human", "{input}"),
MessagesPlaceholder("agent_scratchpad"),
])
```

Figure 3.17. LangChain Agent - input overview

```
agent = create_openai_tools_agent(llm, tools, prompt)

return AgentExecutor(agent=agent, tools=tools, verbose=True)
```

Figure 3.18. LangChain Agent - invocation overview

```
next_action = agent.get_action(...)
while next_action != AgentFinish:
    observation = run(next_action)
    next_action = agent.get_action(..., next_action, observation)
return next_action
```

Figure 3.19. AgentExecutor - pseudocode

```
openai.BadRequestError: Error code: 400 - {'error': {'message': "'Get Issues'
does not match '^[a-zA-Z0-9_-]{1,64}$' - 'tools.0.function.name'", 'type':
'invalid_request_error', 'param': None, 'code': None}}
```

Figure 3.20. LangChain - library error example

various fixes done inside the library has been created, with the general description of the error, why this happens, the correspondent GitHub issue on the official LangChain repository and the solution of the correspondent error.

Also, it has opened an issue on the official LangChain repository, since a task of the GitHub toolkit was not working correctly (the navigation of files

```
pattern = re.compile(r'[^a-zA-Z0-9_-]')

for tool in tools:
    tool.name = re.sub(pattern, '_', tool.name)

agent = create_openai_tools_agent(llm, tools, prompt)
```

Figure 3.21. LangChain - library error solution

in a pull request) and it was necessary to solve to continue with the project.

3.6 GitHub Webhooks

To make sure that the Agent operations are triggered at the correct moment, it has been used one of the most important GitHub features: **GitHub Webhooks** [134]. They provide a way for notifications to be delivered to an external web server whenever certain events occur on GitHub (opened issue, opened pull request, merged code, etc). As opposed to polling an API (which

Figure 3.22. GitHub Webhooks logo

is an activity that wastes a lot of resources and time), webhooks are used to **receive data as it happens**; the user only needs to express interest in an event once, when the webhook is created.

By creating a webhook, the developer specifies a URL and subscribes to events that occur on GitHub: when an event that the webhook is subscribed to occurs, GitHub sends an HTTP request with data about the event to the specified URL.

To underline the advantages, webhooks **scale better** than singular API calls: if many resources need to be monitored, calling the API for each one may cause it to hit the API rate limit quota quickly and the developer may even not realize that. Instead, the user can subscribe to multiple webhooks and receive information only when an event happens.

It must also be noted that to create and manage webhooks, the user must own or **have admin access** to the resource where the webhook is created and listen for events. For example, to manage webhooks in an organization, the user needs admin permissions for that organization.

In this case, four main events are the ones needed for the interaction of the application (all of these are repository webhooks):

- An issue has been opened;
- A pull request has been opened;
- A GitHub Action has failed and the bug must be fixed;
- A reviewer requested changes in the code.

A webhook can be inserted in the repository easily, however, it is important to make sure that it is triggered only when needed, so that the amount of work done by the servers is minimal.

Also, its URL **cannot be the localhost**, but it is necessary to use another tool to expose the localhost URL as if it is external [135]. In this case, **ngrok** [136] is the tool used to test the functionalities of the webhooks.

The webhooks can be inserted directly through GitHub (Figure 3.23); the needed parameters are the webhook URL, the content type (*application/json* in this example), the webhook secret (it is a good practice to use it, to ensure

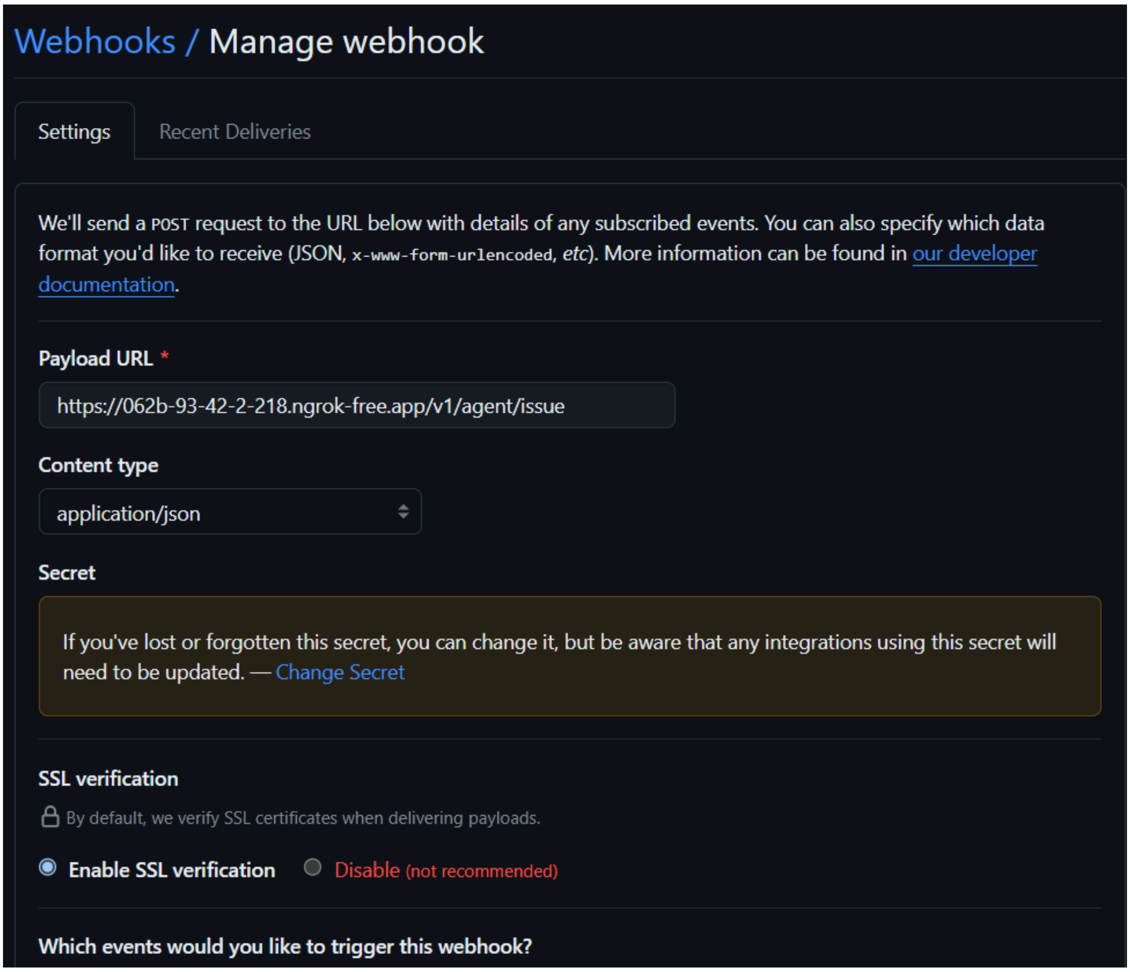


Figure 3.23. GitHub Webhook - overview

that the webhook delivery is from GitHub), the SSL verification and the type of event for which the webhook must be triggered.

In the following example, the webhook is activated when something that involves the issue event is triggered (opened, closed, labeled, etc.) (Figure 3.24).

The detail on when the Agent is triggered can be found in the code: as seen in the picture below (Figure 3.25), the Agent operation is triggered only if the issue is opened or reopened.

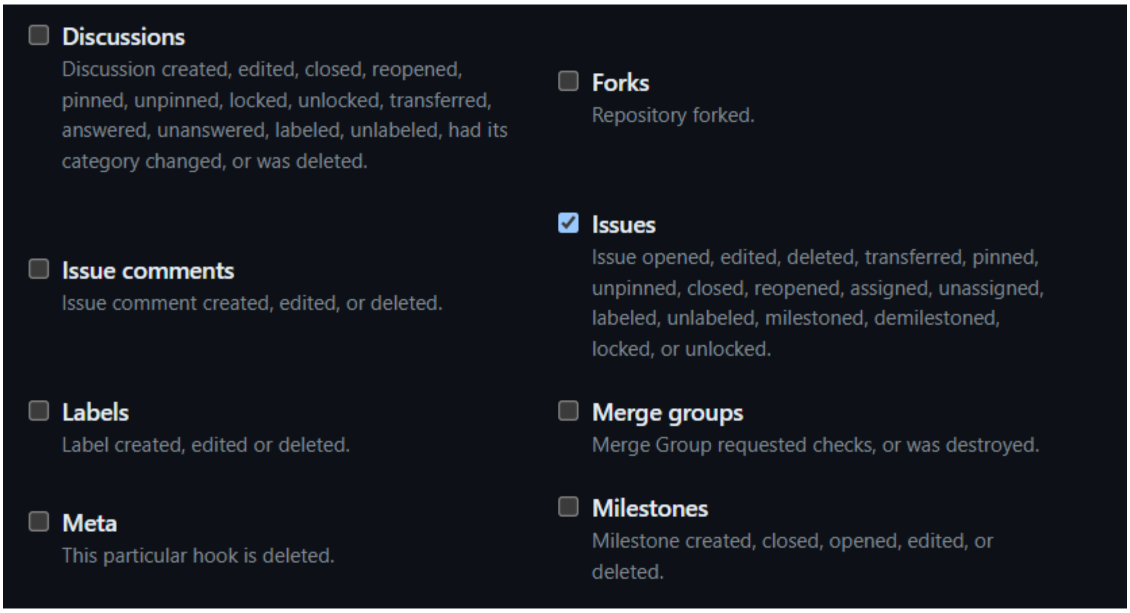


Figure 3.24. GitHub Webhook - detail on issues

```
#http://127.0.0.1:5000/v1/agent/issue
@app.route('/v1/agent/issue', methods=['POST'])
def handle_issue_webhook():
    data = request.json
    event_type = request.headers.get('X-GitHub-Event')

    if verify_signature(request):
        if event_type == 'issues' and (data['action'] == 'opened' or data['action'] == 'reopened'):
            issue_id = data['issue']['number']
            sender = data['sender']['login']
            handle_issues(issue_id, sender)
        return jsonify({'message': 'Issue handled correctly.'})
    else:
        return jsonify({'error': 'Invalid signature'}), 403
```

Figure 3.25. GitHub Webhook - restriction to opened/reopened issue

3.7 GitHub Actions

To automate the user's build, test and deployment pipeline, GitHub has a feature called **GitHub Actions** [137]. It is a continuous integration and continuous delivery (CI/CD) platform that lets the user **automate**, **customize** and **execute the software development workflows** right in their repository.

Figure 3.26. GitHub Actions logo

A GitHub Actions *workflow* can be configured to be triggered when an *event* occurs in the repository, such as a pull request being opened or an issue being created. The workflow contains one or more *jobs* that can run sequentially or in parallel. Each job will run inside its virtual machine *runner*, or inside a container, and has one or more *steps* that either run a script that you define or run an *action*, which is a reusable extension that can simplify the entire workflow (Figure 3.27).

The terms written above must be examined in depth to understand better how a single GitHub Action is composed:

- A **workflow** is a configurable automated process that will run one or more jobs. Workflows are defined by a *yaml* file checked into the repository and will run when triggered by an event in the same repository, or they can be triggered manually, or at a defined schedule. There could be multiple workflows in the same repository, each one related to a specific task, and a workflow can also be referenced within another one;

- An **event** is a specific activity in a repository that triggers a workflow run. For example, activity can originate from GitHub when someone creates a pull request, opens an issue or pushes a commit to a repository. The event in question can also trigger a workflow to run on a schedule, by posting to a REST API, or manually;

- A **job** is a set of steps in a workflow that is executed on the same runner. Each step is either a shell script that will be executed or an action that will be run. Steps are executed *in order* and are *dependent on each other*. Since each step is executed on the same runner, the data can be shared from one step to another. Job's dependencies can also be related to other jobs (by default, jobs have no dependencies and run in parallel with each other). When a job takes a dependency on another job, it will wait for the dependent job to complete before it can run;

- An **action** is a custom application for the GitHub Actions platform that performs a complex but frequently repeated task. Using an action helps to reduce the amount of repetitive code that may be written in your workflow files. An action can pull a git repository from GitHub, set up the correct toolchain for the build environment or set up the authentication to a cloud provider.

- A **runner** is a server that runs workflows when they're triggered. Each runner can run a single job at a time. GitHub provides Ubuntu Linux, Microsoft Windows and macOS runners to run workflows; each workflow run executes in a fresh, newly-provisioned virtual machine. GitHub also offers larger runners, which are available in larger configurations.

GitHub Actions feature is free for public repositories, to follow the open source legacy that characterizes the whole GitHub environment; however, it comes with a price in usage when the repository in question is private or the action is used in a company context (GitHub Enterprise system). Even though there is this apparent stumbling block, GitHub Actions plan includes free usage based on how much the action is run (using a "pay-as-you-go" pricing technique).

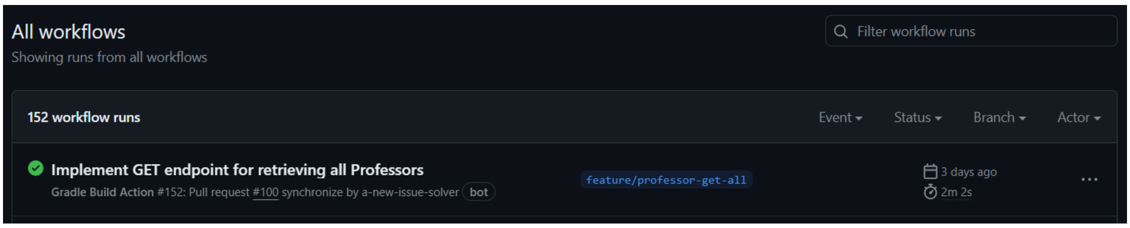


Figure 3.27. GitHub Action - Workflow run overview

In this project, the GitHub Action that is configured is activated each time a pull request is opened or edited. Since the repository contains a Spring Boot application that uses Gradle, this action performs a *Gradle Build* and a *Gradle Run Test* operations, to make sure that the code written by the agents is correct (Figure 3.28).

If the code contains errors, the log of the failed GitHub Action is retrieved and it is given to the agent to solve the problem(s) underlined in the workflow.

```yaml
name: Gradle Build Action
on: pull_request
jobs:
  build:
    runs-on: windows-2022
    steps:
      - uses: actions/checkout@v2
      - name: Set up JDK 17
        uses: actions/setup-java@v2
        with:
          java-version: 17
          distribution: 'adopt'
          cache: gradle
      - name: Change wrapper permissions
        run: chmod +x ./gradlew
      - name: Build with Gradle
        run: ./gradlew build
      - name: Run tests
        run: ./gradlew test
```

Figure 3.28. GitHub Action - Workflow code

3.8 GitHub branch protection rules

To ensure that the main branch is protected from irresponsible PR merges, a **GitHub branch protection rule** [138] can be added. In this case, the most important rule is to request a review of the code in the PR before merging it into the main branch.

If the code is not reviewed for the time being (Figure 3.29), or if the reviewer has requested some changes (Figure 3.30), the "Merge pull request" button is black, to indicate that this action cannot be done for now.

Instead, if the code is reviewed and it is considered approved (Figure 3.31), the "Merge pull request" button is green and the code can be merged into the main branch correctly.

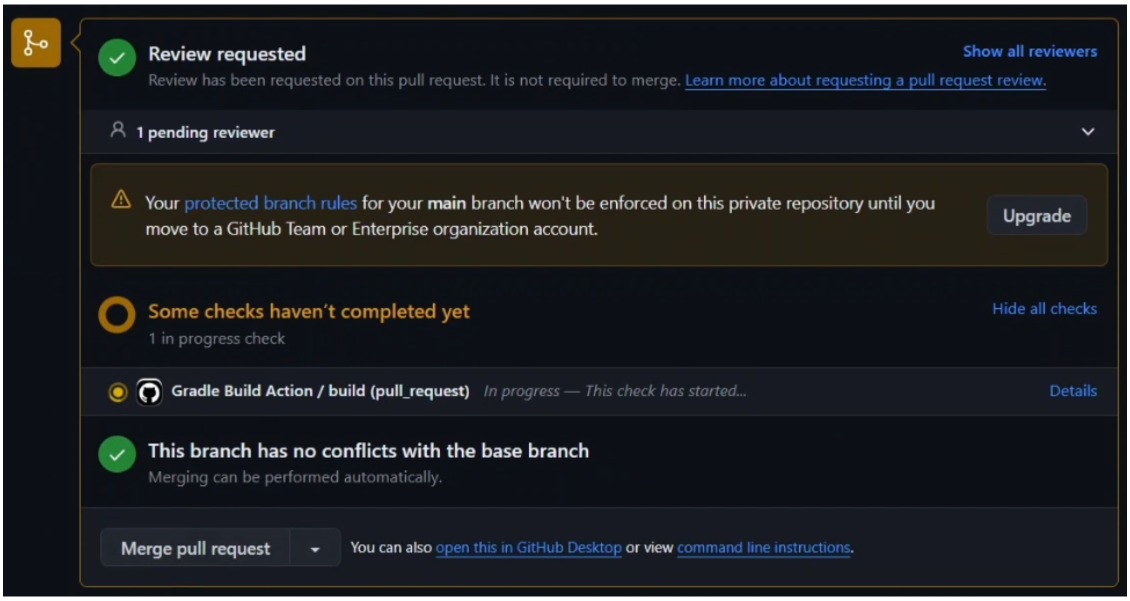


Figure 3.29. Merge status - Code to be reviewed

A note to be added is that this rule is actually bypassed if the branch is private and the GitHub profile is not of type *Team* or *Enterprise*. To follow the rule (as if it was applied correctly), the code is not merged until the approval of the code review.

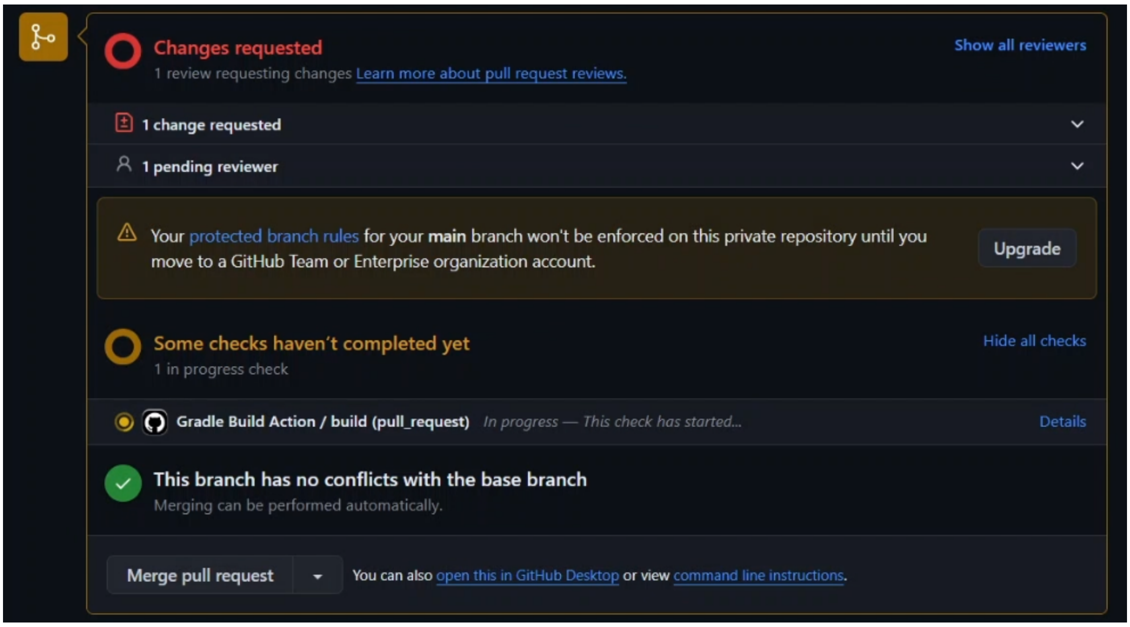


Figure 3.30. Merge status - Changes requested by the reviewer

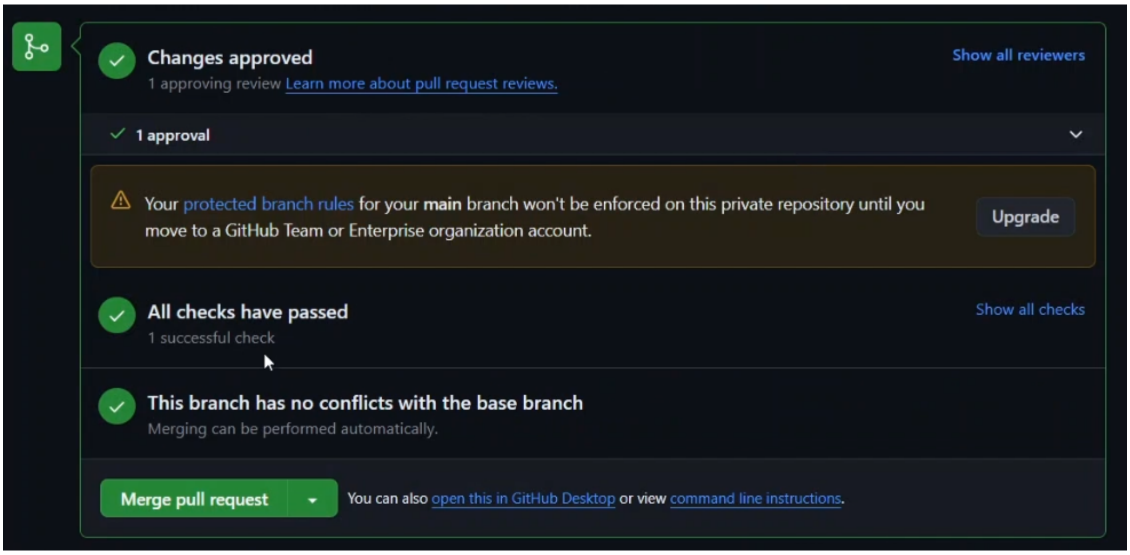


Figure 3.31. Merge status - Review approved

Chapter 4

Results

4.1 Structure of the application

The final application contains 4 LangChain GitHub Agents that will be triggered by a specific webhook; 3 of them are specific actions related to operations on a GitHub repository (opened issue, opened pull request and pull request review), while one is related to a GitHub Action.

Specifically, in this case, the flow of the application is the following:

1. First, a user opens an issue;

2. When the issue is opened, the first GitHub Webhook is triggered. This webhook calls the **Code Gen Agent** (the *code generator*);

3. The Code Gen Agent will write the code to solve the issue, by creating a branch and pushing the code there. Once it is done, it will open a pull request and request a review to the issue sender;

4. Once the PR is opened, the second webhook is triggered. This one will call the **Unit Test Agent**, that will write unit tests related to the code inside the PR and push it in the same branch of this PR;

5. At the same time, another webhook is triggered. This one governs the GitHub Action, which will perform a Gradle Build on the code and a Gradle Run Tests on the unit tests. If the Action fails, the **Bug Fix Agent** is called to solve the bugs noted in the log of the failed GitHub Action. Once it is done, the code will be pushed to the same branch of the PR;

6. If the Action doesn't fail, the last operation is to perform the code review (requested by the various Agents) to a specific developer;

7. If there are some changes to be done, this will trigger the final webhook, that calls the **Review Agent**. As the ones before, this Agent analyzes the code in the branch and studies the changes requested by the reviewer;

8. Once the Review Agent is done, it performs the necessary corrections to the code and pushes it into the PR branch. To ensure that everything is correct, the reviewer is asked again to check if the code now satisfies entirely the issue request;

9. If everything is correct, the Action doesn't fail, the code is approved by the reviewer and there are no conflicts with the main branch, the code can be merged.

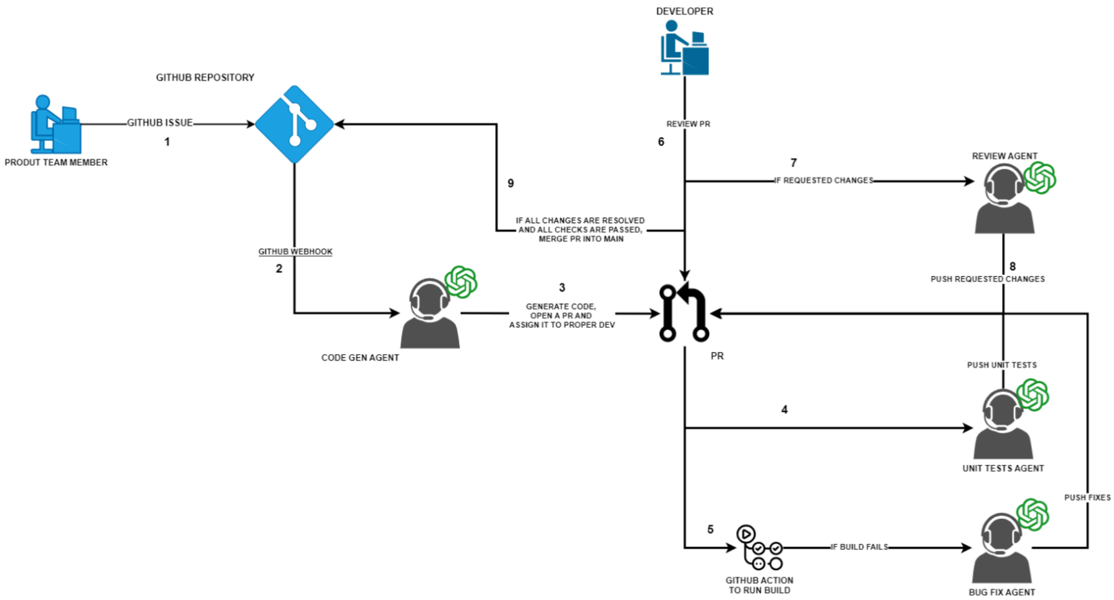
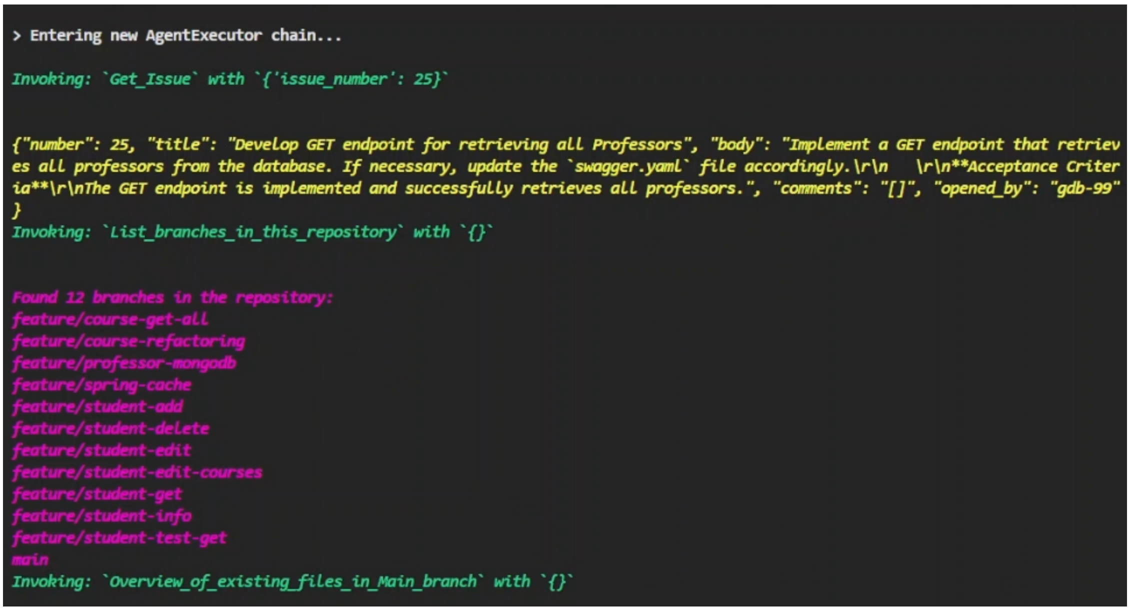


Figure 4.1. Final application flow



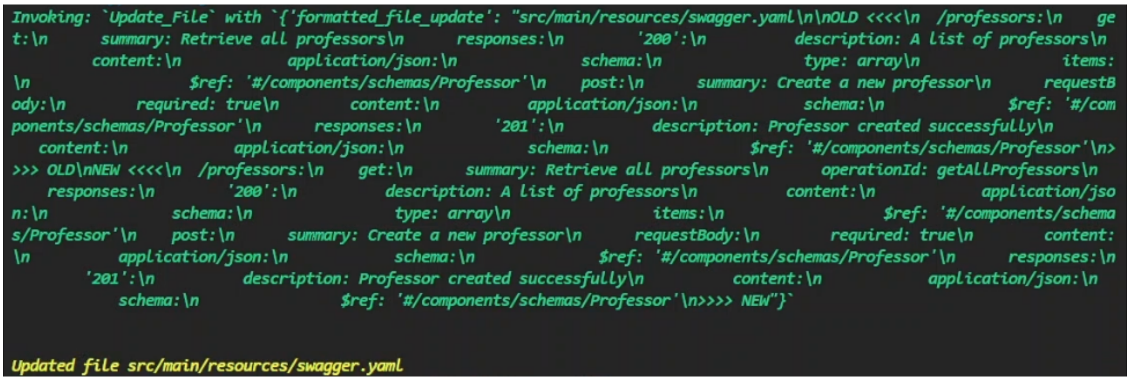
```
> Entering new AgentExecutor chain...

Invoking: `Get_Issue` with `{'issue_number': 25}`


{"number": 25, "title": "Develop GET endpoint for retrieving all Professors", "body": "Implement a GET endpoint that retriev
es all professors from the database. If necessary, update the `swagger.yaml` file accordingly.\r\n   \r\n**Acceptance Criter
ia**\r\nThe GET endpoint is implemented and successfully retrieves all professors.", "comments": "[]", "opened_by": "gdb-99"
}
Invoking: `List_branches_in_this_repository` with `{}`


Found 12 branches in the repository:
feature/course-get-all
feature/course-refactoring
feature/professor-mongodb
feature/spring-cache
feature/student-add
feature/student-delete
feature/student-edit
feature/student-edit-courses
feature/student-get
feature/student-info
feature/student-test-get
main
Invoking: `Overview_of_existing_files_in_Main_branch` with `{}`
```

Figure 4.2. Code Gen Agent - Start of the application



```
Invoking: `Update_File` with `{'formatted_file_update': "src/main/resources/swagger.yaml\n\nOLD <<<<\n  /professors:\n    ge
t:\n      summary: Retrieve all professors\n      responses:\n        '200':\n          description: A list of professors\n
        content:\n            application/json:\n              schema:\n                type: array\n                items:
\n                  $ref: '#/components/schemas/Professor'\n    post:\n      summary: Create a new professor\n      requestB
ody:\n        required: true\n        content:\n          application/json:\n            schema:\n              $ref: '#/com
ponents/schemas/Professor'\n      responses:\n        '201':\n          description: Professor created successfully\n
   content:\n            application/json:\n              schema:\n                $ref: '#/components/schemas/Professor'\n>
>>> OLD\nNEW <<<<\n  /professors:\n    get:\n      summary: Retrieve all professors\n      operationId: getAllProfessors\n
    responses:\n        '200':\n          description: A list of professors\n          content:\n            application/jso
n:\n              schema:\n                type: array\n                items:\n                  $ref: '#/components/schema
s/Professor'\n    post:\n      summary: Create a new professor\n      requestBody:\n        required: true\n        content:
\n          application/json:\n            schema:\n              $ref: '#/components/schemas/Professor'\n      responses:\n
        '201':\n          description: Professor created successfully\n          content:\n            application/json:\n
              schema:\n                $ref: '#/components/schemas/Professor'\n>>>> NEW"}`


Updated file src/main/resources/swagger.yaml
```

Figure 4.3. Code Gen Agent - Example of file update

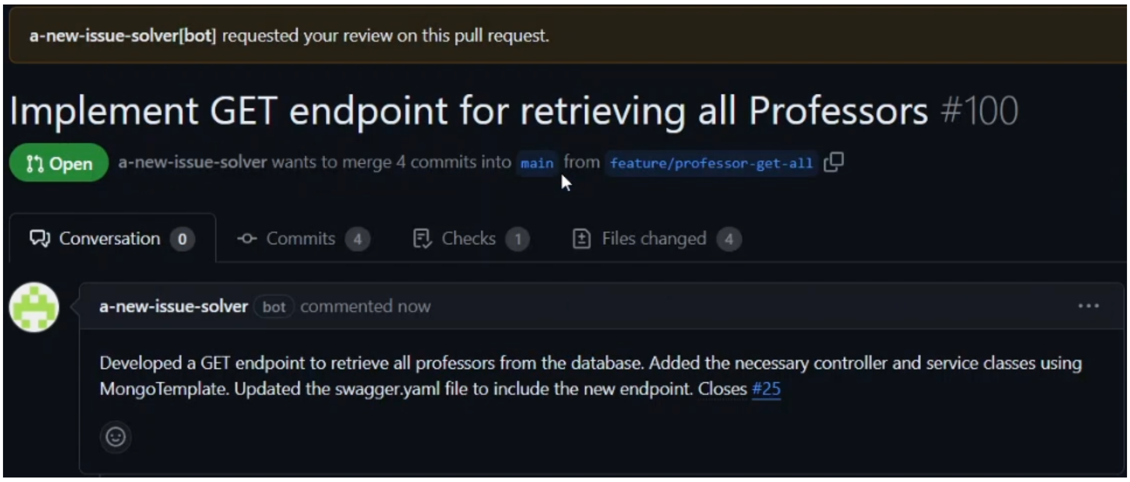


Figure 4.4. Code Gen Agent - Creation of pull request

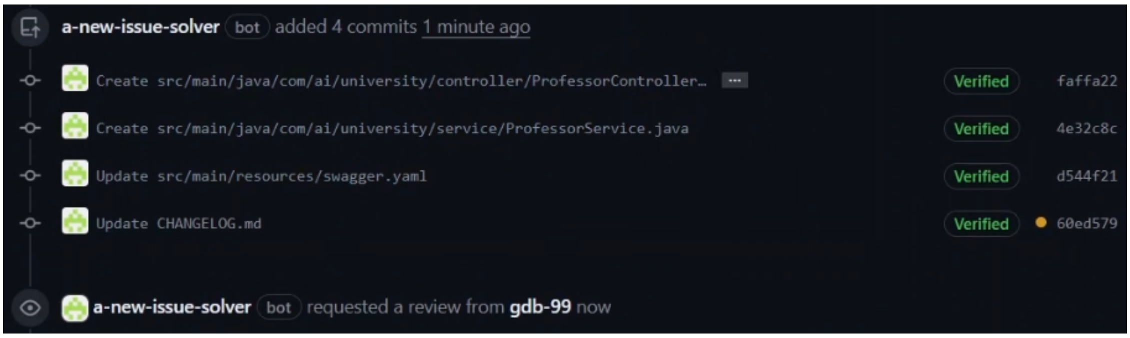


Figure 4.5. Code Gen Agent - Overview of commits

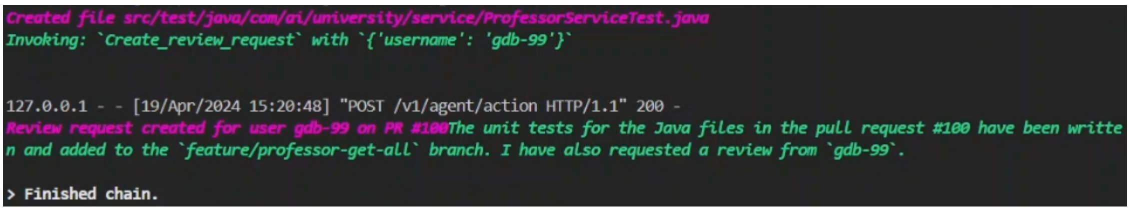


Figure 4.6. Code Gen Agent - Creation of review request

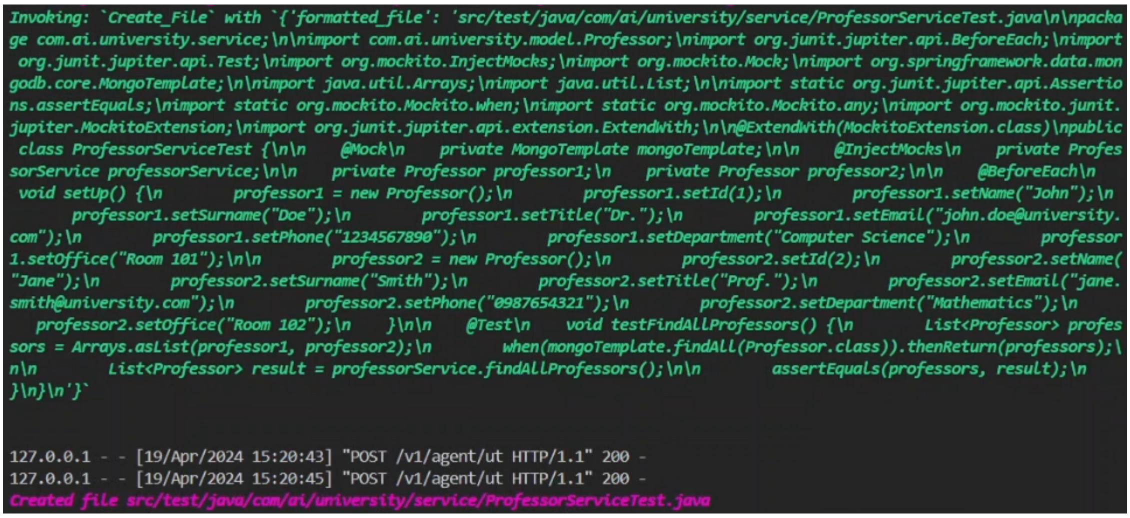


Figure 4.7. Unit Test Agent - Creation of unit tests

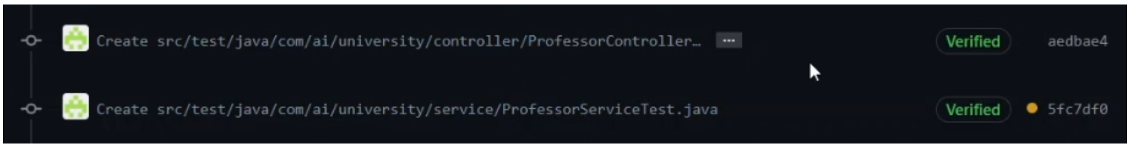


Figure 4.8. Unit Test Agent - Overview of commits

Figure 4.9. Successful conclusion of the Action - Bug Fix Agent not activated

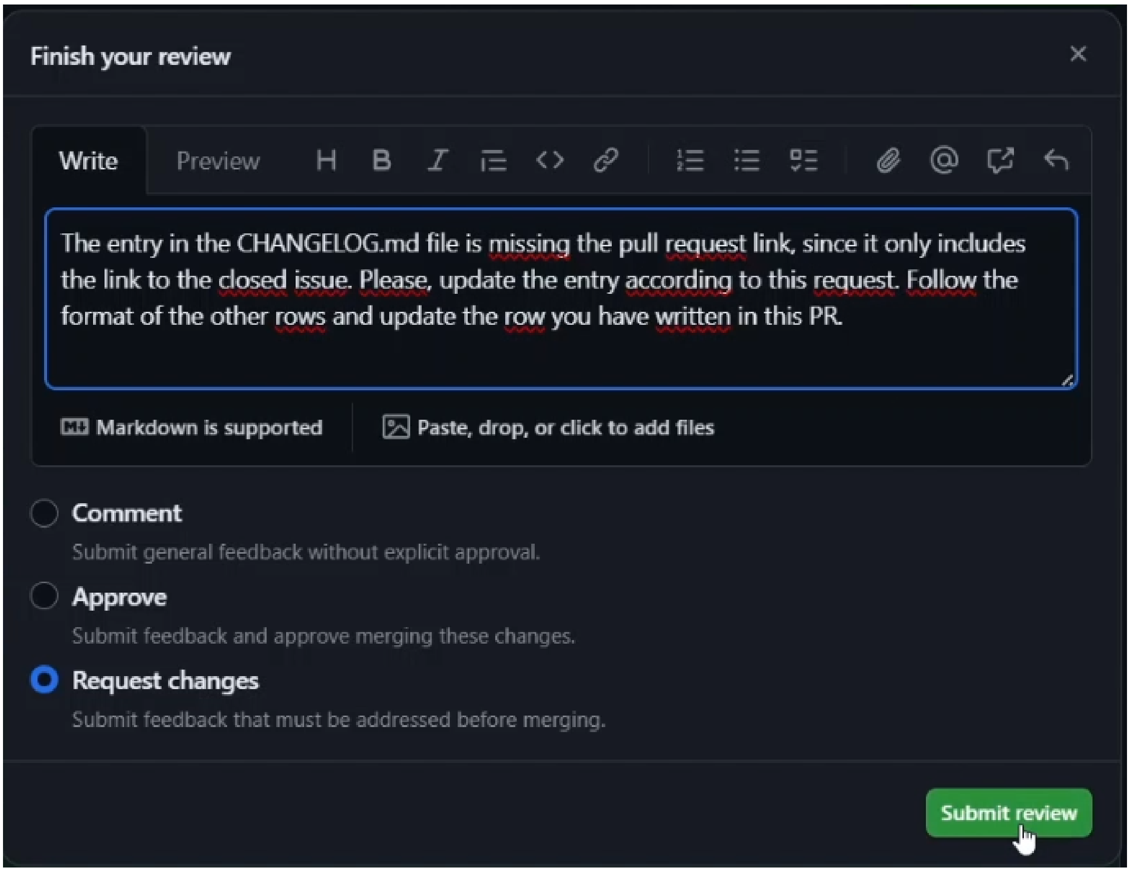


Figure 4.10. Example of changes requested

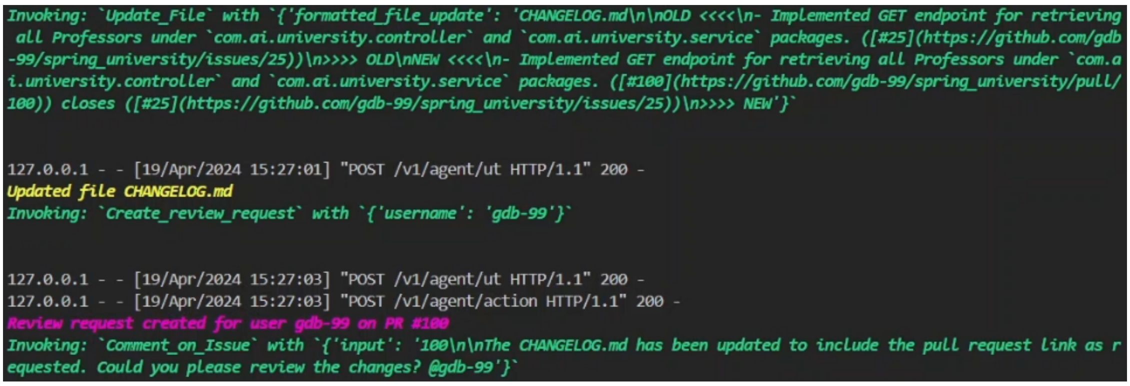


Figure 4.11. Review Agent - Solving requested changes

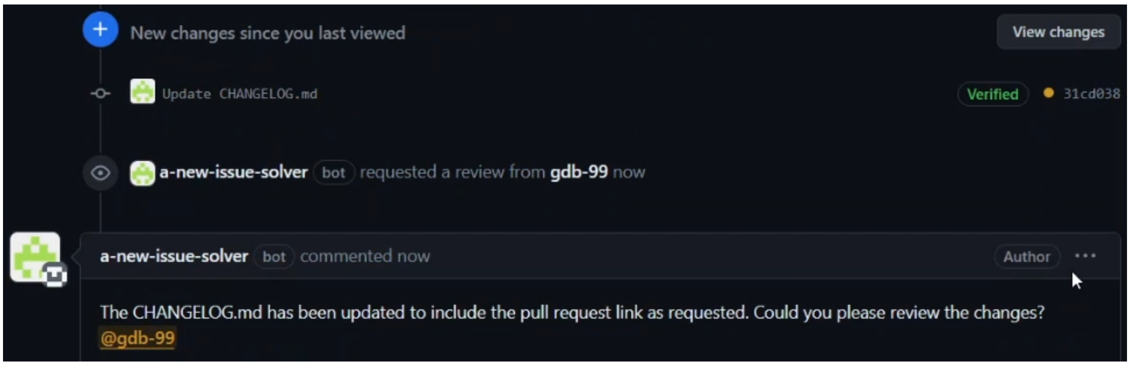


Figure 4.12. Review Agent - Commits overview and creation of a new review request

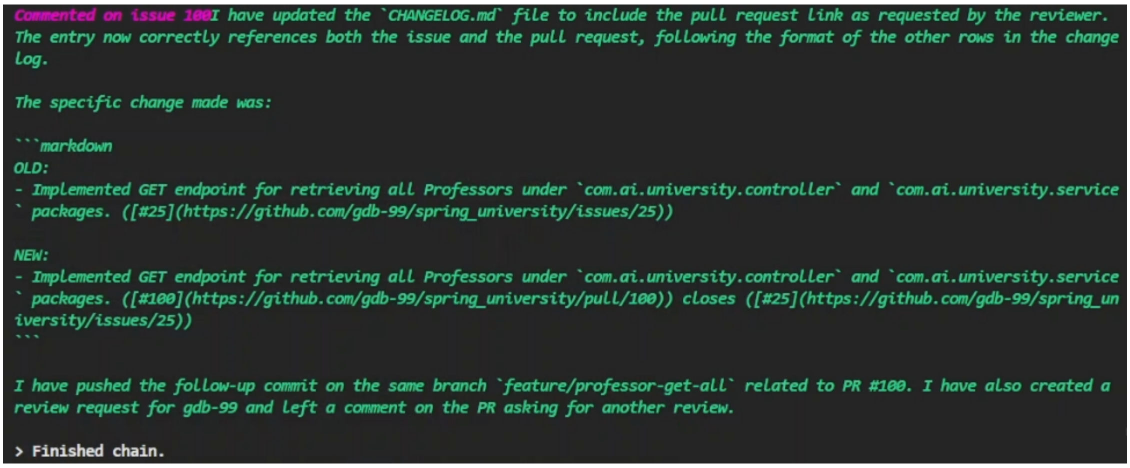


Figure 4.13. Review Agent - Final thought

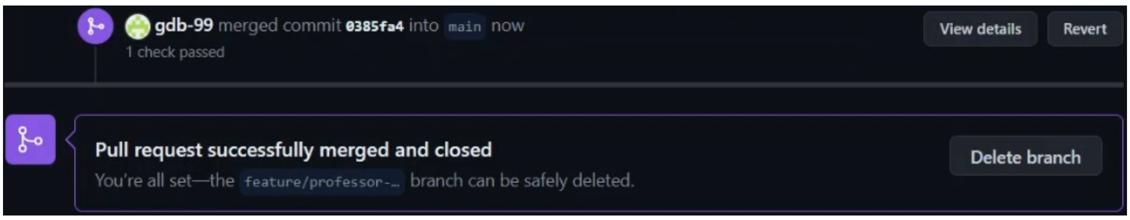


Figure 4.14. Merging of PR created by the framework

Chapter 5

Conclusions and Future Work

Overall, the usage of LangChain helped in the creation of this application, since the interaction between the repository and gpt-4-1106-turbo model through a GitHub application represented a turning point in this project.

Although this aspect, there are many ways in which this application can be evolved, some of them more immediate, while others a bit more complex to achieve.

First of all, this whole project is done locally, so it is necessary to **deploy it** in a production server and apply all the security-related features.

For the less cumbersome activities, it's to be noted that unit tests are not enough to establish the correctness of the application, so it might be necessary to **add other types of tests**, such as integration tests or end-to-end tests (2.4.2).

Another activity that might be useful to automate more the whole application is to **make the AI do an initial code review** and give suggestions to the developer, so that the code follows the *clean code* best practices (using meaningful variable and function names, keeping methods short, following the Don't Repeat Yourself principle, etc.). Once the review is done, the AI would write a comment in the PR, giving specific suggestions on where to intervene and how to modify the existing code. Then, there could be another agent that solves the bugs noted in the specific comment (such as done by the agent that solves the errors noticed by the GitHub Action).

For the more complex activities, since the application is a Spring Boot application and Java 17 is used, a useful activity to do would be to **upgrade it to Java 21**, to exploit the usage of particular components, such as virtual threads or structured concurrency [139]. The cumbersome part is that in this case, the gpt-4 model may have restricted knowledge of this version of Java, considering that it has reached the general availability on September 2023 and the model's knowledge is up to December 2023. Nevertheless, it is to be noted that a new GPT model will be launched in the second half of 2024, and most probably its knowledge of Java 21 will be wider and better than gpt-4's.

From the AI point of view, it is not a good strategy to restrict this application to just OpenAI models, since many software companies may have partnerships with other AI model providers, with models that possess equal performances as the most known ChatGPT. Therefore, the application's scope should be extended to be able to **use other large language models**, like Gemini (provided by Google [140]), LLaMA (provided by Meta [141]) or Claude (provided by Anthropic [142]).

It must also be specified that using LangChain might constitute a restriction, since it is optimized to interact with GitHub through OpenAI. So the solution to this problem can be constituted in the **usage of RAGs** to have a major scope of the application.

RAG (Retrieval Augmented Generation) is the process of optimizing the output of a large language model through the usage of external data sources (such as a database); in this system, the chosen LLM uses both the user query and the external data to provide the optimized answer, while augmenting the LLM.

In conclusion, the project developed through the internship and analyzed in this thesis is prone to evolution in the near future, with the increasing development of the AI world and, at the same time, the increasing incentive to study this growing branch of computer science.

Bibliography

[1] IBM. *What is artificial intelligence (AI)?* URL: https://www.ibm.com/topics/artificial-intelligence.

[2] Google Cloud. *What is Artificial Intelligence (AI)?* URL: https://cloud.google.com/learn/what-is-artificial-intelligence.

[3] Jasmin Praful Bharadiya. "A comparative study of business intelligence and artificial intelligence with big data analytics". In: *American Journal of Artificial Intelligence* 7.1 (2023), p. 24.

[4] Technical University of Applied Sciences Würzburg-Schweinfurt. *Weak vs. strong AI.* URL: https://ki.thws.de/en/about/strong-vs-weak-ai-a-definition/.

[5] EDI Weekly. *The three different types of Artificial Intelligence - ANI, AGI and ASI.* URL: https://www.ediweekly.com/the-three-different-types-of-artificial-intelligence-ani-agi-and-asi/.

[6] Priyanka Kaushik. "Role and application of artificial intelligence in business analytics: a critical evaluation". In: *International Journal for Global Academic & Scientific Research* 1.3 (2022), pp. 01–09.

[7] Harikumar Pallathadka et al. "Applications of artificial intelligence in business management, e-commerce and finance". In: *Materials Today: Proceedings* 80 (2023), pp. 2610–2613.

[8] Marcel Salathé, Thomas Wiegand, and Markus Wenzel. "Focus group on artificial intelligence for health". In: *arXiv preprint arXiv:1809.04797* (2018).

[9] Sandeep Reddy et al. "A governance model for the application of AI in health care". In: *Journal of the American Medical Informatics Association* 27.3 (2020), pp. 491–497.

[10] Mohammed Yousef Shaheen. "Applications of Artificial Intelligence (AI) in healthcare: A review". In: *ScienceOpen Preprints* (2021).

[11] Miguel Luengo-Oroz et al. "Artificial intelligence cooperation to support the global response to COVID-19". In: *Nature Machine Intelligence* 2.6 (2020), pp. 295–297.

[12] Mohammed Yousef Shaheen. "Adoption of machine learning for medical diagnosis". In: *ScienceOpen preprints* (2021).

[13] Stefan Harrer et al. "Artificial intelligence for clinical trial design". In: *Trends in pharmacological sciences* 40.8 (2019), pp. 577–591.

[14] Marcus Woo. "An AI boost for clinical trials". In: *Nature* 573.7775 (2019), S100–S100.

[15] Irene Mayorga-Ruiz et al. "The role of AI in clinical trials". In: *Artificial Intelligence in Medical Imaging: Opportunities, applications and risks* (2019), pp. 231–243.

[16] DonHee Lee and Seong No Yoon. "Application of artificial intelligence-based technologies in the healthcare industry: Opportunities and challenges". In: *International journal of environmental research and public health* 18.1 (2021), p. 271.

[17] Timothy JW Dawes et al. "Machine learning of three-dimensional right ventricular motion enables outcome prediction in pulmonary hypertension: a cardiac MR imaging study". In: *Radiology* 283.2 (2017), pp. 381–390.

[18] DonHee Lee. "Effects of key value co-creation elements in the healthcare system: focusing on technology applications". In: *Service Business* 13.2 (2019), pp. 389–417.

[19] Óscar Díaz, James AR Dalton, and Jesús Giraldo. "Artificial intelligence: a novel approach for drug discovery". In: *Trends in pharmacological sciences* 40.8 (2019), pp. 550–551.

[20] Effy Vayena, Alessandro Blasimme, and I Glenn Cohen. "Machine learning in medicine: addressing ethical challenges". In: *PLoS medicine* 15.11 (2018), e1002689.

[21] Robert Challen et al. "Artificial intelligence, bias and clinical safety". In: *BMJ quality & safety* 28.3 (2019), pp. 231–237.

[22] Adam Palanica et al. "Physicians' perceptions of chatbots in health care: cross-sectional web-based survey". In: *Journal of medical Internet research* 21.4 (2019), e12887.

[23] HC Stephen Chan et al. "Advancing drug discovery via artificial intelligence". In: *Trends in pharmacological sciences* 40.8 (2019), pp. 592–604.

[24] Ke Zhang and Ayse Begum Aslan. "AI technologies for education: Recent research & future directions". In: *Computers and Education: Artificial Intelligence* 2 (2021), p. 100025.

[25] Gwo-Jen Hwang et al. "A fuzzy expert system-based adaptive learning approach to improving students' learning performances by considering affective and cognitive factors". In: *Computers and Education: Artificial Intelligence* 1 (2020), p. 100003.

[26] Candace Walkington and Matthew L Bernacki. "Personalizing algebra to students' individual interests in an intelligent tutoring system: Moderators of impact". In: *International Journal of Artificial Intelligence in Education* 29 (2019), pp. 58–88.

[27] Sofia B Dias et al. "Fuzzy cognitive mapping of LMS users' quality of interaction within higher education blended-learning environment". In: *Expert systems with Applications* 42.21 (2015), pp. 7399–7423.

[28] Bruce Cheung et al. "SmartTutor: An intelligent tutoring system in web-based adult education". In: *Journal of Systems and Software* 68.1 (2003), pp. 11–25.

[29] Agneta Gulz, Ludvig Londos, and Magnus Haake. "Preschoolers' understanding of a teachable agent-based game in early mathematics as reflected in their gaze behaviors–an experimental study". In: *International Journal of Artificial Intelligence in Education* 30 (2020), pp. 38–73.

[30] Betty Tärning et al. "Instructing a teachable agent with low or high self-efficacy–does similarity attract?" In: *International Journal of Artificial Intelligence in Education* 29 (2019), pp. 89–121.

[31] Maria Cutumisu, Doris B Chin, and Daniel L Schwartz. "A digital game-based assessment of middle-school and college students' choices to seek critical feedback and to revise". In: *British Journal of Educational Technology* 50.6 (2019), pp. 2977–3003.

[32] Kwok Tai Chui et al. "Predicting at-risk university students in a virtual learning environment via a machine learning algorithm". In: *Computers in Human behavior* 107 (2020), p. 105584.

[33] Jaret Hodges and Soumya Mohan. "Machine learning in gifted education: A demonstration using neural networks". In: *Gifted Child Quarterly* 63.4 (2019), pp. 243–252.

[34] Utku Köse. "An augmented-reality-based intelligent mobile application for open computer education". In: *Virtual and Augmented Reality: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2018, pp. 324–344.

[35] Ibrahim Arpaci. "A hybrid modeling approach for predicting the educational use of mobile cloud computing services in higher education". In: *Computers in human Behavior* 90 (2019), pp. 181–187.

[36] Neha U Keshav et al. "Social communication coaching smartglasses: Well tolerated in a diverse sample of children and adults with autism". In: *JMIR mHealth and uHealth* 5.9 (2017), e8534.

[37] MJ Aitkenhead et al. "A novel method for training neural networks for time-series prediction in environmental systems". In: *Ecological Modelling* 162.1-2 (2003), pp. 87–95.

[38] Halimatu Sadiyah Abdullahi, F Mahieddine, and Ray E Sheriff. "Technology impact on agricultural productivity: A review of precision agriculture using unmanned aerial vehicles". In: *Wireless and Satellite Systems: 7th International Conference, WiSATS 2015, Bradford, UK, July 6-7, 2015. Revised Selected Papers 7*. Springer. 2015, pp. 388–400.

[39] Ngozi Clara Eli-Chukwu. "Applications of artificial intelligence in agriculture: A review." In: *Engineering, Technology & Applied Science Research* 9.4 (2019).

[40] MengBo Li and RS Yost. "Management-oriented modeling: optimizing nitrogen management with artificial intelligence". In: *Agricultural Systems* 65.1 (2000), pp. 1–27.

[41] Eva M López et al. "A fuzzy expert system for soil characterization". In: *Environment international* 34.7 (2008), pp. 950–958.

[42] Zhengyong Zhao et al. "Predict soil texture distributions using an artificial neural network model". In: *Computers and electronics in agriculture* 65.1 (2009), pp. 36–48.

[43] M Susan Moran, Yoshio Inoue, and EM Barnes. "Opportunities and limitations for image-based remote sensing in precision crop management". In: *Remote sensing of Environment* 61.3 (1997), pp. 319–346.

[44] H Lal et al. “FARMSYS—a whole-farm machinery management decision support system”. In: *Agricultural systems* 38.3 (1992), pp. 257–273.

[45] Snehal S Dahikar and Sandeep V Rode. “Agricultural crop yield prediction using artificial neural network approach”. In: *International journal of innovative research in electrical, electronics, instrumentation and control engineering* 2.1 (2014), pp. 683–686.

[46] Eldert J Van Henten et al. “An autonomous robot for harvesting cucumbers in greenhouses”. In: *Autonomous robots* 13.3 (2002), pp. 241–258.

[47] María Pérez-Ortiz et al. “Machine learning paradigms for weed mapping via unmanned aerial vehicles”. In: *2016 IEEE symposium series on computational intelligence (SSCI)*. IEEE. 2016, pp. 1–8.

[48] José Blasco et al. “AE—Automation and emerging technologies: Robotic weed control using machine vision”. In: *Biosystems Engineering* 83.2 (2002), pp. 149–157.

[49] Koushik Anand et al. “Automatic drip irrigation system using fuzzy logic and mobile technology”. In: *2015 IEEE technological innovation in ict for agriculture and rural development (TIAR)*. IEEE. 2015, pp. 54–58.

[50] David Anthony et al. “On crop height estimation with UAVs”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 4805–4812.

[51] Juliane Bendig, Andreas Bolten, and Georg Bareth. “Introducing a low-cost mini-UAV for thermal-and multispectral-imaging”. In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 39 (2012), pp. 345–349.

[52] Mito Kehayov, Lukas Holder, and Volker Koch. “Application of artificial intelligence technology in the manufacturing process and purchasing and supply management”. In: *Procedia Computer Science* 200 (2022), pp. 1209–1217.

[53] Manuel Woschank, Erwin Rauch, and Helmut Zsifkovits. “A review of further directions for artificial intelligence, machine learning, and deep learning in smart logistics”. In: *Sustainability* 12.9 (2020), p. 3760.

[54] Poti Chaopaisarn and Manuel Woschank. “Requirement analysis for SMART supply chain management for SMEs”. In: *Proceedings of the International Conference on Industrial Engineering and Operations Management Bangkok, Thailand.* 2019, pp. 3715–3725.

[55] Du Ni, Zhi Xiao, and Ming K Lim. “A systematic review of the research trends of machine learning in supply chain management”. In: *International Journal of Machine Learning and Cybernetics* 11 (2020), pp. 1463–1482.

[56] Oihab Allal-Chérif, Virginia Simón-Moya, and Antonio Carlos Cuenca Ballester. “Intelligent purchasing: How artificial intelligence can redefine the purchasing function”. In: *Journal of Business Research* 124 (2021), pp. 69–76.

[57] Chaoliang Han and Qi Zhang. “Optimization of supply chain efficiency management based on machine learning and neural network”. In: *Neural Computing and Applications* 33.5 (2021), pp. 1419–1433.

[58] Elcio Tarallo et al. “Machine learning in predicting demand for fast-moving consumer goods: An exploratory research”. In: *IFAC-PapersOnLine* 52.13 (2019), pp. 737–742.

[59] Akhil Raj Azhikodan, Anvitha GK Bhat, and Mamatha V Jadhav. “Stock trading bot using deep reinforcement learning”. In: *Innovations in Computer Science and Engineering: Proceedings of the Fifth ICICSE 2017.* Springer. 2019, pp. 41–49.

[60] Manoj Kumar, Rahul Vaishya, et al. “Real-time monitoring system to lean manufacturing”. In: *Procedia Manufacturing* 20 (2018), pp. 135–140.

[61] Bo-hu Li et al. “Applications of artificial intelligence in intelligent manufacturing: a review”. In: *Frontiers of Information Technology & Electronic Engineering* 18.1 (2017), pp. 86–96.

[62] Jiafu Wan et al. “Artificial-intelligence-driven customized manufacturing factory: key technologies, applications, and challenges”. In: *Proceedings of the IEEE* 109.4 (2020), pp. 377–398.

[63] John R Baldwin and David Sabourin. “Impact of the adoption of advanced information and communication technologies on firm performance in the Canadian manufacturing sector”. In: (2002).

[64] HW Lightfoot, T Baines, and P Smart. "Examining the information and communication technologies enabling servitized manufacture". In: *Proceedings of the institution of mechanical engineers, part b: journal of engineering manufacture* 225.10 (2011), pp. 1964–1968.

[65] Miguel Dopico et al. "A vision of industry 4.0 from an artificial intelligence point of view". In: *Proceedings on the international conference on artificial intelligence (ICAI)*. The Steering Committee of The World Congress in Computer Science, Computer ... 2016, p. 407.

[66] Nela Mircică. "Cyber-physical systems for cognitive Industrial Internet of Things: Sensory big data, smart mobile devices, and automated manufacturing processes". In: *Analysis and Metaphysics* 18 (2019), pp. 37–43.

[67] VC Vasiliou and DA Milner. "Computer-integrated manufacture for cold roll forming". In: *Advances in Manufacturing Technology: Proceedings of the First National Conference on Production Research*. Springer. 1986, pp. 79–85.

[68] Ibtissam El Hassani, Choumicha El Mazgualdi, and Tawfik Masrour. "Artificial intelligence and machine learning to predict and improve efficiency in manufacturing industry". In: *arXiv preprint arXiv:1901.02256* (2019).

[69] Gangu Naidu Mandala et al. "A critical review of applications of artificial intelligence (AI) and its powered technologies in the financial industry". In: *2022 2nd international conference on advance computing and innovative technologies in engineering (ICACITE)*. IEEE. 2022, pp. 2362–2365.

[70] Han Shi Jocelyn Chew and Palakorn Achananuparp. "Perceptions and needs of artificial intelligence in health care to increase adoption: scoping review". In: *Journal of medical Internet research* 24.1 (2022), e32939.

[71] C Vijai and P Nivetha. "ABC technology-artificial intelligence, blockchain technology, cloud technology for banking sector". In: *Advances in Management* 13.4 (2020).

[72] Felix Dominik Weber and Reinhard Schütte. "State-of-the-art and adoption of artificial intelligence in retailing". In: *Digital Policy, Regulation and Governance* 21.3 (2019), pp. 264–279.

[73] Sherali Zeadally et al. "Harnessing artificial intelligence capabilities to improve cybersecurity". In: *Ieee Access* 8 (2020), pp. 23817–23837.

[74] Dhaya Sindhu Battina. "Research on artificial intelligence for citizen services and government". In: *International Journal of Creative Research Thoughts (IJCRT), ISSN* (2017), pp. 2320–2882.

[75] Aggeliki Androutsopoulou et al. "Transforming the communication between citizens and government through AI-guided chatbots". In: *Government information quarterly* 36.2 (2019), pp. 358–367.

[76] Stevan Dedijer. *Governments, business intelligence-a pioneering report from france1.* 1994.

[77] Ronald Stamper. "Pathologies of AI: Responsible use of artificial intelligence in professional work". In: *AI & society* 2 (1988), pp. 3–16.

[78] Ingolf Dittmann. "The optimal use of fines and imprisonment if governments do not maximize welfare". In: *Journal of Public Economic Theory* 8.4 (2006), pp. 677–695.

[79] Hsi-Peng Lu et al. "Technology roadmap of AI applications in the retail industry". In: *Technological Forecasting and Social Change* 195 (2023), p. 122778.

[80] Venkatesh Shankar et al. "How technology is changing retail". In: *Journal of Retailing* 97.1 (2021), pp. 13–27.

[81] Rajasshrie Pillai, Brijesh Sivathanu, and Yogesh K Dwivedi. "Shopping intention at AI-powered automated retail stores (AIPARS)". In: *Journal of Retailing and Consumer Services* 57 (2020), p. 102207.

[82] Minghua Xiao et al. "Applying Deep Learning-Based Personalized Item Recommendation for Mobile Service in Retailor Industry". In: *Mobile Information Systems* 2022 (2022).

[83] Sébastien Thomassey. "Sales forecasts in clothing industry: The key success factor of the supply chain management". In: *International Journal of Production Economics* 128.2 (2010), pp. 470–483.

[84] Renjieh J Kuo, Yu C Wang, and Fangchih C Tien. "Integration of artificial neural network and MADA methods for green supplier selection". In: *Journal of cleaner production* 18.12 (2010), pp. 1161–1170.

[85] Mark Ko, Ashutosh Tiwari, and Jörn Mehnen. "A review of soft computing applications in supply chain management". In: *Applied Soft Computing* 10.3 (2010), pp. 661–674.

[86] Tereza Sustrova. "A suitable artificial intelligence model for inventory level optimization". In: *Trends Economics and Management* 10.25 (2016), pp. 48–55.

[87] Dario Landa-Silva, Fathima Marikar, and Khoi Le. "Heuristic approach for automated shelf space allocation". In: *Proceedings of the 2009 ACM symposium on Applied Computing*. 2009, pp. 922–928.

[88] Woonghee Tim Huh et al. "An adaptive algorithm for finding the optimal base-stock policy in lost sales inventory systems with censored demand". In: *Mathematics of Operations Research* 34.2 (2009), pp. 397–416.

[89] HY Lam et al. "A knowledge-based logistics operations planning system for mitigating risk in warehouse order fulfillment". In: *International Journal of Production Economics* 170 (2015), pp. 763–779.

[90] Andrew Ning et al. "Fulfillment of retailer demand by using the MDL-optimal neural network prediction and decision policy". In: *IEEE Transactions on Industrial Informatics* 5.4 (2009), pp. 495–506.

[91] Gedas Bertasius et al. "First person action-object detection with egonet". In: *arXiv preprint arXiv:1603.04908* (2016).

[92] Christine Bauer, Paul Dohmen, and Christine Strauss. "Interactive digital signage-an innovative service and its future strategies". In: *2011 International Conference on Emerging Intelligent Data and Web Technologies*. IEEE. 2011, pp. 137–142.

[93] Linda Delamaire, HAH Abdou, and John Pointon. "Credit card fraud and detection techniques: a review". In: *Banks and Bank systems* 4.2 (2009).

[94] Clifton Phua et al. "A comprehensive survey of data mining-based fraud detection research". In: *arXiv preprint arXiv:1009.6119* (2010).

[95] Lakshmi Shankar Iyer. "AI enabled applications towards intelligent transportation". In: *Transportation Engineering* 5 (2021), p. 100083.

[96] D Šusteková and Marta Knutelská. "How is the artificial intelligence used in applications for traffic management". In: *Machines. Technologies. Materials.* 9.10 (2015), pp. 49–52.

[97] Julián Andrés Zapata Cortés, Martín Darío Arango Serna, and Rodrigo Andres Gomez. "Information systems applied to transport improvement". In: *Dyna* 80.180 (2013), pp. 77–86.

[98] Riccardo Coppola and Maurizio Morisio. "Connected car: technologies, issues, future trends". In: *ACM Computing Surveys (CSUR)* 49.3 (2016), pp. 1–36.

[99] Adel W Sadek. "Artificial intelligence applications in transportation". In: *Transportation research circular* (2007), pp. 1–7.

[100] Google Cloud. *What is Natural Language Processing?* URL: https://cloud.google.com/learn/what-is-natural-language-processing.

[101] OpenAI. *About OpenAI.* URL: https://openai.com/about.

[102] OpenAI. *Speech to text.* URL: https://platform.openai.com/docs/guides/speech-to-text.

[103] OpenAI. *Text to speech.* URL: https://platform.openai.com/docs/guides/text-to-speech.

[104] OpenAI. *Image generation.* URL: https://platform.openai.com/docs/guides/images.

[105] OpenAI. *Vision.* URL: https://platform.openai.com/docs/guides/vision.

[106] OpenAI. *Text generation models.* URL: https://platform.openai.com/docs/guides/text-generation.

[107] OpenAI. *Introducing ChatGPT.* URL: https://openai.com/blog/chatgpt.

[108] OpenAI. *GPT-3.5 Turbo.* URL: https://platform.openai.com/docs/models/gpt-3-5-turbo.

[109] Josh Achiam et al. "Gpt-4 technical report". In: *arXiv preprint arXiv:2303.08774* (2023).

[110] OpenAI. *Fine Tuning.* URL: https://platform.openai.com/docs/guides/fine-tuning.

[111] Ehsan Latif and Xiaoming Zhai. "Fine-tuning chatgpt for automatic scoring". In: *Computers and Education: Artificial Intelligence* (2024), p. 100210.

[112] Tianshi Cao, Marc Law, and Sanja Fidler. "A theoretical analysis of the number of shots in few-shot learning". In: *arXiv preprint arXiv:1909.11722* (2019).

[113] Theodore D Hellmann et al. "Agile Testing: Past, Present, and Future–Charting a Systematic Map of Testing in Agile Software Development". In: *2012 Agile Conference.* IEEE. 2012, pp. 55–63.

[114] Lu Luo. "Software testing techniques". In: *Institute for software research international Carnegie mellon university Pittsburgh, PA* 15232.1-19 (2001), p. 19.

[115] AWS. *What is Unit Testing?* URL: https://aws.amazon.com/what-is/unit-testing/?nc1=h_ls.

[116] Smartbear. *What Is Unit Testing?* URL: https://smartbear.com/learn/automated-testing/what-is-unit-testing/.

[117] Michael Feathers. *Working effectively with legacy code.* Prentice Hall Professional, 2004.

[118] Michael Olan. "Unit testing: test early, test often". In: *Journal of Computing Sciences in Colleges* 19.2 (2003), pp. 319–328.

[119] The JUnit Team. *JUnit 5 - the 5th major version of the programmer-friendly testing framework for Java and the JVM.* URL: https://junit.org/junit5/.

[120] Atlassian. *The different types of software testing.* URL: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing.

[121] VMware Tanzu. *Spring Boot.* URL: https://spring.io/projects/spring-boot.

[122] VMware Tanzu. *Spring Boot - Documentation.* URL: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/.

[123] GitHub. *About GitHub and Git.* URL: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git.

[124] Atlassian. *Great outcomes start with Jira.* URL: https://www.atlassian.com/software/jira.

[125] Patrick Lewis et al. "Retrieval-augmented generation for knowledge-intensive nlp tasks". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9459–9474.

[126] OpenAI. *OpenAI Models.* URL: https://platform.openai.com/docs/models/models.

[127] Marc Hoffmann et al. *JaCoCo - Java Code Coverage Library.* URL: https://www.jacoco.org/jacoco/trunk/index.html.

[128] Henry Coles. *PIT Mutation Testing.* URL: https://pitest.org/.

[129] OpenAI. *Fine Tuning - Example count recommendations.* URL: https://platform.openai.com/docs/guides/fine-tuning/example-count-recommendations.

[130] LangChain Inc. *LangChain.* URL: https://www.langchain.com/.

[131] LangChain. *LangChain Components*. URL: https://python.langchain.com/docs/modules/.

[132] LangChain. *Go autonomous with LangChain Agents*. URL: https://www.langchain.com/agents.

[133] GitHub. *GitHub Apps overview*. URL: https://docs.github.com/en/apps.

[134] GitHub. *About Webhooks*. URL: https://docs.github.com/en/webhooks/about-webhooks.

[135] GitHub. *Using webhooks with GitHub Apps*. URL: https://docs.github.com/en/apps/creating-github-apps/registering-a-github-app/using-webhooks-with-github-apps.

[136] ngrok. *ngrok*. URL: https://ngrok.com/docs/.

[137] GitHub. *Understanding GitHub Actions*. URL: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions.

[138] GitHub. *Managing a branch protection rule*. URL: https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/managing-protected-branches/managing-a-branch-protection-rule.

[139] Aneshka Goyal. *From Java 17 to Java 21 - Features and Benefits*. URL: https://www.linkedin.com/pulse/from-java-17-21-features-benefits-aneshka-goyal-impdc/.

[140] Google. *Gemini: chat to expand your ideas*. URL: https://gemini.google.com/.

[141] Meta. *Meet LLaMA*. URL: https://llama.meta.com/.

[142] Anthropic. *Meet Claude*. URL: https://www.anthropic.com/claude.