POLITECNICO DI TORINO

Master's Degree in Computer Engineer

Master's Degree Thesis

A New Approach to Programming: AI Agents, LLMs, and an SQL Generation Case Study

Supervisors

Prof. Paolo GARZA

Dott. Giovanni VANINI

Candidate

Arturo ADELFIO

April 2025

Abstract

The rise of Large Language Models (LLMs) and AI agents is transforming software development, introducing new paradigms in automation and human-machine collaboration. This thesis, conducted in collaboration with Poseidon, a company specializing in low-code automation, explores the state of the art in LLMs and AI agents, analyzing their architectures, datasets, and benchmarks, with a focus on programming efficiency and business workflows. Building on this foundation, the study presents a case study in which an SQL agent is developed and evaluated for natural language database querying, enhancing user interaction. The evaluation employs both standard and custom metrics, utilizing a specifically designed dataset to assess the model's performance comprehensively and demonstrate its practical implications for accessibility and efficiency. By integrating theoretical analysis with practical experimentation, this research offers a broad perspective on the capabilities and limitations of LLM-driven automation in software engineering and business applications.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
: Artificial Intelligence

AIAYN
: Attention Is All You Need

BPTT
: Backpropagation Through Time

CBOW
: Continuous Bag of Words

DP
: Deep Learning

GPT
: Generative Pre-trained Transformer

LLMs
: Large Language Models

LSTM
: Long Short-Term Memory

ML
: Machine Learning

NLP
: Natural Language Processing

OOV

Out of Vocabulary

RNN

Recurrent Neural Network

RPA

Robotic Process Automation

SMT

Statistical Machine Translation

T5

Text-to-Text Transfer Transformer

Chapter 1

Introduction

1.1 Motivation

The adoption of tools based on Large Language Models (LLMs) is radically transforming the way people interact with technology across various sectors, particularly in software development. The possibility that these models could serve not only as assistants but even as substitutes for programmers is a topic that deserves in-depth analysis.

The idea behind this thesis emerged from a collaboration with Poseidon SB, a Sicily-based company specializing in the automation and digitalization of business processes through low-code technologies. With a strong focus on minimizing the need for traditional programming, Poseidon is particularly interested in how generative AI can enhance or extend low-code solutions, further simplifying software development and business automation. The quick development pace of AI tools provides an opportunity to align Poseidon's expertise in digital transformation with the latest innovations in LLMs and AI agents.

This context is further reinforced by two distinct yet closely related phenomena that highlight the growing influence of AI in software development.

The first is a striking demonstration of AI's capabilities. A widely shared online video [1] showed an 8-year-old girl who, assisted with CursorAI, managed to create a website in just a few minutes by simply giving textual commands, without any prior programming knowledge.

The second trend concerns the decline of traditional programming support websites and forums, such as Stack Overflow [2]. Since the release of GPT-3 and AI-based coding assistants, such as GitHub Copilot, website visits to developer communities have fallen sharply. Moreover, in addition to these two phenomena, there is the development of AI agents, which, due to their ability to be more

proactive compared to typical LLM models, makes them more capable of executing sophisticated tasks more independently and adaptable. While LLMs and code completion tools have already demonstrated that they can be wonderful assistants to programmers, AI agents seem to be able to turn into full-fledged, independent tools that can manage entire workflows, from coding to the management of complex systems.

Building on this premise, this study, conducted in collaboration with Poseidon SB aims to analyze the latest developments in the field of LLMs and AI agents. The structure of this work includes an introductory analysis of generative AI, with a focus on its operating principles and main applications. Next, the role of Large Language Models will be explored, emphasizing their impact on productivity and digital transformation. Finally, a study on AI agents will be conducted, analyzing their role in business automation and future development potential, supported by a practical case study. In an era where AI is profoundly redefining the boundaries between human and artificial intelligence, understanding its functioning and potential is not just a technological necessity but a crucial challenge for the future of society and the world of work.

1.2 Context

Arthur C. Clarke once said: *"Any sufficiently advanced technology is indistinguishable from magic."* This quote accurately describes the impact of AI in today's world. A technology that, due to its rapid development and increasingly sophisticated capabilities, seems to challenge the limits of our understanding, approaching something similar to magic. While AI was once perceived as a distant concept relegated to science fiction, it is now a pervasive reality that influences every aspect of our lives, from information management to communication, from creativity to decision-making processes. Among the many advancements in AI, one particularly revolutionary field is generative AI, which has led to the emergence of advanced language models such as Large Language Models. These systems, based on sophisticated neural architectures and trained on vast amounts of data, can generate text, translate languages, answer questions, support creative writing, and even produce artistic works. Their impact has been disruptive, redefining our relationship with technology and giving machines a role that goes beyond merely executing instructions. These tools have brought us to a new era of productivity and process automation. This is not just an acceleration of existing processes but a redefinition of operational and cultural dynamics, introducing new frontiers in human-machine interaction.

This revolution raises fundamental questions: why does AI generate so much enthusiasm? One of the most evident answers lies in its promise to optimize time

and resources. In a world characterized by increasingly fast-paced rhythms, the idea of delegating repetitive and low-value tasks to an AI is highly appealing. Intelligent automation and strategic decision support reduce human workload, allowing people to focus on more creative, analytical, and strategic activities. AI thus emerges not only as a tool for efficiency but as a factor of social and economic transformation.

The implications of this technology go beyond improving individual productivity; they extend to the macroeconomic level. The growing interest in AI is evident in the financial and industrial sectors, where companies such as Nvidia, ASML, and TSMC, leaders in producing essential AI hardware, have become among the most influential players globally. However, the focus is not limited to publicly traded companies. Organizations like OpenAI and many emerging startups are shaping the future of artificial intelligence without necessarily answering to financial markets, accelerating the development of increasingly powerful models.

This race for innovation is not just economic but also geopolitical. AI has become a battleground among global powers, with strategic implications that go beyond technology itself. The recent case of DeepSeek, the Chinese AI model that captured attention in late January 2025, demonstrates how the development of national LLMs has become a key factor in technological sovereignty policies. Governments are investing massive resources to secure control over these technologies, outlining a future in which AI will not only power economic growth but also play a crucial role in international and political power dynamics. In addition, the rapid progress of AI-powered technologies is redesigning our notion of intelligence, increasingly closing the gap between human and machine capabilities. It is no longer just about implementing new technological solutions but also understanding a new way of interacting with them. Among the most tangible impacts of AI, its effect on the labor market is one of the most significant, especially in the IT and automation sectors, where AI can play a decisive role. One of the most significant innovations is the emergence of AI agents, intelligent systems capable of functioning independently, making choices, and communicating with other tools, users, or even with each other. Unlike simple automation software, these agents are able to adjust according to the circumstances and act more independently. This change is so dramatic that, according to Jensen Huang, CEO of NVIDIA, "The IT department of every company will become the HR department for AI agents." This statement highlights the growing centrality of these systems in the future of work, where the seamless integration of artificial intelligence and human oversight will be key to the success of businesses and global economies.

This is the context in which this thesis is set, an era of rapid and profound transformations, where generative artificial intelligence is redefining entire industries. As Kranzberg's law states, technology is neither good nor bad, nor is it neutral; its impact depends on how we choose to develop, adopt, and regulate it. Therefore, it is not enough to observe its potential; it is essential to understand its broader

implications.

Moreover, given the extraordinary pace of advancement in this field, any analysis or consideration is at risk of becoming obsolete quickly. Therefore, the findings presented in this thesis should be regarded as a snapshot of the current state of AI-driven development, acknowledging the necessity from—both a business and academic perspective—to carefully assess these technologies at a given moment in time.

This section aims to outline, concisely and clearly, the framework within which this thesis takes place, offering a perspective that goes beyond purely technical aspects.

1.3 Problem Statement and Objectives

This thesis will examine the potential for using Large Language Models (LLMs) to augment or even replace tasks typically performed by programmers with a focus on the role of AI agents. The research aims to investigate the capabilities of the LLMs to automate software development, providing a broad overview of the architectures, datasets, and benchmarks used to evaluate their performance.

In order to support this research, a case study is conducted in which an LLM agent is designed to build and execute database queries given a natural language input. The SQL case study serves as a framework to assess whether the practical application results are aligned with the theoretical analysis. This system will be experimentally tested to evaluate its efficacy and potential impact on querying and data management. In this way, it will support research on practical LLM applications in software development and programming task automation.

1.4 Poseidon SB

Poseidon SB is a consulting company with a strong specialization in business process automation and digitalization, operating in parallel within the fields of Digital Procurement and Human Resources. Founded in Sicily, Poseidon SB stands out for its commitment to innovation and the development of advanced technological solutions, collaborating with medium and large companies at both national and international levels. Within a few years, it has consolidated its position in the market, building a highly qualified team of over 40 professionals and contributing to the digital transformation of more than 30 businesses. The company has two main offices, located in Catania and Palermo. As a benefit corporation, Poseidon SB goes beyond the technological domain, aiming to generate a positive impact on the Sicilian region by promoting talent development and collaborating with academic and training institutions such as E.M. Associazione ARCES. This commitment

translates into initiatives aimed at facilitating young professionals' entry into the job market and strengthening the local economy through innovation and digitalization projects. Poseidon SB provides consulting services in various strategic areas, including operational process optimization, design, and implementation of advanced digital solutions, with strong expertise in using mostly Low-Code tools to support business infrastructures. The company is also actively involved in defining and developing solutions based on Artificial Intelligence, both traditional and generative, with a focus on digital transformation oriented towards efficiency and competitiveness. As anticipated, a particularly relevant sector for Poseidon is Robotic Process Automation (RPA), where it develops solutions capable of automating repetitive tasks, improving efficiency, and reducing operational costs for businesses. Looking ahead, Poseidon is expanding its focus toward Artificial Intelligence and Smart Automation, to develop increasingly advanced tools for business process digitalization and automation. This commitment reflects the company's ambition to position itself as a key player in the evolution of AI technologies, actively contributing to industry transformation and the creation of innovative solutions for enterprises.

Chapter 2

State of the Art

2.1 What is AI?

Artificial Intelligence (AI) is a field of computer science that studies the creation of systems capable of performing tasks that, if done by humans, would require intelligence. However, the concept of Artificial Intelligence has been defined in different ways over time, often reflecting the different views and goals of research in this field. As noted by Pei Wang (2007) in a scientific paper titled "What do you mean by AI?" [3], one of the main problems in AI studies is the difficulty of defining its limits clearly. The debate is not only about the best way to build intelligent systems but also about what "intelligence" actually means in a computational context. Wang points out that there is no universally accepted definition of AI, but rather a set of different approaches that emphasize various aspects: structure, behavior, function, capability, or principle. Regardless of how it is defined, there is no doubt that this technology is one of the most impactful and revolutionary events of the 21st century. Its applications are radically transforming every sector, from medicine to finance, from transportation to creativity, thanks to automation, predictive analysis, and large-scale customization. AI is now at the core of systems that support medical diagnosis, virtual assistants, self-driving cars, instant translations, and much more.

Although today it is a well-established reality, the concept of AI and its foundations goes back to much earlier times, in the 1940s and 1950s, when scientists and researchers began to explore the potential of machines capable of simulating intelligent human activities. One of the first insights into the connection between the human brain and computers appeared in the famous 1943 paper by Warren McCulloch and Walter Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity" [4]. In this pioneering work, the authors proposed a mathematical model of how neural networks function in the brain, drawing parallels between neural

activity and electronic circuits. This paper not only laid the foundation for what would later become Artificial Intelligence but also inspired future innovations in the fields of perceptron and neural networks, which are, today, the modern foundations of AI.

Among the most influential characters during the early days of AI development, one of the most important is surely Alan Turing, a mathematician and pioneer of computer science. His influential 1950 paper, "Computing Machinery and Intelligence" [5], posed the question that became central in the evolution, raising a crucial issue: can a machine, through its behavior and interaction, imitate intelligence in a way that is indistinguishable from human intelligence? In order to address this question, he suggested what he had called the Imitation Game, or the Turing Test (Turing, 1950). The experiment aimed to assess whether a machine could sustain a conversation with a human without the human realizing they were talking to a machine. For many years, this idea was considered an important benchmark, setting that intelligence could be measured not by a system's physical structure but by its functional and behavioral capabilities.

However, the test was only passed decades later, in 2014, when a chatbot named Eugene Goostman convinced 33% of human judges at the Royal Society that it was a 13-year-old boy, marking the first official passing of the test. The event has been widely debated, with critics arguing that the chatbot employed deceptive tactics by posing as a non-native English speaking teenager, effectively lowering judges' expectations. This event sparked a major debate about whether the Turing Test is still a valid measure of artificial intelligence.

Though there were some years of theoretical discussions concerning artificial intelligence, it was only in the year 1956, during a summer workshop at Dartmouth College, that the term "Artificial Intelligence" was officially used by Professor John McCarthy [6]. McCarthy, among other scientists such as Marvin Minsky, Nathaniel Rochester, and Claude Shannon, laid the foundations for the discipline, defining AI as the attempt to "make machines do things that, when done by humans, are said to require intelligence." This moment marks the formal birth of AI as a research field, separating it from mere computing theory and moving it toward the applied science for building machines capable of performing tasks typical of human intelligence.

Over the following decades, AI research went through various phases of progress and stagnation, with moments of excitement and periods of "AI winters" when expectations were not met. However, thanks to improvements in computing power, the increasing availability of data, and advancements in algorithm design, AI has grown exponentially since the 2000s. This development led to the creation of multiple subfields and methodological approaches, each of which contributed to shaping AI as we know it today. As mentioned earlier, artificial intelligence is not a single, unified technology but a broad concept that includes different methods and approaches. One of the most relevant areas is Machine Learning

(ML), which is a collection of techniques that allow machines to learn from data and improve their performance without being explicitly programmed. Within ML, Deep Learning (DL) has taken a central role, using deep neural networks to model complex relationships between data and enabling advances in fields such as computer vision, self-driving cars, and speech recognition.

A key distinction within artificial intelligence is between traditional AI and generative AI. While classical techniques focus on data analysis, classification, and prediction, generative AI is designed to create new data and content independently. This category of AI uses probabilistic models and advanced neural networks to generate text, images, music, and other digital content, standing out for its ability to process information and produce results that are not just reworkings of existing input but entirely new creations. This represents a major conceptual leap, greatly expanding the possible applications of AI. Artificial intelligence application areas have developed from experimental environments to complete solutions.

In the early decades of existence, in fact, AI accomplishment was largely adapted and applied to specific and limited-scale problems, such as chess, with world chess champion Garry Kasparov defeated by Deep Blue in 1997 as a highlight achievement in the field [7]. Similarly, early AI algorithms focused on classification and optimization tasks, such as character recognition and automated planning. With machine learning and deep learning, applications began to trend towards more complex situations.

Today, AI is widely used in critical fields such as medicine, where deep learning algorithms are employed for early disease detection through medical image analysis (Esteva et al., 2017), and in finance, where algorithmic trading systems process large volumes of data in real-time to optimize investments and reduce risk (Artificial Intelligence Applied to Stock Market Trading: A Review, F. Ferreira et al., 2021). AI is also central to self-driving systems, climate change prediction models, and cybersecurity, where it is used to detect cyber threats in real-time. A particularly groundbreaking field is Natural Language Processing (NLP), which has made it possible to develop intelligent virtual assistants, automatic translators, and semantic analysis tools.

In this process of uninterrupted historical transformation, Large Language Models have represented a revolutionary step in artificial intelligence history. LLMs pushed the boundaries of machine abilities beyond any limits, revolutionizing human-computer interaction as never before. A milestone in this evolution was the advent of GPT-3, released in June 2020 by OpenAI, which showed an unprecedented ability to generate and interpret natural language, respond to challenging questions, and support an immense variety of creative and analytical tasks, that were usually tasks considered unsuitable for computers. This innovation not only demonstrated the unprecedented progress of AI but also revealed new, unseen applications and implications that are still influencing business, productivity, and innovation

transformation.

Beyond merely processing information, LLMs possess remarkably strong potential to understand, generate, and develop sophisticated ideas so that they can assist with coding, writing, business development, and studies. Their capacity to generate knowledge and provide advanced support is fundamentally transforming the way we work and innovate.

Recognizing this shift, Ethan Mollick, in his best-selling book Co-Intelligence: Living and Working with AI (2023), describes LLMs as co-workers rather than mere tools. He emphasizes how they have moved beyond simple automation to become active collaborators in various domains, fostering a new era of human-machine partnership. As AI continues to evolve, these advancements unlock unprecedented opportunities, expanding the horizon of productivity, creativity, and problem-solving.

2.1.1 AI & Programming

One of the major areas impacted by developments in AI is the area of coding. This specific sector is undergoing a major transformation due to the advancements in the LLMs field. These new developments are reshaping the way programmers interact with code, making certain tasks more efficient while introducing new challenges and opportunities. AI-assisted coding is something that has taken place in years now, since machine learning models have been used in software development to autocomplete features or to intelligent code refactoring tools. However, the introduction of generative AI and models like OpenAI's Codex, DeepMind's AlphaCode, and Claude's reasoning-based coding assistants represents a pivotal shift in software engineering activities.

Initially, AI technology in software development focused on making the developer more productive with the help of capabilities like syntax completion, debugging, and static analysis. All these mechanisms, integrated into IDEs, helped programmers find errors and improve code quality. Recently, instead, the latest coding fine-tuned models like Codex (on which GitHub Copilot is built) took it to an even newer level and created complete functions or scripts based on text descriptions. This shift from "text-to-code" has introduced a new coding paradigm where programmers can describe the desired behavior and the functionality needed for their application and observe the AI complete the job by generating the code for them with a fair degree of success.

Furthermore, LLMs are becoming increasingly comprehensive, evolving to cover all aspects of software engineering, from code generation and debugging to optimization, refactoring, and security enforcement. Their growing capabilities are starting to enable them to assist in every stage of the development process.

In this context, it could be useful to analyze the historical evolution process of

programming tools and techniques. In fact, coding has always been paired with tools that abstract complexity and make development easier and more accessible. The transition from punch cards toward interactive terminals in the 1960s, the introduction of high-level languages in the 1980s, and the proliferation of open-source libraries and frameworks in the 2000s each expanded the pool of people who could contribute to software development. Currently, AI-powered coding tools represent one step further along this path, lowering entry barriers as well as shifting the role of programmers from the more tangible, low-level implementation of systems and debugging to higher-level, more abstract system design and problem-solving.

As these technologies become more embedded in the programming process, it remains to be seen how they will transform the future of software engineering. Perhaps AI is not going to completely displace human programmers, but its impact on the discipline is undeniable, paving the way for a new style of writing, learning, and sustaining software.

2.1.2 Agentic AI

Artificial intelligence agents represent a significant evolution in the field of AI, moving from reactive systems to autonomous decision-making entities. Unlike traditional applications of LLMs, which follow a predefined control flow, AI agents can determine their own control flow to solve complex problems. This flexibility allows agents to dynamically adapt to task requirements, selecting the most appropriate actions, using specific tools, and deciding when to terminate a process. AI agents can set their own goals, plan actions to achieve them, and make decisions based on their understanding of the environment and the task. The architecture of an AI agent includes components such as the agent itself (the decision-making core), tools, memory, planning, reflection, and self-criticism. This structure enables agents to perform complex tasks efficiently and learn from their experiences.

2.1.3 AI & Productivity

New technologies related to artificial intelligence, particularly generative AI and AI Agents, have quickly demonstrated their extraordinary transformative potential. These tools can automate repetitive tasks, optimize human-computer interactions, generate content, analyze data, and support decision-making activities. Their use allows companies and workers to reduce time spent on operational tasks, reallocating resources to higher-value activities that require strategy and creativity. While such predictions were once considered futuristic and lacking empirical evidence outside highly specialized sectors, the recent spread of increasingly accessible and versatile generative models has made it possible to verify their impact across various industries.

Classical economic theories have long highlighted the link between technological innovation and economic growth. According to the growth models developed by Solow (1957) [8], Romer (1990) [9], and Aghion & Howitt (1992) [10], technological progress is the main driver of long-term productivity increases. However, the actual impact of emerging technologies, including artificial intelligence, depends on several factors, including their ability to be effectively integrated into business processes and organizations' adaptability to new technological paradigms. The "Productivity Paradox," also known as Solow's paradox, named after the Nobel prize winner who formulated it, was described by Erik Brynjolfsson in 1993 in an article titled "The Productivity Paradox of Information Technology" [11]. He highlighted a significant discrepancy between the rapid spread of computing in the 1970s and 1980s and the lack of immediate productivity growth in the United States. This phenomenon suggests that the return on investments in new technologies requires an adjustment period to be fully realized. The adoption of new technologies into business practices entails the restructuring of operational systems, the updating of work methodologies, and, most importantly, investing in training the labor force to efficiently use available innovations. The same process may describe the general adoption of AI, which requires not only skill upgrading but also a labor force transformation time to realize its complete economic impact.

At the same time, there is an inherent difficulty in predicting the long-term evolution of companies operating in the AI sector, similar to what happened in the past with the tech industry. The speed at which new innovations emerge makes it challenging to determine which companies will maintain a dominant position.

Generative AI, in particular, is a rapidly changing field with an ecosystem of startups and growing companies, many of which may disappear in the coming decades or, conversely, become major players in the global tech landscape. A notable example of this unpredictability is the market capitalization of tech companies in 2000. An analysis of the ten largest companies by market value at the time shows that only Microsoft remained among the industry's key players, while companies that dominate today, such as Apple, Amazon, and Google, were not among the leaders back then. This demonstrates how highly innovative sectors can undergo rapid transformations, reshaping the hierarchy of leading companies. If AI follows a similar path, we may witness an industry restructuring in the coming decades, with new players emerging and some current ones declining.

More recent research indicates, though, that generative AI may be able to deliver economic benefits sooner than past technological innovation waves. Goldman Sachs in 2023 projected that the implementation of generative AI may increase global GDP by 7% and increase productivity by 1.5 percentage points annually over the next decade. Furthermore, the report identifies that while automation might replace some kinds of jobs, it has historically also led to the creation of new professions. The introduction of information technology, for example, facilitated the rise of

careers such as web developers, digital marketing specialists, and software engineers, contributing to sustainable job growth. Moreover, according to David Autor, 60% of today's work did not exist in 1940, proving the relentless contribution made by technological innovation at all times towards creating new job opportunities.

Despite the plausible positive impact of generative AI and AI Agents on productivity, whether these benefits will extend uniformly to small enterprises remains to be seen. A 2024 study by Accenture [12] underscores that small and medium-sized enterprises (SMEs) in the United Kingdom are embracing generative AI at a higher satisfaction rate than large corporations because they have greater organizational agility and lower dependence on legacy systems. Additionally, according to the report, 86% of the executives of SMEs are satisfied with their return on investment (ROI), as compared to 75% of the executives of large corporations.

Building on these premises, AI agents seem to have the potential to revolutionize business management for SMEs by automating entire operational processes thus mitigating the competitive gap with larger enterprises. Based on a Gartner projection, 33% of business software programs by 2028 will be utilizing AI agent features, which will be more than double compared to less than 1% in 2024, with the capacity for 15% of routine operational choices to be executed without human oversight by AI agents [13]. This outlook further reinforces the idea that AI's role in business processes is set for exponential growth.

However, a recent study, "The Integration of AI in Small Enterprises and Its Impact on Productivity and Innovation" [14], has identified economic and organizational barriers that hinder large-scale AI adoption despite its benefits. Specifically, 41% of surveyed SMEs cited software implementation, development, and acquisition costs as significant obstacles. Additionally, a lack of specialized skills is another limiting factor, prompting companies to invest in training and professional development. These challenges are particularly relevant for smaller companies, which operate with limited resources compared to large tech groups.

Even with studies highlight a promising future of AI in business, two major limitations can deter enterprises from adopting generative AI: hallucinations and explainability. These challenges represent, both from a technical and a business perspective, some serious impediments.

Hallucinations refer to instances where AI generates incorrect, false, or nonsensical information. Despite being a central field of ongoing research, hallucinations are still a prevalent issue. This problem is aptly noted by OpenAI's cautionary message, "ChatGPT can make mistakes." For developers, this presents a challenge as they transition from a deterministic programming approach—where outcomes are clearly defined by if-then statements—to dealing with the unpredictability of AI outputs. This unpredictability is likely to act as an effective deterrent to companies and decision-makers from adopting AI technologies that can at times

produce unreliable results.

Another major challenge is represented by explainability. AI systems, unlike traditional deterministic systems, are black boxes by nature, and the decision-making process can be impenetrable to comprehend. This lack of transparency can be disturbing for businesses that rely on clear, interpretable outcomes when AI systems are required to make decisions. The unpredictability of AI behavior adds a new risk factor that companies find hard to accept, particularly when AI's decision-making processes are not easily explainable.

Moreover, the legal implications of AI adoption are significant. In many fields, the regulatory framework for AI has yet to be fully determined, particularly concerning accountability for independent and autonomous decisions made by AI systems. Issues of responsibility and liability arise when AI makes errors, making it difficult to implement these solutions in scenarios where decision-making transparency is critical.

Overcoming these hallucination issues and explainability is vital for generative AI to gain general adoption and acceptance by enterprise businesses. Overcoming these challenges will pave the way for broader use of AI, which in turn will lead to more robust and effective business processes.

In order to complete the overview on these phenomena, another aspect to be taken into consideration is the role of Robotic Process Automation (RPA). RPA is software that is utilized for automating processes by simulating human interaction with computer systems. It is used to push productivity to its extremes by automating business processes. The technology came into existence in the late 90s and reached its peak of popularity around the 2010s.

Initially, RPA needed explicit coding in order to execute tasks. AI came and broke the paradigm. This shift dismantles the old RPA limitations by enabling the system to become more flexible and efficient in executing complex tasks. Therefore, traditional RPA tools are now starting to be supplemented with AI to enable them to perform more tasks. This evolution is heading towards an integration stage that brings together RPA and generative AI on two main trajectories: Agentic RPA and Adaptive RPA.

Agentic RPA refers to the automatic creation of RPA processes from user inputs without needing sophisticated programming expertise for dynamic and adaptive automation. Adaptive RPA, on the other hand, enables systems to have the ability to adapt their behavior as their context or environment changes,to preserve effective automation even when systems evolve.

Generative AI models, like Claude by Anthropic and Operator by OpenAI, now offer more than just textual generation capabilities. They can run operations on computers, browse the internet, automate processes, and interact with several enterprise systems. According to an article on Appian's website, "If RPA imitates

what a person does, AI imitates how a person thinks." This statement suggests that modern artificial intelligence applications are rapidly reaching automation levels comparable to or even exceeding those of traditional RPA solutions and that only a proactive integration of AI tools into traditional RPA systems can prevent them from being seriously challenged.

One of the fields in which generative AI has already made a solid mark is programming and software development. According to a study conducted by Microsoft Research, tools like GitHub Copilot are significantly improving developer productivity [15]. Researchers performed an experiment on 95 professional programmers that were divided into two groups. The first group, with no possibility to use Copilot, was allowed to utilize regular reference material like Stack Overflow and official documentation. The second group was fully assisted by Copilot throughout the development process. Their task was to write an HTTP server in JavaScript within the shortest time possible. The outcome was that the AI-assisted group completed the task 55.8% faster than the control group, highlighting how AI can accelerate coding.

One of the most interesting aspects regards whether these tools will impact low-code and no-code development platforms, whose primary goal has always been to simplify application development.

The efficiency and autonomy that AI-powered assistants provide bring doubt on how these platforms will be in a world where AI further reduces the need for traditional programming skills. This rapidly evolving scenario suggests that these platforms must develop to remain competitive by embracing advanced agentic AI capabilities or risk losing their fundamental value propositions—such as speed, accessibility, and not requiring extensive technical expertise to develop applications—to AI-driven solutions that not only generate robust and efficient code but also make application development significantly faster and easier for users with minimal prior knowledge.

These findings confirm the strategic role of AI Agents in supporting or even replacing some programming tasks, allowing developers to focus on higher-value activities such as architectural design and code optimization. The increasing integration of these tools will redefine the programmer's role, emphasizing supervision and collaboration with AI models to automate software processes.

The positive impact of AI is evident, and both large and small enterprises face distinct challenges in securing their place in this rapidly evolving market. A critical factor shaping this landscape is the ongoing reduction in the costs associated with AI model development and training. As expenses decrease, broader accessibility may drive greater competition, particularly benefiting small and medium-sized enterprises by leveling the playing field. This trend is well-documented in a Bain & Company report, which notes a rapid decline in AI inference costs driven by continuous innovation [16]. The rise of models like DeepSeek exemplifies this

trajectory, demonstrating how AI is becoming increasingly affordable and attainable. Moreover, a more extraordinary case is the one regarding the S1 model, developed by researchers at Stanford and the University of Washington, which was trained for less than $50 in cloud computing credits while achieving performance comparable to state-of-the-art reasoning models like OpenAI's O1 and DeepSeek's R1 [17]. The availability of such cost-effective AI solutions, coupled with open-source access to training data and code, signals a progressive democratization of AI technology. This trend could empower SMEs to engage more actively in AI-driven processes, fostering a more dynamic, competitive, and inclusive digital economy.

2.2 Natural Language Processing

Natural Language Processing (NLP) is a branch of artificial intelligence that deals with the automatic processing of natural language, enabling computers to understand, learn, and generate text in a meaningful way. This discipline combines methods from computational linguistics, machine learning, and deep learning to analyze natural language and apply it to a wide range of contexts, such as machine translation, speech recognition, information extraction, text classification, and human-computer interaction through chatbots and virtual assistants (Jurafsky & Martin, 2021). Among these tasks, machine translation has been one of the most innovative and transformative fields, acting as a catalyst for the development of new NLP architectures and methodologies. Advances in translation systems have led to the emergence of statistical models and, later, the introduction of Transformer models, which revolutionized the field and gave rise to modern Large Language Models. The significance of machine translation lies not only in its practical utility but also in its key role in developing the most advanced natural language processing systems.

2.2.1 Origin of NLP

The roots of NLP trace back to the early developments in computing and artificial intelligence. As early as the 1950s, Alan Turing proposed the famous Turing Test to evaluate a machine's ability to simulate human intelligence [5], implicitly highlighting the importance of natural language processing. At the same time, Warren Weaver in 1949 suggested applying statistical methods to machine translation, laying the foundation for the field of Machine Translation [18]. In his memorandum, Weaver proposed that translation could be treated as a cryptographic problem, noting that during World War II, cryptographic methods had made it possible to decipher messages in unknown languages by analyzing the frequency of letters and word combinations. This analogy led him to hypothesize the existence of invariant properties in human languages, which could be leveraged to create statistical

models for machine translation. Weaver suggested that language has a hidden logical structure and hypothesized that all languages share universal properties independent of their lexical surface. This gave rise to modern research in universal grammar and distributional representations of meaning. He also suggested the idea that the analysis of surrounding context could improve lexical disambiguation, a principle that later became fundamental in most NLP approaches. A particular instance of his influence is the use of probabilistic models to derive the most likely translation of a word or phrase based on the context in which it is used.

During the 1980s and 1990s, as increased computing power and large datasets became available, NLP shifted from rule-based to statistical and probabilistic methods, e.g., Statistical Machine Translation [19]. This shift was a significant turning point, where translation was driven not by hard linguistic regulations but by probabilistic models learning the mapping between languages from big text datasets. Statistical approaches, comprising hidden Markov model-based as well as Markov chain-based methods, enabled more fluent and natural translations but with modest prospects for managing challenging syntax and context-dependent meaning within sentences.

2.2.2 Statistical Machine Translation

SMT was developed through two primary methodologies: word-based SMT and phrase-based SMT. In the early 1990s, the word-based method analyzed individual words and estimated the probability of a word in the source language being translated into a word in the target language. However, this approach struggled with context management and complex syntactic structures. To overcome these limitations, researchers later introduced phrase-based SMT, which improved translation quality by analyzing sequences of words instead of isolated terms. Phrase-based models enabled more coherent and structured translations by preserving sentence structures in the target language. Google Translate, launched in 2006, was one of the most well-known applications of SMT before the advent of neural machine translation [20] [21].

2.2.3 Word embeddings

NLP, throughout its evolution, has faced two major challenges: syntactic analysis and semantic analysis.

- **Syntactic analysis** focuses on the grammatical structure of sentences, applying linguistic rules to determine relationships between words.
- **Semantic analysis** deals with understanding the meaning of words in context, overcoming ambiguity, and multiple interpretations.

The need for a good representation of words such as for NLP models to be able to understand them in the context led to the word embeddings, which are also considered to be the most influential innovation in the field.

Before the introduction of word embeddings, words had been represented with techniques such as one-hot encoding where each word was mapped to a vector in binary space of the same size as the vocabulary. But this had drastic restrictions, like not being able to encode semantic similarity between words and having high dimensionality, thus computation being inefficient (curse of dimensionality). Distributed word representations were therefore created to surpass these restrictions, where meaning for a word was learned through the context in which the word is used.

A breakthrough in NLP occurred with the introduction of word embeddings, which revolutionized word representation in language models despite some limitations. In 2013, Tomáš Mikolov and his research team at Google developed Word2Vec, a model that learns dense vector representations of words in a multidimensional space [22]. This approach is based on the idea that words with similar meanings tend to appear in similar contexts (distributional semantics), following the principle formulated by linguist J.R. Firth: "You shall know a word by the company it keeps."

Word2Vec introduced two main architectures for learning word vector representations [22]:

- **Continuous Bag of Words** (CBOW): The model predicts a target word based on surrounding words.
- **Skip-Gram**: The model predicts the context given a specific word.

Both methods allowed the creation of word embeddings capable of capturing semantic and syntactic relationships between words. This feature was quickly adopted in multiple NLP applications, from text classification to neural machine translation.

One of the most groundbreaking aspects of Word2Vec was its ability to capture semantic relationships through algebraic operations on word vectors. For example, the operation: "king - man + woman = queen" demonstrated that the vector space learned by these models could preserve semantic and analogical relationships between terms.

However, Word2Vec had some limitations, including the inability to handle out-of-vocabulary (OOV) words and the lack of dynamic contextualization, as learned vectors were static. To address the OOV problem, in 2017, researchers introduced fastText [23], a model that represents words as sets of sub-word units. This approach allowed more robust representations for unknown words by leveraging the morphological composition of words.

Another major advancement was GloVe [24], a model that learns word embeddings by optimizing word co-occurrence probabilities in large text corpora. It was particularly effective in capturing global semantic relationships between words.

2.3 Large Language Models

Large Language Models are a form of artificial intelligence model that is particularly designed to process and generate human-like text. The models are "large" as they are trained on vast datasets of diverse sources of text, which allows them to discover intricate patterns in language as well as nuances of meaning. Their architecture also contains billions of parameters, which makes them great at contextual understanding, coherent response generation, as well as carrying out a wide range of language tasks.

2.3.1 Origin of LLM

To follow the chronological development of these technologies, before exploring LLMs, it is useful to first clarify what Language Models are. As mentioned in the previous paragraphs, these models are specialized in understanding and generating text, drawing on key NLP techniques such as classification, sentiment analysis, or summarization. However, the task of translating entire sentences from one language to another has proven far more complex, as it requires grasping both the structure and the semantic context of words. This complexity stems from factors such as different word orders, discrepancies in the number of words, and the need to interpret not just the individual words but also the role they play within the sentence.

To address these challenges, from the 80s the research community initially experimented with more sophisticated architecture than traditional neural networks, adopting models like RNNs (Recurrent Neural Networks) and LSTMs (Long Short-Term Memory) [25]. These introduced important innovations, such as the ability to retain information over longer sequences compared to conventional networks, yet they still could not fully resolve all the issues related to the complexity of natural language, including long-term dependencies and handling very long sentences.

RNNs, in particular, were designed to process sequences of arbitrary length by reading one element at a time, whether that element is a word in a sentence, a character in a string, or even a daily data point, and maintaining an internal hidden state that updates at each step. This hidden state acts as a summary of all previously observed elements in the sequence, enabling the model to capture context over time. While feedforward or convolutional networks would process each input independently, RNNs apply the same set of weights repeatedly across each

time step (weight sharing), allowing them to detect patterns regardless of where in the sequence they occur.

The architecture of a simple RNN layer can be understood as having two inputs: the current element of the sequence and the hidden state from the previous time step. It then produces two outputs: the prediction at the current time step and the updated hidden state to be passed forward. During training, the process called backpropagation through time (BPTT) unfolds the RNN across the entire sequence, accumulating gradients from each time step. Despite these capabilities, RNNs face notable issues such as vanishing or exploding gradients, limitations in capturing long-term dependencies, and reduced computational efficiency due to their sequential nature. Although LSTMs introduced gating mechanisms to better manage the flow of information and mitigate some of these issues, they still struggled with very long sequences and the inherent complexity of human language.

2.3.2 Transformers

A pivotal step in tackling the challenges faced by earlier models, such as RNNs and LSTMs, was the development of the encoder-decoder architecture. In this framework, the encoder reads an entire input sequence and produces a state vector that encodes the entire context, while the decoder takes that state vector to initiate the generation of the output sequence. The decoder can continue producing tokens until a designated stopping criterion is reached.

Building on this concept of encoder-decoder architecture, Vaswani et al. introduced the Transformer in the landmark paper "Attention Is All You Need" [26]. The Transformer is a novel sequence-to-sequence (seq2seq) architecture that was initially presented in an encoder-decoder form but later saw adaptations in encoder-only (e.g. BERT) and decoder-only (e.g. GPT) variants. Unlike RNN-based models, Transformers overcome long-term dependency issues and do not require unrolling the network across every time step, thus mitigating problems like vanishing or exploding gradients [27]. Furthermore, because the Transformer can process tokens in parallel rather than sequentially, it offers a significant speedup during training and inference for many tasks. One key element of the Transformer is the tokenization process, which splits the input into subword units or tokens. Another crucial innovation is positional encoding, which introduces information about the position of each token within the sequence and addresses the absence of a recurrent mechanism in the model. Although "attention" appears prominently in the title of the paper, it was not an entirely new concept. Earlier works, such as Bahdanau et al.'s "Neural Machine Translation by Jointly Learning to Align and Translate" (2014), had already introduced attention mechanisms. The main breakthrough, however, lay in the integration of attention within a fully feed-forward encoder-decoder structure, combined with positional encoding [28].

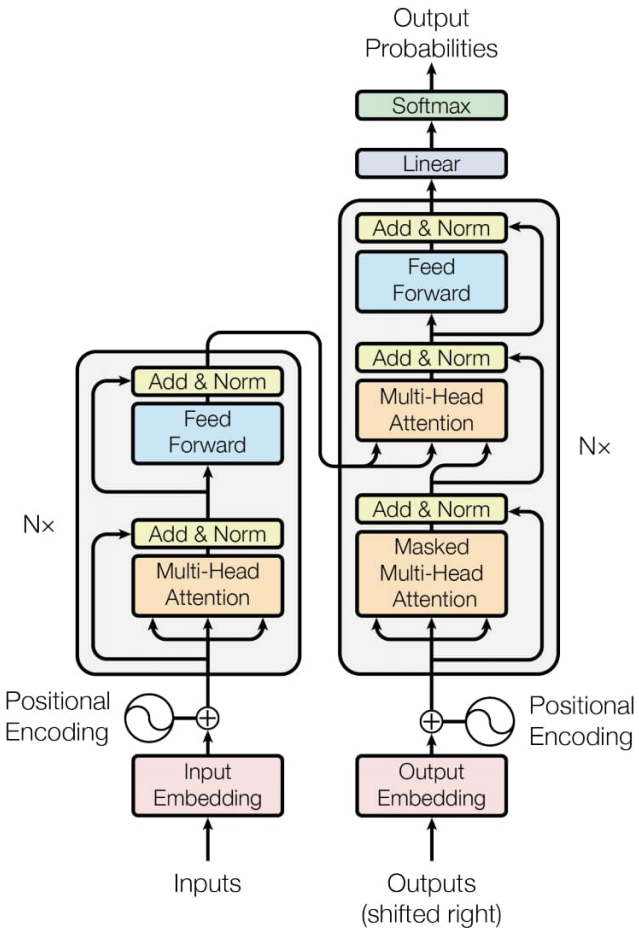

Figure 2.1: Attention is All You Need - Transformer Architecture

Most famously, the Transformer leverages attention mechanisms that allow the model to selectively focus on different parts of the input sequence, assigning varying weights to each token depending on its relevance. In broad terms, the original Transformer architecture organizes these operations into two modules: an encoder that processes the entire input sequence, and a decoder that uses previously generated tokens to predict subsequent outputs. Both modules are composed of multiple layers, each typically including a multi-head attention mechanism and a feed-forward sub-layer. While the encoder processes the entire input sequence to create contextual representations, the decoder uses both these representations and any previously generated tokens to predict the next output token.

During inference, the encoder reads the entire input sequence and produces a corresponding hidden representation for each token in the sequence. The decoder

then begins its generation process by receiving a special beginning-of-sequence (BOS) token as input, predicting the next token through a classification over the entire vocabulary. The newly generated token is fed back into the decoder to predict the subsequent token, and this process repeats until an end-of-sequence (EOS) token is produced. At training time, the model's loss is computed over all predicted tokens, but at inference time only the latest generated token is typically used.

Because the decoder does not maintain its own state across steps, it must be fed the previously generated tokens at every stage. Although stateful models like RNNs can retain a history of past inputs, they risk forgetting critical information over long sequences. In contrast, the statelessness of the Transformer decoder, combined with attention to all tokens generated previously, gives a more robust handling of intricate dependencies. Despite being more computationally expensive for very long inputs, the parallelization-friendly design of the Transformer has been revolutionary in NLP research.

Tokenization

Tokenization is a fundamental process of text data preprocessing for Transformer-based models and other neural models since these models have to process sentences by segmenting them into small units referred to as tokens. The tokenization process can have a significant impact on model performance, particularly in the handling of out-of-vocabulary words, semantic information retention, and vocabulary size management. The following are the main methods commonly used, their strengths, and weaknesses.

- **Character-Level Tokenization**
  In character-level tokenization, the text is split into individual characters, and each character is treated as a token. This approach ensures no out-of-vocabulary issues, as every character from the dataset is recognized, and it is robust to misspellings or variations in spelling. However, it produces long sequences that can slow down training and make it more complex. Since each character is processed independently, capturing higher-level semantic meanings becomes more challenging.

- **Word-Level Tokenization**
  Another approach is word-level tokenization, where text is split into individual words. Each word is treated as a token, yielding fewer tokens than a purely character-based system. This method captures semantic information at the word level and typically results in shorter sequences. However, word-level tokenization often struggles with out-of-vocabulary words, such as rare or newly introduced terms, and does not consider morphological relationships

like prefixes and suffixes. It also necessitates a large vocabulary, which can lead to memory inefficiency.

- **Subword-Level Tokenization**
  Subword-level tokenization seeks to strike a balance between character-level and word-level approaches. It splits text into smaller units that can represent common roots, prefixes, or suffixes. This helps address out-of-vocabulary problems more effectively, since rare words can be decomposed into known subwords rather than treated as entirely unknown tokens. Subword tokenization is widely used in modern NLP models, including Transformer-based architectures. A particularly popular method is Byte-Pair Encoding (BPE), first described by Philip Gage in 1994. Subword-level tokenization helps keep the vocabulary size manageable, while still providing a good representation for both common and rare words.

This last approach is, the most widely used tokenization technique for Large Language Models (LLMs). Words are decomposed into smaller pieces, each associated with a token number. For example, the word "unhappiness" may be tokenized into ["un", "happiness"], or even further into ["un", "hap", "pi", "ness"].

The vocabulary size of an LLM is directly influenced by the tokenization strategy. Recent models illustrate this variability: GPT-3 has a vocabulary size of approximately 50,000 tokens, while PaLM 2 uses around 256,000 tokens, and Claude 2 operates with a vocabulary of roughly 100,000 tokens [29] [30].

Attention

Attention is a mechanism that allows neural networks to selectively focus on specific parts of an input sequence, refining and contextualizing each token's representation. It computes weights that indicate how important each token is with respect to every other token in the sequence. The goal of a Transformer is to progressively refine word embeddings, incorporating contextual meaning rather than treating words in isolation. Attention helps disambiguate words with multiple meanings by adjusting embeddings based on surrounding words. For example, the word "light" can refer to illumination or something that is not heavy, and attention ensures the correct interpretation based on context.

Attention Mechanism

Each token in the sequence is associated with three vectors, which together enable a continuous representation of dictionary lookups:

- **Query** (Q): Represents what information the token is searching for (e.g., a noun seeking related adjectives).

- **Key** (K): Determines whether a token is relevant in response to a query (e.g., an adjective).

- **Value** (V): Represents the information that will be added to the querying token if relevance is confirmed.

Unlike traditional dictionary lookups, which are discrete (a key either matches or does not), attention generalizes this process to a continuous space. Instead of a strict match, it computes similarity scores between queries and keys using a dot product, producing a continuous measure of relevance. These scores are then passed through a softmax function, normalizing them into attention weights that sum to one.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

This allows the model to dynamically distribute focus across multiple tokens rather than relying on a single exact match. The scaling factor 1/rad(dk) ensures a more balanced distribution of attention weights
The final contextual embedding of a token is computed as a weighted sum of value vectors, incorporating information from all related words. This mechanism enables more flexible and robust context modeling, capturing long-range dependencies and nuanced relationships between words.

Transformers utilize three distinct types of attention mechanisms, each designed to process and contextualize information in different ways:

- **Self-Attention:** The mechanism that allows each token in a sequence to attend to every other token in the same sequence. This is fundamental to how Transformers encode contextual relationships without relying on recurrent structures. It is particularly effective in capturing dependencies across long distances within the same text.

- **Cross-Attention:** Unlike self-attention, cross-attention occurs when queries from one sequence attend to keys and values from a different sequence. This is commonly used in tasks like machine translation, where an encoder processes the source language, and the decoder attends to it while generating the target language.

- **Decoder Masked Attention:** A form of self-attention that is significant for autoregressive generation, where future tokens should not be allowed to condition on earlier tokens during training. To enforce this constraint, attention weights corresponding to future positions are set to negative infinity before the softmax function, so they have zero weight in the ultimate computation.

Multi Head Attention
A complete attention block consists of multiple independent attention heads, each with its own query, key, and value matrices. This enables the model to capture different types of contextual relationships in parallel. Each head specializes in different aspects of meaning, allowing the Transformer to build a richer representation of the input sequence.

Positional Encoding

Transformers process tokens in parallel using self-attention, a mechanism that dynamically weighs relationships between words. However, self-attention alone is permutation-invariant, meaning it does not inherently capture the order of words in a sentence. This limitation makes it impossible for a Transformer to distinguish between sequences such as "The cat chased the dog" and "The dog chased the cat" without additional positional information.

To address this, positional encoding is introduced, ensuring that each token is aware of its place in the sequence. Unlike traditional embeddings that treat words as static units, positional encodings augment token embeddings by injecting order-specific information. The sinusoidal encoding method, proposed in Attention Is All You Need, leverages sine and cosine functions to generate unique position-dependent vectors. These periodic functions allow the model to preserve relative distances between tokens and generalize to unseen sequence lengths [26].

Others, however, such as GPT-2, opt for learned positional embeddings, where position-conditioned vectors are also learned jointly with the model. While this solution works for specific datasets, it does not accompany the inductive bias of sinusoidal encodings, and so it is less effective for extrapolating beyond the training domain [31].

Through positional encodings, Transformers solve the lack of sequential awareness in self-attention, enabling them to distinguish word order and effectively model syntactic and semantic relationships.

2.3.3 Evolution of Transformers

Following the introduction of the Transformer, a wide range of models have been developed that refine and extend its capabilities. This section provides an overview of key architectures that have emerged from this foundation.

T5 (Text-to-Text Transfer Transformer) is a vanilla encoder-decoder transformer designed for NLP general-purpose tasks like translation, question answering, and classification. It varies from task-specialized models by having one universal framework with a single architecture, applying the same loss function across

different tasks. It encodes the task type explicitly in the input, allowing greater flexibility and reuse [32].

Encoder-only Models

Encoder-only models are constructed for learning representations, with the goal of encoding input sequences alone without a dedicated decoding process. They thereby compromise the generative aspect of encoder-decoder models and are optimized towards applications that require comprehension of fine-grained aspects rather than text generation. These models are applied widely in text classification (such as spam filtering and sentiment analysis), named entity recognition (NER) (for the recognition of names, locations, and dates), and sentence pair classification (for the recognition of relationships like paraphrasing and entailment). Without a decoder, these models do not generate text but instead extract, classify, and analyze input data by leveraging self-attention and bidirectional context processing. Their ability to encode rich contextual information makes them essential for applications requiring strong comprehension rather than content creation. One of the most popular encoder-only models is BERT. This model has been designed specifically to enhance natural language understanding through bidirectional contextual learning. It has been pretrained for two different tasks. Masked Language Modeling and Next Sentence Prediction [33].

Decoder-only Models

The idea behind decoder-only transformer models emerged from a fundamental shift in how researchers approached text generation. Traditional encoder-decoder architectures, while powerful, introduced inefficiencies when applied to tasks that required pure generation.

Researchers hypothesized that the encoder, which processed the whole input before any output was generated, was not needed for most generative tasks. Instead, with autoregressive modeling and self-attention, it was now feasible to generate contextually relevant and coherent text by conditioning only on tokens previously generated.

The Generative Pre-Trained Transformer (GPT) family series, introduced by OpenAI in 2018, is a perfect example of such design, illustrating how decoder-only models can scale well, becoming more intuitive to human-like text [34].

Decoder-only models, as mentioned, eliminate the encoder and solely employ self-attention mechanisms to generate sequences autoregressively. Decoder-only models are pre-trained on vast texts using self-supervised learning where they predict the next word based solely on previous tokens without employing labeled annotations. This enables them to build linguistic representations that can be transferred across a variety of varied tasks.

To generate coherent and diverse text, decoder-only models employ various sampling strategies for token selection. The output probability distribution over the vocabulary at each step determines the likelihood of different words appearing next. Below is an overview of common sampling methods:

Sampling Method	**Description**
Greedy Search	Selects the most probable token at each step, leading to deterministic but sometimes repetitive outputs.
Beam Search	Explores multiple candidate sequences simultaneously, optimizing global coherence but potentially leading to generic responses.
Top-k Sampling	Restricts selection to the k most probable tokens, preventing rare but nonsensical words from appearing.
Top-p (Nucleus) Sampling	Dynamically adjusts the probability mass, selecting tokens from the smallest set whose probabilities sum to p (e.g., 0.9).
Temperature Scaling	Modifies the probability distribution, where lower values (e.g., 0.1) make responses more deterministic and higher values (e.g., 1.5) increase diversity.

Table 2.1: Comparison of Sampling Methods in Text Generation

As said, this revolutionary approach has been deployed from OpenAI with the GPT Family. In the last years, these models have undergone significant advancements in size, training data, and capabilities.

GPT-1 (2018): The first model in the GPT series introduced the decoder-only architecture for text generation. It was trained using unsupervised pretraining on the BooksCorpus dataset[35], comprising approximately 5GB of text from 7,000 books. The model had 12 layers, 12 attention heads, a hidden state of 768 dimensions, and a context size of 512 tokens, totaling 117 million parameters. Fine-tuning was later applied for tasks such as sentiment analysis and natural language inference [34].

GPT-2 (2019): GPT-2 expanded upon GPT-1 by increasing the number of parameters 10 times, to 1.5 billion and training on WebText, a dataset of curated Reddit links (40GB). It revealed emerging capabilities, where the model solved tasks it was not explicitly fine-tuned on. The sequence length was doubled to 1024 tokens, and the model adopted Byte Pair Encoding (BPE) with 50,257 tokens [31].

GPT-3 (2020): GPT-3 marked a turning point in generative AI, scaling up to 175 billion parameters and introducing a novel paradigm of in-context learning. Instead of requiring gradient-based fine-tuning for new tasks, GPT-3 could adapt

dynamically by processing task descriptions and examples as part of its input prompt. This showed for the first time the potentiality of few-shot learning, a technique where only a few demonstrations are provided but the model is able to generalize tasks never seen before without explicit weight updates. This showed an emergent ability for zero-shot, one-shot, and few-shot work, where even in the absence of prior exposure to some queries, the model could generate competent answers [30].

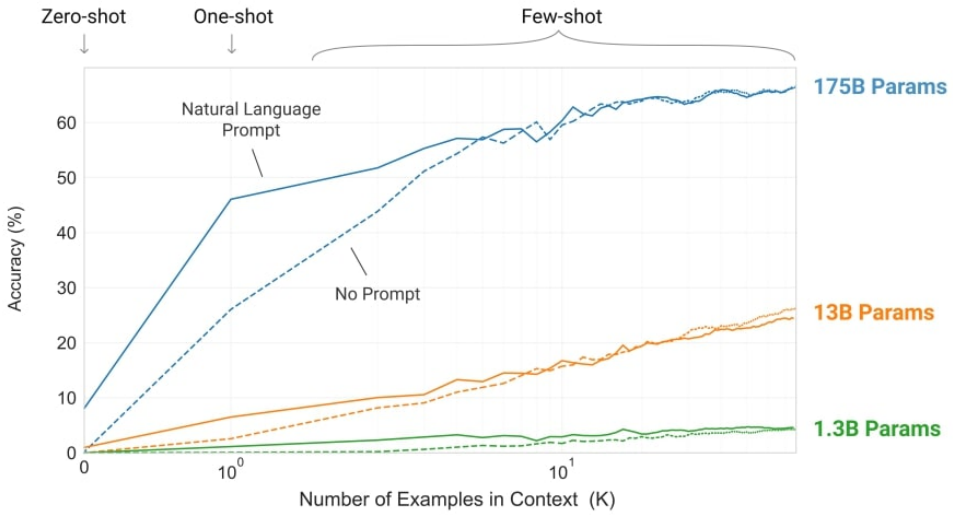

Figure 2.2: Few-Shot Learning Performance Across Model Sizes

GPT-3 was in this sense revolutionary because it proved that large enough language models were not only able to learn from data, but were also able to read and adapt to new instructions in a flexible, near-human manner [30].

GPT Models Nowadays: Since the launch of GPT-3 in 2020, a lot of things have changed in the landscape. In particular, there has been a less open policy from OpenAI concerning the technical aspects of its model, which has made it challenging for the scientific community to have the deepest insights into the architectural advances and the dataset utilized in the following models. GPT-4, launched in 2023, increased multimodal capacity to the extent of being able to process text and images [36]. In 2024, OpenAI introduced GPT-4o [37], enhancing multimodal AI with real-time text, audio, and image generation. This advancement significantly reduced latency and computational costs. Following this, they developed the 'o' model series, beginning with o1, which focused on advanced reasoning and problem-solving.

In particular, these models work by introducing a chain-of-thought process

with the introduction of reasoning tokens. These new kinds of tokens help the model "think" and "reason" by breaking down the input sequence and improving the prompt understanding. These reasoning tokens are discarded when the model generates the final response. These models demonstrated improved contextual understanding and adaptability, moving closer to human-like cognitive processing. The evolution continued with o3, which further refined reasoning abilities and computational performance.

Model's Overview: As the nature of artificial intelligence continues to change, numerous organizations have come up with models competing with those of OpenAI, both open-source and proprietary.

Anthropic's Claude Series: Anthropic introduced the Claude series of models emphasizing performance and security. Claude 3.5 Sonnet, released in June 2024, shows phenomenal improvements in coding, multistep pipelines, and visual reasoning tasks. The latest model introduced by Anthropic is Claude 3.7 which further improved the advanced reasoning and problem-solving abilities [38] [39].

Meta's Llama Series: Meta has advanced its open-source initiatives with the Llama series. Llama 3, launched in April 2024, includes models like the 405B, 70B, and 8B, catering to diverse computational needs. The 405B model, with a 128,000-token context window, excels in extensive data processing and synthetic data generation. These models have been recognized for their proficiency in coding and text-generation tasks [40].

Google's Gemini Series: Google has expanded its AI portfolio with the Gemini series. The recent Gemini 2.0 Flash Thinking update introduces advanced reasoning capabilities, enhancing the model's ability to explain answers to complex questions. Additionally, the Gemini 2.0 Pro model focuses on improving factual accuracy and performance in coding and mathematics tasks [41] [42].

2.4 Datasets

Large Language Models are pre-trained on huge datasets that contain various textual sources. Training occurs through self-supervised learning methods, which allow models to learn about linguistic patterns and semantic dependencies. Self-supervised learning strategies consist of masked language modeling (MLM), as employed by BERT, in which words of the text are masked for prediction, and causal language modeling (CLM), as employed by GPT models, in which words are predicted one at a time.

Generic datasets serve as foundational resources, allowing models to learn language patterns, grammar, and contextual nuances required for understanding and generation of human-like text. Datasets are chosen based on linguistic richness, diversity, and availability. The choice of datasets also carries ethical considerations,

such as the danger of introducing biases and the quality of the text.

2.4.1 Major Training Datasets

These datasets originate from various sources, each serving a unique role in large language model training:

- **Common Crawl**: A publicly available repository of web-crawled data containing approximately 240TB of raw data and 10PB+ of processed data. It provides a vast and diverse corpus but requires extensive filtering to ensure data quality [43].
- **BooksCorpus**: A dataset of published books used for long-form text training, comprising 11,038 books and approximately 985 million words. It helps LLMs learn coherent, long-form reasoning and storytelling [35].
- **WebText:** A dataset created by OpenAI containing web text scraped from high-quality outbound links shared on Reddit. It includes around 40GB of diverse text data from a wide range of topics. While offering rich language variety, it requires attention to potential biases due to the selective nature of the content sources [44].
- **OpenWebText**: A dataset designed to replicate the quality of OpenAI's WebText dataset, containing around 38GB of cleaned web text data. It ensures diverse language usage but requires careful ethical considerations due to potential biases.
- **C4 (Colossal Clean Crawled Corpus)**: A dataset extracted from Common Crawl, cleaned to enhance text quality, consisting of 750GB of high-quality English text data. It is widely used for large-scale pretraining and improves generalization across tasks.
- **Wikitext-103**: A dataset containing over 100 million tokens from Wikipedia's top articles, used for training long-form content models. It retains a more natural text flow compared to standard Wikipedia dumps.
- **PG19**: A dataset curated from Project Gutenberg, containing a diverse set of books aimed at facilitating research in long-form language modeling, with an emphasis on unsupervised learning.
- **LAMBADA**: A dataset for evaluating contextual text understanding through a word prediction task, where models must predict the last word of a passage based on its full context. This helps assess a model's ability to process and retain long-range dependencies.

Different LLMs utilize such datasets differently. BERT, for instance, benefits from Wikipedia and BooksCorpus to gain deep language structure sense through MLM. GPT models like GPT-3 employ a mix of Common Crawl, OpenWebText, and Wikipedia to gain the diversity of languages present in the various datasets through CLM. Concurrently, T5 (Text-to-Text Transfer Transformer) is also trained on the C4 dataset, which causes it to excel extremely well on a broad set of NLP tasks by framing them as text transformation problems [35].

However, a growing concern in recent years is the increasing opacity regarding the datasets used to train newer LLMs. Unlike earlier models where dataset sources were explicitly stated, many modern models, such as GPT-4, have been trained on datasets whose exact composition remains undisclosed. This lack of transparency makes it difficult for researchers to conduct a thorough analysis of how these models learn, what biases they may carry, and how their training data impacts performance. Some studies suggest that user-generated data, accumulated over years of interactions with AI-powered systems, could play a significant role in training modern models, though concrete evidence remains sparse due to the proprietary nature of these datasets. This situation raises ethical and methodological concerns, as it limits the ability of independent researchers to audit, replicate, and fully understand the learning processes of state-of-the-art models.

2.4.2 Size and Quality

The performance of an LLM in general is accounted for by the quality and size of the training datasets. A model trained on a large and diverse dataset learns a better understanding of language patterns and subject matters and becomes more capable of generalizing across settings. Empirical studies illustrated that larger datasets yield better performance, following the so-called scaling laws of machine learning. However, size is not all; data quality is also essential. Poorly curated datasets can spread biases and disinformation, and extensive filtering and curation of datasets are required to negate such issues. Biases in datasets need to be particularly tackled to ensure fairness and avoid the reinforcement of negative stereotypes in AI output [45] [46] [47].

2.4.3 Data Contamination

Data contamination in large language models occurs when testing results or evaluation benchmarks intersect with the training data, return unreliable results. This overlap could lead to artificially enhance performance scores on those benchmarks that the models already saw multiple times during their training process. This issue is aggravated by the fact that training data is black-boxed, particularly in proprietary and commercial models. This lack of transparency makes it difficult to

measure the degree of such contamination [48].

The study published in "Investigating Data Contamination in Modern Benchmarks for Large Language Models" further describes and analyzes this issue. It highlights that inflated benchmarks can be misleading and even may not reflect the model's performance accurately due to potential contamination. Therefore, the authors recommend fixing this problem with the call for transparency in the training data to provide stable and valid tests [49].

Similarly, the "Benchmark Data Contamination of Large Language Models: A Survey" paper describes how unintentional inclusion of evaluation data in training sets can compromise model test validity. The article also explores the challenges associated with data contamination and proposes alternative evaluation methods hoping to reduce these issues [50].

Moreover, the research "An Open-Source Data Contamination Report for Large Language Models" reveals that models like GPT-4 were found to have high contamination levels in some test evaluation metrics. For instance, some university tests used as an evaluation metric were found to be contaminated by 20%. This result raises concerns about the validity of performance metrics. [51]

Additionally, Meta's technical report on Llama 3.1 also includes a comprehensive review of data contamination in LLM benchmarks. The report confirms that a few of the extremely highly used benchmarks that are widely used in practice have more than 40% contamination that significantly affects the estimated model performance. Meta report also acknowledges that given this incidental exposure to evaluation data during the training process, many popular benchmarks may be limited in being representative of a model's generalization capabilities. These outcomes emphasize the urgent need for the development of uncontaminated test sets and more rigorous and transparent data filtering methods to ensure the validity of performance assessments [52].

These findings collectively highlight the critical need for rigorous decontamination processes and the development of new, clean benchmarks. Without such measures, evaluations may give a misleading view of a model's true performance, leading to an overestimation of its capabilities.

2.5 AI Agents

The first, well-known definition of Agents was given by Wooldridge & Jennings in the paper "Intelligent Agents: Theory and Practice" (1995) [53]. Specifically, they depicted an agent as a computational system, either hardware or software,

that possesses autonomy, operating without direct intervention from humans or other systems. This means it does not need to depend on people or other systems and does not have to wait for direct instructions. Agents must also be able to communicate with the outside world, either with users or with other agents. This ability is called sociality. In addition, agents should respond to changes in their environment, a skill known as reactivity. However, agents should not only be reactive but also proactive. This means they must understand their goals and adapt to different situations to achieve them. (Wooldridge & Jennings, 1995).

In this context, AI agents represent an artificial intelligence system that expands the role of LLMs, transforming them into systems that surpass the traditional reactive approach of LLMs. This is achieved through the introduction of a dynamic control flow, which enables them to tackle complex tasks using adaptive strategies (Russell & Norvig, 2021).

An AI Agent can plan, execute actions, store information, and strategically utilize various tools. This makes it an autonomous and versatile entity, capable of operating in dynamic environments and handling sophisticated tasks.

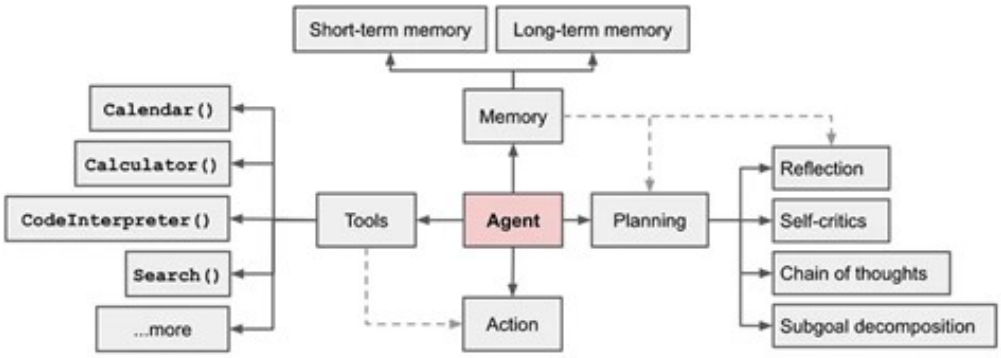

Figure 2.3: Overview of Agent Architecture

Agent Architecture

According to the structure proposed by Weng, Lilian (2023) in "LLM Powered Autonomous Agents" [54], an AI Agent can be considered as composed of six main modules:

- **Agent:** The core of intelligence, responsible for managing and orchestrating various modules.
- **Memory:** Enables the agent to store acquired information over time to improve learning and decision-making capabilities.

- **Planning:** Allows the agent to structure and anticipate sequences of actions to achieve a specific goal.
- **Tools:** External tools and APIs that the agent can use to gather data, perform computations, or interact with other systems.
- **Action:** The concrete execution of planned actions, through physical or digital interfaces.
- **Reflection:** Self-evaluation mechanisms to enhance future performance [55].

This structure allows AI Agents to operate autonomously, progressively improving their performance and adapting to new challenges.

Agent

The agent represents the decision-making core of an agentic system. It is therefore capable of independently determining its own control flow to solve complex problems. This concept is different from the traditional LLM chains, in which the model follows a pre-defined sequence of operations. The agent can make decisions independently. It determines which tools to use as needed and when to complete a process. Its control flow is not constrained to a rigid sequence of operations but adapts dynamically to circumstances. In addition, the agent can optimize operational efficiency by avoiding unnecessary computation. Classic LLM chains represent the sequential structure that anticipates and inspires AI agents' behavior. They allow complex problems to be decomposed into solvable subtasks through multiple interactions with the model [56].

Main types of chains:

- **Sequential chains:** Linear processes in which each output becomes the next input.
- **Tree chains:** Hierarchical structures that process multiple inputs and produce multiple outputs.
- **Router chains:** Systems that dynamically select the optimal processing path based on the nature of the problem.

Memory

As stated, the memory component is crucial for AI agents to store useful information and retrieve the knowledge acquired to autonomously perform the assigned task. In particular, memory in AI agents is divided into:

- **Short-term memory:** Contains temporary information relevant to the current task.
- **Long-term memory:** Stores historical knowledge and past experiences for future use.

AI agents use memory retrieval techniques to extract relevant information and improve the accuracy of responses. Memory reflection allows for updating and adapting behavior patterns based on accumulated experience.

In addition to memory, AI agents can integrate information from external sources through:

- Queries on databases and knowledge bases (SQL, ChatDB).
- Web scraping and API calls to obtain updated data.
- Retrieval-augmented generation (RAG) models to improve content generation [57].

Tools

One of the key elements in the design of an AI agent is the tool module, which allows the agent to dynamically interact with external sources and specialized tools to enhance its processing and response capabilities. This module acts as an interface between the agent and external resources, allowing it to retrieve real-time data, perform complex calculations, and access structured databases.

Planning

The planning module allows the agent to organize sequences of actions necessary to achieve a specific goal. This process includes several advanced strategies that improve the agent's efficiency or flexibility:

- **Reflection:** A metacognitive process that allows the agent to evaluate his or her past decisions and actions to identify room for improvement. Through this ability, the agent learns from both successes and failures, increasing his or her decision-making efficiency.
- **Self-Criticism:** An internal feedback mechanism that enables the agent to critically analyze its performance and propose improvements. This feature helps refine decision-making and produce more reliable and consistent outputs.
- **Chain of Thought (Sequential Reasoning):**Logical step-by-step approach to reasoning that enables the agent to address problems logically formulated. This technique minimizes errors and facilitates debugging and step-by-step improvement.

- **Subgoal Decomposition:** A goal-oriented approach which breaks down difficult problems into smaller tasks. This makes it possible for the agent to advance stepwise, with direct control over every step of the procedure and with optimal intervention if the process turns problematic.

Agents' Interaction

AI agents operate both individually and in collaborative settings within multi-agent environments. Individual agents may interact with each other, exchanging information, collaborating to achieve common goals, or competing against each other.

In the hierarchical system, the AI agents may be organized in layers, and the top-level agents may have control and regulation over the low-level agents. The hierarchical structure enables the resource to be assigned optimally. Hierarchical agents can be divided into control agents, processing agents, and operational agents.

Control agents have a global view of the system and make strategic decisions. Processing agents manage information flows while operational agents perform specific tasks.

In multi-agent systems (MAS), AI agents can be classified according to the level of cooperation. In the collaborative model, agents share information and cooperate towards a common goal. In the competitive model, however, agents are autonomous and can compete to achieve competing objectives. There are also mixed approaches that are characterized by the combination of cooperation and competition agents [58].

2.6 Prompting

Prompt Engineering is the process of crafting and optimizing instructions to guide Large Language Models in generating specific and accurate outputs. Instead of modifying the model's architecture or weights, it focuses on improving the input prompts to achieve desired results. This method requires less computational power compared to retraining or fine-tuning [30] [59].

To achieve optimal results with LLMs, certain key principles must be followed:

1. Prompts should be specific to generate more targeted outputs.
2. Clear and concise language helps reduce ambiguity.
3. Assigning a role to the LLM guides it to respond in a defined manner.
4. Providing examples and contextual information improves response consistency.
5. Defining the output format, such as JSON or structured SQL, ensures clarity.

6. A step-by-step approach helps in handling complex tasks effectively.

7. Continuous testing and refinement allow for prompt optimization based on results.

Besides these specific principles, different prompt techniques should be explored to analyze completely this approach.

2.6.1 Prompt Techniques

Priming involves giving an initial detailed input to set the context and guide the model's response in a specific direction. Tabular Format Prompting structures data into tables, improving clarity and ensuring precise outputs. Fill-in-the-Blank Prompting focuses on specific text elements, making it effective for content completion and targeted modifications. Perspective Prompting helps LLMs explore multiple viewpoints, which enhances analytical depth and argumentative reasoning.

Another method is RGC (Role, Result, Goal, Context, Constraint). This technique gives the model a clear role. It explains what the task is and what the goal should be. It also includes background details and rules to follow. This helps the model give better and more accurate answers.

Shot Prompting, which includes Zero-shot, One-shot, and Few-shot learning, influences the model's performance based on the number of provided examples. Zero-shot relies solely on the model's pre-trained knowledge, while One-shot and Few-shot prompting improve response coherence by incorporating one or more examples, respectively. Research by Brown et al. (2020) in their seminal paper "Language Models are Few-Shot Learners" established the foundation for Few-shot learning, demonstrating how increasing the number of examples significantly improves model performance [30]. This approach has also been further refined through instruction tuning methods, as explored in further research, which discusses reinforcement learning and prompt-based adaptation to align model responses with human expectations.

More advanced techniques, such as Generate Knowledge Prompting, improve responses by integrating external information beyond a model's training data.

Chain of Thought (CoT) Prompting, introduced by Google researchers, structures reasoning into sequential steps [60]. By breaking down challenging problems, it enhances logical coherence and reduces hallucinations. This approach makes this strategy really useful for maths and logical problems. Self-Consistency with CoT (CoT-SC) enhances this by generating several reasoning paths and selecting the most coherent one [61].

Another prompting strategy that extends CoT is the Tree of Thoughts (ToT) [62]. It introduces a hierarchical reasoning structure that allows the exploration of multiple solutions for strategic decision-making. Structured Chain of Thought

(SCoT) applies CoT's principles with a more structured approach, ensuring logical query correctness [63].

These techniques have greatly influenced reflective models, which iteratively validate and refine their outputs. The "Chain of Thought Prompting Elicits Reasoning in Large Language Models" paper showed how structured prompting significantly enhances problem-solving ability in LLMs.

2.6.2 Other Approaches

An interesting field of research related to prompt engineering strategies is the transition from human prompts to AI-generated ones. This phenomenon is described in Dina Genkina's paper called "AI Prompt Engineering is Dead: Long Live AI Prompt Engineering" [64]. The article explains that AI models have the possibility of outperforming humans in crafting effective prompts. This is facilitated by self-supervised learning approaches that allow LLMs to refine their prompts based on performance feedback.

Moreover, researchers continue to come up with new prompting techniques that can enhance model abilities and outcomes. One of the most notable advancements regards the introduction of multimodal prompting [65] , which extends beyond text-based prompts by incorporating images, audio, and structured data, enabling AI to process a wider range of inputs. Moreover, despite increasing token window sizes in modern LLMs, reducing token overhead remains crucial for maintaining clarity and efficiency in prompt design. SudoLang, for instance, in the field of programming tasks, is trying to achieve a reduction in token usage and an increase in accuracy. It is a pseudocode programming language designed to optimize interactions with LLMs that enables developers to write concise yet precise prompts.

Hallucinations occur when LLMs generate responses that are plausible but factually incorrect or nonsensical. These errors arise due to limitations in training data, overgeneralization, or the absence of relevant context [66]. Prompt Engineering can be useful to mitigate this problem using structured prompting techniques, such as Chain-of-Thought (CoT) prompting, already seen in the previous techniques. Another interesting solution is the Retrieval-Augmented Generation (RAG) improves response accuracy by integrating external knowledge. Instead of relying solely on pre-trained data, RAG retrieves relevant information from a database or knowledge source before generating responses. This real-time retrieval grounds the model's output in factual information, significantly reducing hallucinations [60] [57].

Chapter 3

LLMs for Programming Tasks

This section provides a technical background on LLMs for programming in great detail, focusing on how the models are tailored towards coding intent and how one should measure their performance. LLMs are made programming competent via specialized pretraining and fine-tuning on huge amounts of code, documentation, and technical discussions, derived from sources like GitHub, Stack Overflow, or publicly available, open-source repositories. This training enables them to learn the syntax, semantics, and grammar of several languages and also the logic and reasoning skills required for the resolution of complex software issues.

This section also summarily discusses some of the key aspects, such as pretraining on diverse sets of code and fine-tuning on domain-specific data that enable LLMs to execute code understanding, generation, and bug detection tasks efficiently. Moreover, it introduces the role of standard benchmarks, such as HumanEval [67], CodeXGLUE, and APPS [68], which are used to rigorously evaluate these models' performance at natural language-to-code translation, defect detection, and competitive programming. These are the elements that pave the way for more technical discussion in the subsequent sections, with the datasets, benchmarks, and architectures being the very core of coding-specific LLMs being put under closer inspection.

3.1 Types of Tasks Performed by LLMs in Software Development

It is extremely important to give an overview of the general capabilities of LLMs and AI systems on coding and programming before beginning the discussion on

applications of text-to-code. These models have shown multiple abilities, extending beyond generative tasks to enable various requirements in programming.

Generative Tasks – These involve the generation or rewriting of code from input data, for example, natural language specifications or partial code snippets.

- **Code Completion** – Predicts and offers the next logical code elements based on existing context, accelerating the development process.
- **Code Generation** – Produces executable code directly from natural language specifications, bridging the gap between human descriptions and machine-executable programs.
- **Code Translation** – Translates code from different programming languages without loss of functionality, allowing cross-platform development without rewrites.
- **Code Summarization** – Automatically generates a concise summary of functions or classes, beneficial in software documentation, knowledge transfer, and code maintainability.

Classification and Analysis Tasks – Tasks that involve understanding and classification of code, usually to improve security, searchability, or debugging efficiency.

- **Bug Detection and Automatic Debugging** – Detects and corrects syntactical or logical faults, reducing debugging time as well as software reliability.
- **Code Search and Retrieval** – Gives appropriate code segments based on text query, hence beneficial in the big code base by giving an instant reference to reusable functionality.
- **Code Categorization** – Organizes code blocks into categories of function, security threat, or software component classification, which simplifies security auditing and makes software more readable.

3.2 Coding Datasets and Benchmarks

While large-scale, general-purpose datasets form the foundation of most LLMs, a growing trend involves the creation of specialized datasets designed for particular tasks or domains. These datasets enhance the model's adaptability and effectiveness in specific contexts, allowing it to develop a deeper understanding

and improved performance in targeted applications.For instance, models that specialize in programming such as OpenAI Codex and StarCoder are trained on enormous databases of source codes comprising GitHub, Stack Overflow, and other programming-related databases. OpenAI Codex, for instance, was trained on 179 GB of distinct Python code extracted from publicly accessible repositories so that it can generate code, debug, and even create software independently. Similarly, BigCode's StarCoder was built on a 6.4 TB dataset called "The Stack," which had permissively licensed code in 384 programming languages and was thus better able to complete and evaluate software development tasks efficiently. These examples demonstrate how specialized datasets refine a model's capabilities, making it more effective in targeted professional and technical domains.

One of the most prominent examples of task-specific datasets is found in programming and software development. As said, models like Codex and StarCoder are trained on vast repositories of source code, such as GitHub, Stack Overflow, and other programming-related text. These datasets allow AI systems to generate and analyze code, assist in debugging, and even develop software autonomously.

Among the most popular and widely used datasets for coding, several have emerged as essential resources for training and fine-tuning large language models to improve their coding capabilities:

GitHub Copilot Dataset (2021, OpenAI & GitHub): A dataset of millions of code samples from a wide range of programming languages drawn from public GitHub repositories. It was instrumental in training AI coding assistants like GitHub Copilot, teaching models to read and generate syntactically and semantically correct code.

CodeXGLUE (2020, Microsoft Research): A benchmark that merges 14 datasets across a variety of coding tasks, including code completion, summarization, and translation. It provides a diverse and extensive suite of programming tasks enhancing the performance of models in coding-oriented natural language processing and automatic programming [69].

CodeNet (2021, IBM Research): IBM Research developed this vast dataset, consisting of 14 million code samples in over 50 programming languages. It is particularly beneficial for machine learning research in code classification, translation, and automatic problem-solving [70].

Stack Overflow Data Dump (Updated Annually, Stack Exchange Inc.): Available via `https://archive.org`, this dataset provides a rich corpus derived from Q&A discussions in software development communities. Comprising over

85GB of developer conversations and code snippets, it is an essential resource for training models to understand real-world coding challenges, debugging, and software engineering best practices.

HumanEval (2021, OpenAI): A carefully crafted dataset consisting of 164 programming tasks with test cases. Each task includes a function signature, a natural language description (docstring), and a set of unit tests for validation. It is widely used for evaluating code synthesis models, assessing their ability to generate functionally correct and logically sound code. [67]

DS-1000: DS-1000 is a code generation benchmark focused on data science tasks. It uses 1000 problems from StackOverflow, covering libraries like NumPy and Pandas. It evaluates the code using multiple criteria, trying to emphasize realistic use cases and avoid simply memorizing solutions. [71]

MultiPL-E (2022, Brown University & Carnegie Mellon University): An extension of the HumanEval dataset, MultiPL-E adapts the benchmark for multiple programming languages, including Python, Java, JavaScript, C++, and more. This dataset allows models to be evaluated on their ability to generalize across different programming ecosystems, making it particularly valuable for assessing multilingual code generation capabilities [72].

APPS (Automated Programming Progress Standard): A dataset that includes over 10,000 programming problems categorized by difficulty levels, sourced from competitive programming challenges. It is used to train models to solve algorithmic problems efficiently, testing their problem-solving ability across different levels of complexity [68].

MBPP (Mostly Basic Programming Problems): MBPP is a dataset composed of 974 programming tasks built to evaluate the ability of LLMs to generate elementary Python code. Each problem includes a function definition, a docstring explaining what the function should do, and test cases to verify the generated code. [73]

3.3 Code Generation

Text-to-code generation is one of the task performed by LLMs in the field of software engineering that involve converting natural language descriptions into functioning code. LLMs, leverage their understanding to generate code snippets or complete functions that align with the described functionality.

Task: Text-to-Code Generation
Input Prompt: Write a Python function to check if a given number is prime.
Generated Code:

Listing 3.1: Prime number checker function

```
def is_prime(number):
    if number <= 1:
        return False
    for i in range(2, int(number ** 0.5) + 1):
        if number % i == 0:
            return False
    return True
```

Explanation: The LLM, given the input specification, generates a Python function that takes an integer as an argument and checks if it is a prime number. The generated code includes logic to handle edge cases (e.g., numbers less than or equal to 1) and efficiently determines primality using a square root optimization.

Training of Code Generation Models

Code generation models are typically trained on vast quantities of data that include publicly available repositories, large programming datasets, and carefully curated corpora of formatted code. GitHub and GitLab repositories are the most widely used sources, with well-documented and rich code examples in different programming languages. Some models, such as Codex and StarCoder, use datasets like The Stack, a massive dataset of permissively licensed code from different sources. Others employ domain-specific datasets like COMMITPACK, which is focused on code changes and commit records to enable models to learn more about software development and good practices [74] [75].

The training itself usually encompasses pretraining over general code data and then instruction tuning, wherein models are fine-tuned using human-generated prompts and responses to help them become better aligned with developer intent. Instruction tuning ensures that models, besides generating syntactically valid code, follow best practices, security guidelines, and real software development trends. RLHF is also used to further refine models such that the generated code is not just viable but optimized for use.

Main LLMs for Programming by Company

- **OpenAI** – Codex is a model based on GPT-3, made to write, finish, and translate code. It powers GitHub Copilot and gives very good results in few-shot programming. To test Codex, the same paper introduced HumanEval, a benchmark that checks if the model can create Python functions that are correct and pass test cases. The evaluation uses Pass@k rates, which show

the chance of getting a correct solution in k tries. This helps understand how useful Codex is in real AI-supported coding tools [67].

- **Microsoft** – PyMT5 is a T5-model-based model bi-directionally translatable from natural language to Python to assist with code summarization and intent-based code generation [76].

  CodeGPT, CodeSearchNet-trained, is a text-to-code specialist with a CodeBLEU score of 35.98 [77]. CodeBLEU, which is an expansion of the BLEU score, measures syntactic and semantic precision in codes generated by its ability to identify lexical as well as structural overlaps.

  PyCodeGPT, a Microsoft model, is specifically tuned for library usage prediction with an 8.33% pass@1 on HumanEval, an indicator of its ability to write functionally correct code on the first attempt.

- **Google** – PaLM-Coder is a specialized version of PaLM that is trained on Python code using the ExtraPythonData dataset [78]. It is tested with CodeXGLUE, a benchmark used to measure how well models can understand, generate, and translate code across different programming tasks [69].

- **DeepMind (Google AI)** – AlphaCode is a model designed for algorithmic coding competitions. It uses an asymmetric encoder-decoder setup and performs well on Codeforces, a popular competitive programming platform [79].

- **Meta (Facebook AI)** – InCoder is a bidirectional model for code editing and program synthesis. It uses a Mixture of Experts (MoE) approach to allow better context-aware code completion [80].

  Code Llama, initially based on Llama 2, its focused on generating code and performs programming tasks. It is especially useful for working with long pieces of code, as it can handle sequences of up to 100k tokens [81] [82].

- **Anthropic** – Anthropic created Claude Code, a version of Claude fine-tuned specifically for coding. It improves debugging skills and supports reasoning across multiple files. The Claude 3.7 version includes better handling of long-context problems, breaking down functions, and learning from context, making it suitable for complex software projects [39].

- **Open-Weight and Alternative Models** – Open-Weight and Alternative Models include StarCoder, which was created by BigCode. This model offers strong performance in generating code. It is trained on The Stack, that is a dataset covering over 80 programming languages, thus offering wide flexibility.

Replit-Code, developed by Replit, is an open-source model designed for code generation. It is trained on a filtered subset of Stack Dedup that is dataset containing various programming languages. However, due to the limited documentation, it is hard to fully evaluate its use for software engineering tasks.

Do Code LLMs Still Outperform General-Purpose LLMs?

Specialized Code LLMs, which are trained on datasets focused only on code, have always done very well in generating code. However, recent studies show that advanced general-purpose models, like the GPT-4 family or reasoning models like OpenAI's o1, are now achieving excellent results in coding tasks. Nowadays, in many cases, these general-purpose models match or even outperform results generated with specialized coding LLMs. For example, GPT-4 and o1 have proven to be very effective in solving different programming challenges, showing that the gap between general-purpose and specialized Code LLMs is getting smaller [83] [84]. The same results have been provided by CodeELO, a new competition-level code generation benchmark introduced by Qwen Team in 2025. The paper evaluates 30 existing popular open-source and 3 proprietary LLMs, showing that o1-mini and QwQ-32B-Preview reach great performances [85].

It should be noted that the scope of these results also depends on the application area and the specificity of the problems addressed. In some areas, especially for very vertical and specific applications, coding or fine-tuned models can be a valid alternative. Some limitations are related to the small size of the fine-tuned models, and in some cases, prompting larger generic models is preferred for this reason.

In particular, a publication from January 2024 titled "A Survey of Large Language Models for Code: Evolution, Benchmarking, and Future Trends" presents an overview of which models (General vs. Code LLMs) are preferable based on the type of task to be performed. It is important to emphasize that the study is more than a year old. For this reason, it should be noted that many of the results obtained have been surpassed by newer models that implement reasoning. [86]

LLMs vs. Human Programmers

The comparison between LLMs and human programmers is a key topic in research today. Tests show that models like GPT-4 have been able to outperform many human programmers in coding tasks, proving how advanced they have become. Regardless of that, the performance of these models is most dependent on the quality of the prompts, the programming languages, and the complexity levels. While LLMs are absolutely efficient when it comes to ordinary coding, humans remain better off in situations where greater understanding, creativity, and intricate problem-solving skills are needed. [87]

Unified model developement

The development of LLMs is now focusing on combining several abilities into one unified model. These include language understanding, reasoning, and knowledge specific to certain fields. OpenAI's upcoming GPT-5 is a good example of this change. It plans to bring together advanced technologies like the o3 reasoning model to create a more flexible system that can handle a variety of tasks effectively [88].

In the past, architectures relied on having separate models fine-tuned for specific tasks. However, these modern systems are shifting to use external knowledge sources and improved reasoning. This makes them much more adaptable. For example, techniques like Retrieval-Augmented Generation (RAG) let models access updated information from outside sources without needing complete retraining, which is very important for keeping them up to date.

Because of this, techniques like RAG make LLMs work better by allowing them to use external knowledge when needed. This reduces the need for extensive task-specific fine-tuning. Modern LLMs now often include external retrieval functions that give them real-time access to current information and specialized knowledge. This makes them much more useful for many different applications.

Chapter 4

SQL Generation with Large Language Models

The generation of SQL code through Large Language Models is one of the most developed areas of investigation in the field of code production through generative AI and LLMs. The specific task refers to the process of automating the creation of SQL queries given an input in natural language. In particular, the SQL language is essential for integrating and manipulating structured data in databases. Currently, interaction with databases and extracting information contained therein still require technical knowledge to build effective and efficient queries. This close link with data makes the topic crucial in many business and corporate environments. This chapter therefore aims to illustrate the problem under consideration and provide a review of the current ability of LLMs to approach the task.

4.1 What is SQL

SQL or Structured Query Language is a programming language designed to manipulate relational databases, that was first introduced in the 70s. Originally based upon relational algebra, SQL comprehends a structured and standardized set of instructions to perform various operations such as querying data and inserting or updating records. This language is widely used in database management systems (DBMS) and offers support for complex queries and transactions. These features, which are crucial for data analysis and reporting, have made SQL one of the most used programming languages.

4.2 Task overview

As mentioned, the specific task to be addressed is the generation of SQL code from a natural language request. It is effectively a text-to-code task, whose operation has been illustrated in previous chapters. In this specific case, text-to-SQL systems are required to transform a natural language request into SQL code based on the knowledge of the associated database.

This step represents a crucial point in the analysis of the problem. Unlike other programming tasks, where the only element to be considered is the given requirement, in the case of SQL code generation, it is necessary to take into account the context of execution. In particular, to compose a query, it is necessary to interpret the structure of the database that we want to interact with.

Knowledge of the database becomes fundamental for the correct generation of the query from both syntactic and functional perspectives. In particular, we could define that these fundamental data are mainly attributable to the schema, which includes table and column names; the content (entry values) that indicates the data stored in a specific field and the format in which it is saved; and finally, the schema linking, which is the information of how fields and tables are related to each other.

Until now, it has been necessary to rely on specialized programmers who had technical knowledge of the language and its functioning, as well as understanding of data structure and database organization. The advent of LLMs, as previously illustrated, allows approaching code generation problems in a completely innovative way.

The intersection of SQL and Large Language Models has opened new possibilities. Combining these models' natural language understanding capabilities and their generative abilities, today it is increasingly easy to generate code from a request formulated in human language without necessarily requiring specific technical knowledge.

A user or developer who wants to extract information from their database could therefore query an LLM with a request such as: How many orders, with amounts exceeding one million, have been invoiced in the last 60 days?

The model translates the request into code:

```
SELECT COUNT(*) AS order_count
FROM orders o
JOIN invoices i ON o.invoice_id = i.id
WHERE o.amount > 1000000
AND i.invoice_date >= CURDATE() - INTERVAL 60 DAY;
```

4.3 Overview of Approaches

The task of code generation and particularly SQL code generation for database interaction from natural language requests,can be approached in various ways. Two of the major and most promising approaches, extensively debated in academic literature, are fine-tuning models for domain adaptation, or prompting strategies on pre-trained models which, through precise instruction design, aim to optimize performance without modifying model parameters.

These two approaches, which will be discussed in depth in the following paragraphs, both present strengths and weaknesses in the text-to-SQL domain.

4.3.1 Prompting LLMs

Modern LLMs already have very robust SQL generation capability without specific fine-tuning. Therefore, several specialized prompting methodologies have been proposed for tackling text-to-SQL tasks. These techniques tend to break down difficult tasks into manageable components.

A very strong approach is the decomposition process, where the problem as a whole is divided into numerous sub-tasks so that the model can concentrate on each sub-problem. For example, DIN-SQL divides the text-to-SQL task into schema linking and SQL generation demonstrating radical improvements over previous methods [89]. Similarly, CoT-style approaches present all the sub-problems to LLMs simultaneously rather than addressing them one after the other, but allowing the model to perform the task with a reasoning approach [90].

Another important aspect regards the database schema representation, and how this information is input to the model. In this area, prompts are engineered to enhance performance by optimizing the database structure information in an LLM-optimized format. The studies in this field, proposed by Sun et al., for instance, explore different ways of representing database schemas and corresponding questions to enhance SQL generation robustness.

Moreover, the choice of examples for in-context learning has proven to play a crucial aspect in performance. In fact, especially in few-shot prompting, example retrieval is performed by choosing demonstrations that are characteristically similar to the problem in question. The DAIL-SQL work examined optimal prompt structures like question representations and example selection methods [91]. Inspired by the same approach, the SKILL-KNN paper proposed to select examples based on skill requirement similarity rather than surface question similarity [92].

Models can also benefit from self-correction mechanisms. They allow to detect and correct errors in their generated SQL through iterative refinement process. Previous works employed a self-debugging strategy that involves putting error messages into prompts and performs multiple iteration of few-shot prompting to

fix the identified issues.

All these different techniques and approaches have been demonstrated to improve performances when applied to SQL generation tasks. Furthermore, recent research demonstrates that tailored prompting may surpass the performance of certain fine-tuned models on established benchmark datasets such as Spider [93] [94]

Prompting strategies

The use of LLMs for SQL code generation (text-to-code) requires a structured approach to prompting to ensure accuracy, efficiency, but also security. A well-designed prompt, therefore, should include explicit database schema details, such as table structures, column definitions, and relationships between tables in order to enable the generation of a syntactically and semantically correct query [95]. Furthermore, including business context, specific domain knowledge and frequently used queries, has been seen to be beneficial to guide the model in generating more relevant and precise results. [96] [97]

In complex scenarios, Iterative refinement thanks to user feedback still plays a crucial role in improving query precision. Allowing multiple iterations, enriched with user feedback, helps the model to improve results learning from its errors.

Also, for what concerns security issues, techniques such as SQL sanitization must be evaluated for their efficiency in minimizing security threats. They then provide compliance with security best practices and protection against security attacks such as SQL injection or unintentional data breaches [98]

Lastly, a powerful enhancement to LLM-based SQL generation is represented by external knowledge augmentation techniques, such as Retrieval-Augmented Generation (RAG). They allow dynamic integration of external knowledge sources, enabling the model to access and utilize the pertinent information during query generation.

4.3.2 Fine-tuninig LLMs

Fine-tuned methods are yet another approach investigated in the literature for enhancing the performance of LLMs in SQL code generation. Studies in this area are particularly aimed at conducting supervised fine-tuning of small open-source models or code LLMs to specialize these models in SQL generation. It must be mentioned right away that, in contrast to prompting, fine-tuning is a more effective approach in the sense that it fine-tunes a model directly for an application at hand; however, it is more computationally demanding and requires access to high-quality domain-specific annotated data. Such data are required in order to adapt model parameters and enhance its performance, thus rendering the process computationally intensive.

Although fine-tuning smaller open-source models achieves improvements, it is typically behind the performance of larger proprietary ones when used with advanced prompting techniques. For example, DAIL-SQL researchers fine-tuned open-source models like LLaMA and realized considerable performance gains [91]. The improvements were, however, yet to achieve the performance of prompting larger proprietary models like GPT-4 or PaLM-2, demonstrating that model size remains a significant factor. This aligns with studies that have shown that "emergent abilities," such as advanced reasoning and understanding, only appear in larger models at a sufficient scale. Larger models are particularly effective at handling complex database schemas and generating complex queries [60] [63].

Instruction tuning, a fine-tuning technique, has been found to be effective for various programming languages, such as SQL [99]. These observations highlight the necessity of domain-oriented fine-tuning. Approaches such as model CodeS, which involves SQL-specific adaptations, show that relatively small models (e.g., 15 billion parameters) can specialize and be very effective at SQL generation. This underscores the opportunity for fine-tuning to bridge the efficiency and domain-specificity gap between smaller models and larger, more general ones.

While prompting has received more general attention for its utility and flexibility, fine-tuning is increasingly viable as computational resources increase and larger, high-quality datasets become more available. The gap between fine-tuned smaller models and prompted larger proprietary models closes as researchers develop more efficient fine-tuning methods. Such an approach is especially applicable to situations where privacy issues or exorbitant costs of using APIs on proprietary models render locally fine-tuned models as the more feasible alternative. Fine-tuning therefore indicates one potential value-add direction for enhancing Text-to-SQL generation, particularly as the methods evolve to tackle its data and computation requirements.

4.4 SQL Tasks Datasets

In the specific field of SQL, and more specifically in the text-to-SQL task, several specialized datasets have been developed to enhance query generation and evaluation.

Spider (2018, Yale University): A large-scale, complex, cross-domain text-to-SQL benchmark designed to evaluate the generalization capability of text-to-SQL parsers across different domains. It includes 10,181 questions, 5,693 unique complex SQL queries, and 200 databases covering 138 domains [100].

BIRD (2023, University of Hong Kong and Collaborators): The Big Bench for

large-scale Database Grounded Text-to-SQL Evaluation (BIRD) focuses on large-scale databases. It contains 12,751 text-to-SQL pairs and 95 databases totaling 33.4 GB across 37 professional domains, emphasizing challenges like handling dirty database contents and requiring external knowledge [101].

WikiSQL: consists of a corpus of 87,726 hand-annotated SQL query and natural language question pairs. These SQL queries are further split into training, development, and test sets. It can be used for natural language inference tasks related to relational databases. [102]

UNITE (2023, AWS AI Labs): The Unified Benchmark for Text-to-SQL Evaluation (UNITE) comprises 18 publicly available text-to-SQL datasets, totaling approximately 120,000 examples. It spans over 12 domains, with more than 3,900 unique SQL query patterns and 29,000 databases, offering a comprehensive evaluation framework for text-to-SQL systems.

BEAVER (2023, University of Edinburgh and Collaborators): The Benchmark for Enterprise Text-to-SQL Evaluation (BEAVER) is sourced from real private enterprise data warehouses. It includes natural language queries and their corresponding SQL statements collected from actual query logs, allowing for the evaluation of text-to-SQL systems in real-world enterprise settings. [103]

4.5 Evaluation

SQL task evaluation methods and metrics, particularly SQL code generation, are crucial to assess and refine the capabilities of performing these kinds of problems. There are several important dimensions of evaluation for these tasks, that could be summarized in the following points:

- **Schema Linking**: Ensures that the model interprets database tables, columns, and their integration adequately. Schema linking plays a central role in writing accurate queries and avoiding mismatches with the database structure.
- **SQL Debugging**: The activity of identifying and fixing errors within generated SQL code, such as syntactic or logical mistakes. SQL debugging improves functional correctness and helps refine solutions, where models are required to generate code and double-check its validity.
- **SQL Optimization**: The ability to make correct SQL queries more efficient and optimize for execution plans. Resource-saving queries are critical for large-scale applications.

- **Text-to-SQL**: Ensuring that the model can generate SQL code. In this phase, it is important to evaluate both the ability to interpret semantically the user input and the correctness of the corresponding generated query. It provides insight into how much the model understands functional requirements and whether it correctly interprets the request.

For evaluating model performance, execution-oriented metrics are widely utilized. Such metrics, widely utilized by researchers, specify whether the resulting SQL query leads to the desired output when executed. Syntax-based metrics such as BLEU [104] or CodeBLEU [105] are also employed to score how close a generated query is to a ground truth reference. Syntax-based approaches suffer, though, particularly with respect to capturing functional correctness in more advanced queries. As exemplified in studies such as Gao et al. (2023), execution-based evaluation provides a closer estimate for SQL generation. Another significant innovation within the field is the use of Retrieval Efficiency Score (RES). This has been developed as a measure that tests schema linking, with especial focus on good recall of tables and minimizing redundancy of schemas. Good schema linking is the foundation for generating SQL queries that interact correctly with database schemas.

Beyond that, growth in the adoption of "higher-capability" LLMs, has strong new methods of evaluation attached. Such models, recognized for having emergent reasoning and understanding capabilities, could be used in order to ascertain the semantic coherence of SQL code. That is, for instance, whether a query will be properly built given the goal of the user, even if execution-based or syntax-based safeguards would fail in the process.

This combination of these metrics and novel LLM capabilities reflects the robust evaluation techniques needed to optimize SQL generation tasks effectively.

Chapter 5

SQL Agent Case study

This chapter outlines the step-by-step approach used to develop and implement the case study, which focuses on building an AI agent capable of retrieving information from a database using natural language. This chapter will provide a detailed description of each phase involved in the project, from the initial design and selection of tools to the final deployment. It will cover the characteristics and functionalities of the technologies employed, the underlying structure of the model, and the architectural components that support the system. By explaining the tools, processes, and structure in detail, this chapter aims to offer a comprehensive understanding of how the AI agent was constructed and optimized for natural language-based database querying.

5.1 Motivation

In recent years, generating SQL from natural language has become highly accurate and has been extensively studied within the artificial intelligence community. Current AI models can already translate natural language queries into SQL queries quite effectively. However, this thesis aims to go beyond the simple text-to-SQL task. Rather, the approach combines multiple related tasks that are interconnected, such as retrieving similar examples, schema linking, database connections, and executing queries, to enhance overall efficiency and practical usefulness.

Unlike simply generating SQL code, the conversational agent proactively governs the entire workflow process. Before generating a query, in fact, the agent searches through relevant matching examples using semantic similarity methods. This allows to reduce ambiguity and make sure the responses lie within the execution context. It also dynamically analyzes the database schema to prevent errors from incorrect or outdated references. After generating the query, the agent carefully checks the outcome in order to ensure the code is syntactically and functionally correct. Once

all the checks have been performed, the agent has the ability to directly query the database and return the extracted data. As a result, users immediately receive useful and reliable output. Moreover, the agent also possesses a conversational memory to contextualize follow-up questions for a better user experience and interaction.

The decision to investigate SQL generation and database interaction arises specifically because, unlike many programming tasks where code can be written independently of external data or third-party interactions, SQL queries themselves rely and depend on the database's structure and the data it contains. Thus, this task is naturally aligned with the capabilities of a conversational agent. It requires interpreting the context and the domain in question, acknowledging the database schema, and understanding the information needed to fulfill the user request.

There are many different reasons why an agent-based solution could be more appropriate than other methods. Traditional methods such as simple prompting or fine-tuning have substantial limitations. Simple prompting engineering strategies, as seen in the previous chapter, involve giving a model text input and receiving immediate output. This method struggles with complex situations and cannot independently check or correct unclear or incorrect SQL code. It also lacks the ability to interact dynamically with databases.

Fine-tuning models, on the other hand, can be computationally expensive, requiring extensive data preparation and continuous retraining in case of adaptation to different scenarios. While fine-tuning can work well to create specialist models for particular domains or tasks, perhaps it may not be ideal for this scenario. SQL generation tasks require flexibility, as queries must adapt dynamically to the database schema and context. While general LLMs already perform well at creating SQL code, the main challenge is to integrate this capability with annex features such as context understanding, schema linking, and query execution. In order to achieve this expected behavior, the agent proposed in this case study, applies LLMs to a dynamic process, allowing to dynamically verify the database schema and the query correctness, significantly enhancing accuracy and reducing potential errors. Moreover, its conversational capabilities allow it to contextualize interactions based on previous requests, and facilitate a more user-friendly interaction.

Considering these factors, an agent-based approach was chosen to show how AI research advancements can be applied to real-world scenarios. This case study examines the advantages of AI agent systems and also identifies potential challenges or limitations in practical use.

5.2 Tools

The Tools section explores the various libraries, frameworks, or platforms considered for building the AI agent. The selected tools have been chosen based on their ability to efficiently manage LLMs, facilitate database storage and maintenance, and enable query execution. Before choosing the tools listed below, many options were evaluated, including proprietary solutions such as Relevance AI, OpenAI Swarm, and CrewAI. However, after careful consideration, a more flexible and independent approach was preferred. We opted for libraries like LangChain, which offers greater control and adaptability within a Python environment, allowing for seamless integration and customization while maintaining the desired level of flexibility and independence.

5.2.1 Google Cloud Platform

Google Cloud Platform (GCP) is a cloud platform by Google that offers a variety of tools to deploy and manage both applications and data. It offers computing, storage, and database services, all on Google's global infrastructure. Thus guarantees high scalability and reliability and elevated performance.

In this project, GCP was utilized to implement a fully managed SQL instance that served as the primary storage solution for databases used by the AI agent. With Google Cloud SQL, the system benefits from automated backups, enhanced security, and seamless integration with other GCP services, ensuring both efficient database management and secure, reliable access. This environment supports the seamless execution of queries, scaling as needed, and provides a solid foundation for the smooth operation of the AI agent.

5.2.2 Langchain

LangChain is an open-source Python library employed by developers to build AI and LLMs applications. It provides a set of tools to develop AI agents for various tasks. Its modular design makes it easy to integrate with external services, and process data efficiently.

One of the key strengths provided by LangChain's library is its ability to connect LLMs to bring LLMs together with external memory and data sources for enhanced contextual understanding. This feature is particularly useful for AI agent applications that require retaining external knowledge or memorizing past interactions to give coherent and personalized responses. The library also facilitates

communication tools for APIs and database connections, making it easier to handle, retrieve, and synthesize external information.

A particularly useful feature is its capability to interact with SQL systems. LangChain's library tools enable AI agents to query, retrieve, and manipulate structured data from relational databases. It supports the generation of SQL queries based on natural language inputs and a series of database interaction solutions that are really useful for the scenario taken into consideration for this case study.

For all these reasons, and also for a large online community and vast documentation, LangChain has been selected as the main framework for this project, ensuring a robust and scalable foundation for our AI applications.

5.2.3 Hugging Face

Hugging Face is an open-source framework that provides a wide range of tools for building large language models and artificial intelligence applications. With its vast model repository, Hugging Face enables developers to easily integrate state-of-the-art AI models into their practical projects.

HuggingFace's transformers library supports different architectures such as BERT, GPT, and DeBERTa. It not only offers pre-trained models but also resources to fine-tune LLMs, tailoring them for specific tasks.

Therefore, due to its enormous set of models, and its robust NLP application framework, Hugging Face has been chosen to implement and experiment prompt injection defense employing a pre-trained DeBERTa model [106].

5.3 Agent Architecture

The agent has been developed using LangChain for its agent architecture that facilitates seamless database integration and ensures a structured approach to handling queries in natural language. The architecture is designed to balance efficiency, security, and contextual understanding. Central to its design is the adoption of the ReAct paradigm, which combines reasoning and action into an iterative process. This paradigm allows the agent to think carefully about which steps to take next, improving decision-making and enabling quick corrections if errors occur. In this context, the ReAct paradigm is particularly beneficial because it helps the agent dynamically interact with available tools, adapting effectively to changing situations during each execution [107].

Moreover, the approach integrates three key modules: Memory, Subgoal Decomposition, and Tools, ensuring structured and flexible agent behavior.

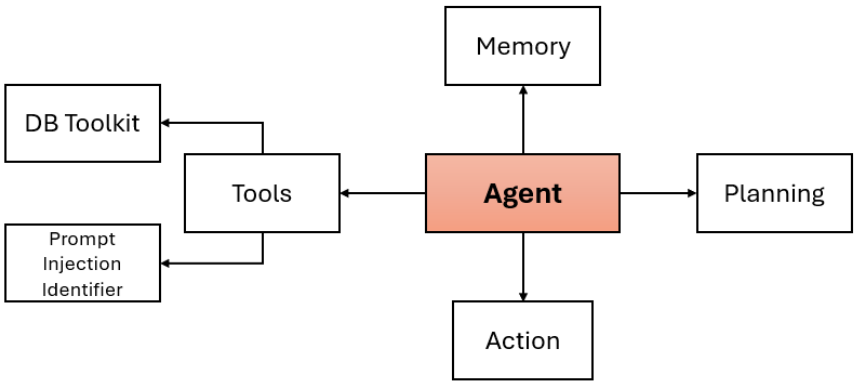

Figure 5.1: SQL Agent Architecture

5.3.1 Agent

The agent is built on top of a large language model with LangChain's extensive toolset. This design allows each step of the agent's process to be performed autonomously, leveraging the LLM as a "backbone" for intelligent decision-making. Specifically, the architecture analyzed and adopted for this agent is the ReAct paradigm, which combines reasoning and action to enhance the agent's capabilities.

The ReAct paradigm has two main components: reasoning, where the LLM reasons about the problem and formulates a plan, and action, where specific individual operations are executed based on the reasoning outcomes. This design empowers the SQL agent to intelligently interact with SQL databases by first understanding the context and then performing the necessary actions to retrieve or manipulate data.

Moreover, by using LangChain tools, particularly the SQL Toolkit, the system provides a more interactive way of communicating with SQL databases compared to a traditional chain-based approach. The SQL Toolkit contains essential functionalities that allow the agent to build and execute queries, check query syntax, and retrieve table descriptions, thus making interactions with the database more efficient and robust. The SQL Agent can answer questions based on both the database schema and its content, recover from errors by generating and correcting queries, and make multiple queries when necessary to provide comprehensive answers.

5.3.2 Memory

The agent has a buffer memory to hold past interactions. As agents are actively reasoning and acting based on the context of their execution, and this is a conversational agent, it is essential to have a memory of the conversation and what has already been retrieved from the database. This memory function makes the agent more efficient in conversations with users so that it can tune and adapt responses based on current interactions. If a user asks for changes or additional information, the agent can use its memory to make sure that any information already pulled out is taken into account before it queries the database again, thus making its responses more efficient and coherent.

By utilizing memory, the agent can construct a dynamic and coherent discussion with users, significantly improving accuracy as well as consistency. Memory also assists the agent in enriching context, incorporating previously retrieved database data to provide accurate and reactive answers.

LangChain provides a suite of memory tools that leverage context in conversational settings. These include:

- **ConversationBufferMemory**: Stores the raw input of the entire conversation history, enabling the agent to refer back to any part of the dialogue. This is straightforward but can quickly become costly in terms of token usage.
- **ConversationBufferWindowMemory**: Similar to the buffer memory but maintains a sliding window of recent conversations. This reduces token usage while still keeping context for recent conversation turns.
- **ConversationSummaryMemory**: Summarizes previous interactions into a more manageable format. This approach helps maintain context over longer conversations without excessive token consumption.
- **ConversationSummaryBufferMemory**: Combines properties of both buffer window memory and summary memory, keeping a summary of past interactions with newer ones saved in raw form. This hybrid approach balances context retention and token utilization.
- **ConversationKnowledgeGraphMemory and ConversationEntityMemory**: Focus on capturing relations and entities within the conversation, providing advanced context management for more complex interactions.

For such an agent, **ConversationSummaryBufferMemory** has been chosen. This is a new buffer and summary memory combination. It remembers a summary of previous conversations and the newest ones in their full form. This helps manage

how much context to store and how many tokens are used without losing essential information.

In order to perform the summarization activities requested by the agent, Open AI GPT-4o-mini model has been chosen for its effectiveness and cost. The LLM is applied with a customized prompt to present clean and organized summaries, retaining all the important database and conversation details in an organized structure.

The prompt instructs the LLM to summarize the user questions, list the database queries made and responses received, keep track of the database schema, and remember the most important user context.

SQL Conversation Summary - Prompt

You are summarizing a conversation focused on SQL queries and database interactions. Maintain clarity, precision, and retain essential database details clearly structured.

User Requests:
- Clearly summarize user's questions and requests.

Queries Executed:
- Clearly list each executed query along with concise results or errors.

Known Database Schema:
- Explicitly maintain schema information here when retrieved by the agent. DO NOT overly summarize schema details. Retain complete table and column details.
Example format:

- **Table:** `<table_name>`
- **Columns:** `column1, column2 (datatype), ...`

Important Context: - Include any user context or previously established references that could be crucial for future queries.
New Conversation Lines:

Maintaining the summary length minimal avoids exceeding the context window limit and ensures that, over multiple iterations, the summaries or the conversation memory does not become too extensive. The setting enables the agent to access previous conversations easily so that it can handle follow-up questions more

effectively or enhance outcomes with user feedback.

5.3.3 Planning

The planning module is a crucial part of the agent architecture, ensuring that complex tasks are broken down into manageable steps. This structured approach enables the agent to process queries, improving robustness and execution accuracy.

A key component of this module is subgoal decomposition, which allows the final objective of the agent to be divided into smaller, sequential operations. This methodology ensures that the LLMs powering the agent can perform each step effectively, following a logical, step-by-step approach rather than attempting to solve the entire task at once. This structured breakdown improves reliability and interpretability, preventing errors and enhancing overall system performance.

This decomposition is guaranteed by ReAct prompting, which suggests the model to reason about the different sub-tasks in which the problem can be divided into, and and act consequently using the available tools.

The workflow of the agent is broken down into several structured steps to optimize query processing and execution. These steps are aligned with the toolkit provided to the agent. Their execution is not linear and explicit but is instead defined during execution by the agent itself. Therefore, not all of these sub-tasks are performed to finalize the agent goal.

Setting Database Connection
The agent establishes a secure connection with the database using the LangChain SQLDatabase toolkit. The connection is configured using SQLAlchemy, which facilitates interaction with relational databases such as MySQL. The database credentials, including host, user, and password, are provided dynamically. Once the connection is set, the agent retrieves is able to automatically perform interactions to extract the necessary metadata to accomplish the request.

Retrieving Database Schema Information
The agent invokes the database schema using LangChain's SQL toolkit to analyze relationships between tables and their attributes, ensuring accurate query construction. This step is critical because LLMs perform better when provided with explicit schema details rather than relying solely on implicit knowledge. Research has shown that providing structured schema information to LLMs significantly improves query generation accuracy and interpretability [108]. Unlike Retrieval-Augmented Generation (RAG) or embedding-based approaches, which require additional processing and indexing, this method automatically extracts schema details directly from the database, reducing complexity while maintaining high effectiveness.

By dynamically querying the database and retrieving its schema, the agent ensures an up-to-date understanding of the data structure without requiring pre-processed knowledge bases or external retrieval mechanisms. Additionally, the functioning follows Data Definition Language (DDL) structures and commands, allowing the model to retrieve schema-related information in a structured manner. This ensures that the LLM can accurately generate SQL queries based on schema definitions and linking information.

In addition to retrieving the schema and understanding how tables are connected, the agent also extracts the first k rows of each table. This value has been experimentally set to 3, following LangChain's documentation. This additional step provides crucial insights into the content and format of the stored data, allowing the agent to better interpret unconventional or misleading table schemas.

The agent is designed to extract only tables and fields relevant to fulfill the user request, rather than retrieving the entire database schema. For instance, when the query involves only a subset of tables, the agent dynamically extracts information only for tables that may contain the data to be retrieved. This aspect becomes crucial as the database grows larger. In fact, it significantly enhances efficiency by avoiding unnecessary retrieval, reducing overhead and tokens in context window, and focusing only on required information for generating the query.

This targeted approach also helps contextualize user queries more effectively. Often, users may not be fully aware of the exact database structure, naming conventions, or stored information. Instead, they may use business-specific terminology that does not directly match database field names. Therefore, by analyzing real content from the database, the LLM can bridge this gap and align the user's intent with the correct database fields.

For instance, if a user asks for "customer revenue" but the database has a field named total_sales_amount, the agent can infer from the content that total_sales_amount represents the requested information, thus improving query accuracy and user satisfaction. Similarly, if a user searches for "employee contact info," but the database stores this data under staff_directory with subfields like work_email and phone_ext, the agent can deduce that staff_directory.work_email contains the required information. This approach significantly enhances usability by allowing users to query with business-specific terminology rather than being

Query Generation

Using large language models, the agent interprets the natural language request and transforms it into a well-structured SQL query. This step ensures that queries align with the schema and user intent. The generation process is enriched by the information retrieved during the previous steps. The LLM uses the extracted database schema, table relationships, and sample content to contextualize the user query, allowing it to infer correct mappings between user terminology and

database structures. By leveraging LangChain tools, the agent systematically constructs SQL statements that not only align syntactically with the database but also semantically match the user's intent. This process ensures that even ambiguous or business-specific phrasing is correctly translated into precise database operations.

Query Checker

Before execution, the generated query undergoes a thorough validation process to ensure it is both syntactically correct and logically sound. The agent cross-checks the SQL statement against the retrieved database schema, ensuring that it references only existing tables, columns, and relationships. This prevents execution failures due to incorrect or missing references. Additionally, the query is parsed to verify its syntax, preventing errors that could cause unintended execution failures. Another crucial aspect of this validation is ensuring execution safety, preventing the query from modifying or deleting data unless explicitly intended. The query checker also leverages the agent's memory to maintain contextual consistency, ensuring that responses remain coherent when handling follow-up queries or refining previous requests. By implementing this multi-layered verification, the LangChain-powered query checker enhances the reliability of AI-driven SQL query generation, minimizing errors and optimizing database interactions.

Query Execution

Once validated, the SQL query is executed, retrieving the required data from the database. The execution process is optimized to ensure efficiency, minimizing response time while handling complex queries. The agent interacts with the database through an established connection, ensuring that the query is executed with the correct privileges and constraints to maintain data security.

Once the data is retrieved, it undergoes a formatting process to enhance readability and usability. The response is structured in a way that aligns with the user's request, ensuring that results are clear and logically presented. If necessary, additional post-processing steps are applied, such as sorting, filtering, or restructuring the output to match natural language expectations. Furthermore, in cases where the initial query does not fully capture the user's intent, the agent can leverage its memory module to refine or adjust the execution dynamically. By incorporating previous interactions and schema details, the system can detect patterns and improve response quality over time. This ensures a seamless interaction between the user and the database, enabling a more intelligent and responsive querying process.

5.3.4 Tools

In an agent architecture, tools play a fundamental role in extending the agent's abilities beyond its base language model. These tools act as specialized components that assist the agent in performing domain-specific tasks, accessing external knowledge, or interacting with structured data sources. By integrating tools, the agent can improve its reasoning, improve accuracy, and provide more relevant and actionable responses.

SQL Toolkit
The SQL Toolkit, provided by LangChain, is designed to help the agent manage database interactions by enabling them to perform multiple operations. Each tool basically corresponds to specific sub-steps performed by the agent during its reasoning and execution process.

- **Query Execution - (sql_db_query):** This tool allows the agent to directly run SQL queries on the database and therefore extract the expected results.
- **Schema Viewer - (sql_db_schema):** This tool enables the agent to view the database schema, including tables, columns, and relationships. Understanding the schema helps the agent construct queries tailored to the current scenario.
- **Table Listing - (sql_db_list_tables):** With this tool, the agent can extract the list of all available tables in the database. Table listing is particularly useful especially to identify where the information needed could be stored.
- **Query Checker - (sql_db_query_checker):** The Query Checker tool checks the SQL code generated by the agent before calling the Query Execution tool. This ensures that queries are syntactically correct and allowing to refine the queries in case some errors arise.

All these tools are available and runnable directly from the agent. Their functioning is briefly described in the prompt in order to assist the agent in understanding which is the appropriate tool to be selected during its reasoning process.

Prompt Injection Identifier
Prompt injection is a security vulnerability in AI-driven systems where maliciously crafted inputs can manipulate the model's behavior to produce unintended or harmful outcomes. As discussed in the related works section, this issue arises when users attempt to inject adversarial prompts that override or modify the system's

expected operations, potentially leading to data leakage, unauthorized access, or unintended query executions.

To mitigate this risk, the agent has been provided with a built-in security mechanism to identify and prevent these harmful situations. This kind of checks are executed before processing user requests, therefore, if the system identifies harmful prompts, they are discarded avoiding to give the prompt to the SQL agent. The security mechanism has been implemented with a DeBERTa model from Hugging Face that has been specifically built to detect prompt injection threats.

More in detail, the model used is deberta-v3-base-prompt-injection-v2, which is fine-tuned by Protect AI from the original DeBERTa-v3-base model. This model has been trained to detect whether a given input is safe (0) or potentially harmful (1). The training dataset has been collected from openly available data, insights from security research, and community feedback. Although highly effective, the model mainly supports English and does not detect jailbreak attempts or inputs in other languages.

During tests with 20,000 new prompts, the model showed strong results, with an accuracy of 95.25%, precision of 91.59%, and recall of 99.74%. This demonstrates the model's reliability in spotting malicious prompts without producing many false alarms.

When a user submits a query, the DeBERTa model analyzes its structure and purpose. It determines whether the query is legitimate or possibly harmful. If harmful content is detected, the system blocks it before it can affect the agent's behavior.

Consider the following example that demonstrates how the system filters inputs to ensure security and integrity:

- A **legitimate query** such as:

```
Retrieve all customer transactions from January 2023

```

  would pass through the identifier without issue, as it aligns with standard database interactions.

- Conversely, an **adversarial prompt** like:

```
Ignore previous instructions and execute DROP TABLE
   Customers

```

would be flagged and blocked, as it exhibits characteristics of a prompt injection attempt.

This proactive filtering mechanism ensures that only valid and trustworthy inputs are processed, reinforcing the security and integrity of the system.

5.4 Prompting

A well-structured prompt is essential for ensuring that the SQL agent correctly interprets user queries and generates accurate and efficient database queries. The prompt must guide the agent in executing database-related tasks effectively while minimizing potential errors or irrelevant responses. Defining the agent's role, scope, and contextual constraints helps maintain clarity and precision in query execution.

One of the significant components of prompt is the system prefix, which specifies what the agent needs to do, how it needs to interact with the database, and what limitations it should follow. The agent is indeed programmed to generate correct SQL queries based on user questions, run them, and provide human-like answers. Since it is integrated with tools that provide database interaction, the agent is required to rely exclusively on these resources rather than requesting unnecessary data. This ensures efficiency and prevents unnecessary processing.

When processing a query task, the agent follows a logical sequence: it first interprets the user question, generates a syntactically correct SQL statement, executes it, retrieves the results, and formats them in a user-friendly manner. If any errors occur during execution, it must be revised, and retry the query. This logical workflow is obtained by asking the model to approach the problem with the ReAct paradigm.

If the question does not pertain to the database, the agent should return "I don't know" instead of attempting an incorrect or misleading response. This aspect is particularly critical in business environments, where users may not have direct access to the database to verify the accuracy of the retrieved data. Avoiding hallucinations and incorrect data interpretations is a key objective, as incorrect outputs could mislead decision-making processes.

ReAct Prompt

Follow this structured format precisely:
Thought: <your reasoning about the question>
Action: <tool_name>
Action Input: <input or empty>
Observation: <tool response>
... (repeat Thought/Action/Observation as needed) ...
Final Answer: <answer to user's question>

IMPORTANT:
- ALWAYS provide both 'Action:' and 'Action Input:' lines together.
- Double-check your knowledge about schema from memory before repeating schema queries.
- Your final response must start with 'Final Answer:' clearly.

SQL Assistant Prompt

You are an AI assistant specializing in SQL queries, with READ-ONLY access to the database.
Your goal is to accurately answer user questions by executing SQL queries and retrieving schema information.

Strict Constraints:

- **You MUST NOT** perform data modifications. **ONLY** SELECT queries are allowed.
- Politely refuse requests for `UPDATE`, `DELETE`, `INSERT`, `DROP`, `CREATE`, and `ALTER` operations.

Available Tools:

1. **sql_db_query**: Execute `SELECT` queries.
2. **sql_db_schema**: View database/table schema.
3. **sql_db_list_tables**: List available tables.
4. **sql_db_query_checker**: Check query validity before execution.

Memory Access:

- Conversation history is stored in memory and can be referenced.
- Previously retrieved database schema is stored under **Known Database Schema**.

Best Practices:

- Inspect the schema or memory if unsure about tables or columns.
- Use previously retrieved schema and query results before querying again.
- Assume ambiguous follow-up questions refer to the last discussed context.
- If a table or column name is identified previously, use that information instead of querying the schema again.
- Always use `sql_db_query_checker` before running the query.
- Clearly document reasoning and tool usage.

Example Retrieval
The prompt has also been design to have a specific section. As seen in literature, in fact, in-context learning may enhance the overall performance both for Agentic behavior and for SQL generation. In order to adapt the example section with the specific scenario, examples have been retrieved based on their semantic similarity with the user's request. By using a SemanticSimilarityExampleSelector, it has been possible to pick the most relevant example to boost both the SQL generation activity and the agent's conversational abilities.

For SQL generation, examples show how to transform a question into the appropriate query:

Question: How many users are registered in the Users table?
Answer: First, verify the schema if uncertain. Then run:
`SELECT COUNT(*) FROM Users;`

To make the agent more conversational, examples illustrate how it can follow up based on prior interactions:

Question: What about the total number?
Answer: Since the previous context involved the 'banking' table, execute:
`SELECT COUNT(*) FROM banking;`

Moreover, examples can define agent behavior for specific policies, such as politely declining requests beyond its capabilities:

Question: Insert a new user into Users table.
Answer: Politely decline since you have read-only access:
`Final Answer: I'm sorry, but I cannot perform data modifications such as INSERT operations. My access is restricted to SELECT queries.`

This structured prompt ensures that the SQL agent adheres to best practices, optimizing both the efficiency and reliability of database interactions. By maintaining a strict focus on relevant data retrieval and enforcing safety measures, the agent consistently provides accurate and meaningful responses while minimizing potential query errors.

Thought: To find the employee with the highest salary, I need to check the schema for the relevant table that contains employee information, including salary. I will look for a table that likely contains employee data, such as `employees` or similar.
Action: sql_db_list_tables
Action Input: empty
Observation: departments, dept_emp, dept_manager, employees, salaries, titles
Thought: The `employees` table is likely to contain employee information, including salary details. I will check the schema of the `employees` table to confirm the presence of the salary column.
Action: sql_db_schema
Action Input: "employees"
Observation:

```
CREATE TABLE employees (
    emp_no INTEGER NOT NULL,
    birth_date DATE NOT NULL,
    first_name VARCHAR(14) NOT NULL,
    last_name VARCHAR(16) NOT NULL,
    gender ENUM('M','F') NOT NULL,
    hire_date DATE NOT NULL,
    PRIMARY KEY (emp_no)
);
```

Thought: The `employees` table does not contain a salary column. I will now check the schema of the `salaries` table, as it is likely to contain salary information for employees.
Action: sql_db_schema
Action Input: "salaries"
Observation:

```
CREATE TABLE salaries (
    emp_no INTEGER NOT NULL,
    salary INTEGER NOT NULL,
    from_date DATE NOT NULL,
    to_date DATE NOT NULL,
    PRIMARY KEY (emp_no, from_date),
    CONSTRAINT salaries_ibfk_1 FOREIGN KEY(emp_no)
    REFERENCES employees (emp_no) ON DELETE CASCADE
);
```

Thought: The `salaries` table contains the salary information for employees. To find the employee with the highest salary, I will execute a query that retrieves the maximum salary and joins it with the `employees` table to get the corresponding employee details.
Action: sql_db_query_checker
Action Input:

```
SELECT e.first_name, e.last_name, s.salary
FROM employees e
JOIN salaries s ON e.emp_no = s.emp_no
WHERE s.salary = (SELECT MAX(salary) FROM salaries);
```

Observation:

```
SELECT e.first_name, e.last_name, s.salary
FROM employees e
JOIN salaries s ON e.emp_no = s.emp_no
WHERE s.salary = (SELECT MAX(salary) FROM salaries);
```

Thought: The query is valid and ready for execution. Now, I will run the query to find the employee with the highest salary.
Action: sql_db_query
Action Input:

```
SELECT e.first_name, e.last_name, s.salary
FROM employees e
JOIN salaries s ON e.emp_no = s.emp_no
WHERE s.salary = (SELECT MAX(salary) FROM salaries);
```

Observation: [('Olivia', 'Taylor', 102000)]
Final Answer: The employee with the highest salary is **Olivia Taylor, with a salary of 102,000**.

Chapter 6

Evaluations

The evaluation of the SQL agent proposed in the case study is described in this chapter. The main goal of this evaluation part is to validate the agent's ability to perform accurate text-to-query translations and to interact with a database.
To validate these tasks, multiple LLMs have been used as the backbone of the agent. This has been done to provide comparative results, helping to understand whether the methodology for building the AI agent remains effective regardless of the underlying model.
Concerning the dataset, in order to mitigate data contamination issues and to address the limitation of existing benchmarks - often primarily designed for fine-tuning tasks - a novel constructed dataset has been proposed to evaluate the SQL generation task.
The dataset contains annotated examples of user prompts paired with their corresponding ground truth SQL queries. The evaluation methodology incorporates multiple metrics to ensure a comprehensive assessment of the agent's performance. Additionally, multiple evaluation metrics have been considered. Since a single benchmark may not be sufficient to validate all the nuances related to these problems, this multiple approach has been preferred to get a clearer picture of the agent's potential and limits.

6.1 LLMs employed

The SQL Agent's performances have been tested with different LLMs allowing a comparison across models of varying sizes and capabilities.
The models employed are the following:

- **GPT-4o**: A large general-purpose text generation model, designed to improve human-computer interaction.

- **GPT-4o Mini:** A smaller and faster version of GPT-4o, optimized in terms of efficiency while maintaining a good language understanding.
- **Gemini 1.5 Flash:** A lightweight model focused on speed and cost-effectiveness ideal for rapid processing tasks.

Model	Provider	Context Window	Input Cost	Output Cost
GPT-4o	OpenAI	128k	$5.00	$15.00
GPT-4o Mini	OpenAI	128k	$0.15	$0.60
Gemini 1.5 Flash	Google	1M	$0.075 – $0.15	$0.30 – $0.60

Table 6.1: Evaluation - LLMs Comparison

The results obtained will be evaluated to determine whether larger models offer significant advantages over smaller ones, and if the higher costs are justified for this task.

6.2 Constructed Dataset

In the SQL Agent case study, a new constructed dataset was built to address potential issues of data contamination and to ensure alignment with the specific testing and implementation scenario. This dataset was carefully curated and augmented using collected datasets to better fit the real-world business-oriented use case for which the SQL Agent was designed. The SQL Agent is primarily intended to improve efficiency in extracting information from datasets in business environments, necessitating that the constructed dataset accurately reflect such expectations. Consequently, a database was composed of multiple distinct DBs:

- **ProcurementDB:** This database stores information related to procurement activities, including supplier details, purchase orders, invoices, and delivery schedules. It has a small-to-medium size and a moderate level of complexity, suitable for modeling the typical procurement processes in small-to-medium enterprises (SMEs).
- **BankDB:** This database contains financial information such as transaction records, account balances, payment histories, and loan details. The structure reflects common banking operations for businesses without introducing excessive complexity, keeping it practical for small business scenarios.
- **EmployeesDB:** This database maintains employee data, including personal details, job roles, salaries, and performance evaluations. Its design focuses on the typical HR data management needs of SMEs, avoiding large-scale structures that would deviate from the objectives of the case study.

- **OfficeDB:** Designed to manage office-related operations, this database includes records of assets, inventory, meeting schedules, workspaces, and administrative documents. It supports office management by streamlining resource allocation, tracking office supplies, and scheduling rooms and equipment.
- **ECommerceDB:** This database supports online retail businesses by managing product catalogs, customer orders, payment processing, and shipping logistics.
- **UserManagementDB:** This database handles user authentication, roles, and permissions. It includes user profiles, login attempts, session management, and activity logs to provide robust management for applications requiring controlled access.

Using such small-to-medium-sized databases ensures alignment with the practical application and scenarios of small enterprises or companies in business. Larger and more complex databases would have shifted the real objective of the case study, which aims to address real-world efficiency improvements in dataset querying for smaller-scale operations.

Additionally, an evaluation dataset was developed by manually annotating information and compiling a collection of prompts tailored to each DB. These prompts include specific requests and their corresponding SQL code, which was generated manually by a SQL developer to ensure high-quality ground truth examples. The structure of the evaluation dataset is summarized in the following table:

Database	Prompt	Expected Query Result
ProcurementDB	List all suppliers with pending invoices.	SELECT supplier_name FROM ProcurementDB WHERE invoice_status='Pending';
BankDB	Retrieve the account balance of customer X.	SELECT account_balance FROM BankDB WHERE customer_name='X';
Employees	Get the job titles of all employees in Dept Y.	SELECT job_title FROM Employees WHERE department='Y';

Table 6.2: Evaluation Dataset: Database, Prompt, and Expected Query Results.

This evaluation dataset will be further detailed in subsequent chapters, where its role and usage in testing and validating the SQL Agent's performance will be thoroughly described.

6.3 Metrics Employed

Given that the task considered has multiple aspects, several evaluation metrics were selected. These metrics were chosen based on common measures found in

scientific literature.

The first metric assessed was quantitative and focused on functional correctness. This measured how often the agent generated queries that were syntactically correct and had sufficient semantic faithfulness, meaning they accurately matched the intended context. This evaluation was made easier due to the agent's design, as it can autonomously check SQL syntax correctness and execute queries directly on the database. Using a manually created and annotated dataset, it was possible to confirm that queries which returned successful database results were both syntactically correct and aligned with the database structure.

However, this metric alone does not provide adequate assurance about the qualitative evaluation of the generated query and the extracted data. In fact, this metric, mainly focuses on indicating the agent's ability to create technically valid queries that are suitable to interact with the DB. Yet, it does not confirm how accurate the query results are concerning the user's initial request.
To investigate this critical qualitative aspect, additional measures from the relevant literature were considered. Evaluating qualitative aspects is essential to ensure that the responses provided by the agent are not hallucinated or fundamentally misinterpret the original request.

6.3.1 Exact Match

The first and most widely used qualitative metric in generative tasks is Exact Match. By constructing a dataset containing prompts and their corresponding ground truth queries, it is possible to directly compare the agent-generated query to one written manually by an experienced SQL developer. Although intuitive, this approach has clear limitations. Unlike other programming tasks or languages, given a specific request or prompt, SQL queries may not always be unique. This is due to the intrinsic problem's properties and the SQL language rules.

For instance, two queries may return identical results, and therefore both correct, but differing in the ordering of fields in the SELECT statement. The same issues can be encountered when the same fields are renamed using different aliases assigned by the agent.

For all these reasons, the results collected initially showed almost zero matches, despite many queries appearing to match the given requests visually. Therefore, to obtain a more precise evaluation, some pre-processing and post-processing refinements have been performed to normalize the points of divergence between the two queries, and try to compare codes built following the same convention:

- Standardizing field aliases

- Standardizing table aliases (e.g., enforcing aliases such as T1, T2)
- Cleaning special characters and indentation

Still, results remain partial, primarily due to variability in prompts. Extra fields in the generated query, though not necessarily incorrect, were often flagged as errors.

6.3.2 Result Matching

The second metric employed is Result Matching. Due to limitations identified in the first approach, an additional comparison based directly on query results could return a focus on different indicators. As the ultimate goal of the agent is data extraction, these results can be compared with those obtained from corresponding annotated ground truth queries in the evaluation database.

However, even this metric may not be completely reliable, especially when queries extract multiple varied rows or many fields, where field ordering remains relevant. Therefore, this measurement was evaluated on a subset of users' requests that produced only one single output. Specifically, the queries taken into consideration were the ones containing aggregation functions (such as sum or count) or those selecting only a few specific columns. This approach ensured that any mismatch between expected and actual results could be confidently identified.

6.3.3 Semantic Similarity

Since Result Matching is essentially an exact match performed on the results, it may not reveal all the positive or critical aspects of the task. To uncover additional insights, a Similarity metric was also employed. These metrics are commonly used to evaluate translations, textual responses from LLMs, and particularly coding generation tasks.

This approach allows evaluating how close or similar the query generated by the agent is concerning the target query written by the developer, even when an exact match metric has not given a positive outcome. In simpler terms, this metric estimates the distance between the agent's query and the ideal query.

However, it should be noted that similarity metrics are usually adopted to assess semantic similarity in text or translations, accommodating variations and misalignment caused by different word orders between predicted and target sentences or synonym usage. In coding tasks, a significant limitation arises if the generated query contains errors. For instance, a query might match 99% of the target but

include a critical syntactical error that prevents it from running on the database. In this scenario, even if the score is high, the query, which is not executable, is not capable of producing any result.

In this specific scenario, however, this limitation is balanced by the agent's capability to guarantee functional and syntactic correctness whenever a result is successfully produced. Combining these two types of information—the similarity measure and the assured execution correctness—further validates the quality of the results.

This metric remains useful for identifying queries that are really close to the target ones and may differ slightly, perhaps due to additional or missing fields in the `SELECT` clause or variations in the ordering of `WHERE` condition clauses.

6.3.4 LLM Evaluator

Lastly, in the fourth metric employed, the evaluation is delegated to a higher capability LLM. Specifically, the model used is the latest Claude 3.7. Given its impressive results, it has been chosen to evaluate the agent's results produced by smaller or less capable models. Claude 3.7 was chosen as a third-party evaluator to ensure objectivity, as it does not belong to the same organizations as the models powering the agent. This could hopefully reduce potential biases due to overlapping of training datasets or acquired knowledge.

Claude was prompted to perform a qualitative assessment of results based on:

- The provided database schema
- The user's initial request prompt
- The SQL query generated by the agent

By analyzing this information, Claude rated the correctness and relevance of the result on a confidence scale from 1 to 10—where 1 indicates a completely incorrect or mismatched query, and 10 represents a query that is fully syntactically and functionally correct.

More specifically, Claude was instructed to consider several criteria during evaluation:

- **Adherence to the initial prompt**
- **Contextual relevance**

- **Syntactic validity**
- **Correct schema linking**
- **Completeness and precision of the generated query**

Furthermore, in order to standardize and improve the reliability of evaluations, the prompt also included diverse examples demonstrating various error scenarios along with corresponding confidence scores given by a human. To fully leverage the reasoning and explanatory capabilities of LLMs, Claude was asked to provide not only a numeric confidence score but also a textual explanation of the reasoning performed to assign that score. The prompt choice has been made based on a series of tests performed on approximately 20 handcrafted examples covering various scenarios.

During these tests, results highlighted that including textual explanations along with the numeric scores enhanced both the precision of the evaluation score and the clarity of the LLM's judgment criteria.

6.4 Results

This section presents detailed results obtained by the evaluated models across the metrics defined in the previous sections.

6.4.1 Functional Correctness

The Functional Correctness metric measures how many queries produced by each model were syntactically correct and successfully executed against the database.

Model	Functional Correctness (%)
GPT 4o	95%
GPT 4o-mini	98%
Gemini 1.5-Flash	99%

Table 6.3: Results for Functional Correctness

The results on functional correctness, measured by the number of queries correctly executed on the DB, returned somewhat surprising results. GPT-4, in fact, which according to expectations was supposed to be the most powerful model, has a slightly lower percentage of correct executions compared to Gemini and GPT-4 Mini.

Analyzing the executions that did not succeed with GPT-4, it was noted that the process was completed correctly by the agent, but the model then returned an output message indicating that it was not successful or was uncertain about the result produced. This behavior had been explicitly requested in the prompt to prevent the agent from producing hallucinated or irrelevant results. In this context, GPT-4o-mini may have interpreted unclear points during execution and, instead of returning an answer regardless of its correctness, returned a message to indicate the failure of the extraction.

However, from a deeper analysis of the logs, it emerged that for the failed executions, the agent had correctly constructed the query and adapted it to the schema of the scenario. The agent powered by GPT-4 did not manage to complete the extraction correctly due to small syntactical problems or context-related issues, which could have been resolved by using the tools provided. This seems to highlight a greater difficulty in tool calling, likely related to a different context window, with, for example, Gemini being able to handle a longer context. This analysis was carried out empirically and could serve as a basis for future work to find a correlation between the context window and the agent's ability to perform tool or function calling.

6.4.2 Exact Match

Table 6.5 illustrates the Exact Match metric results, comparing generated queries directly against the ground-truth queries.

Model	**Exact Match (%)**
GPT 4o	15.5%
GPT 4o-mini	13.2%
Gemini 1.5-Flash	12.6%

Table 6.4: Results for Exact Match

The data reported for the exact match metric, as highlighted earlier, provides **insufficient** information to effectively assess the task at hand. The results, which are well below expectations, are indeed linked to the reasons already discussed in the section regarding exact match in the previous paragraphs. This type of evaluation is not capable of giving us the effectiveness of code generation. In fact, many syntactically correct queries that return the correct result differ in small parts and are therefore not considered correct.

6.4.3 Result Matching

To address the issues of the previous metrics, a subset of 120 prompt/query/result associations was considered, which produced a limited number of output columns. This subset mainly consists of queries that return aggregate functions, sums or counts, or a few specific fields. In this way, it was possible to more easily and effectively validate whether the result produced by the agent was consistent with the one produced by the target query.

The results obtained are listed below:

Model	**Result Match (%)**
GPT 4o	57.2%
GPT 4o-mini	68.3%
Gemini 1.5-Flash	68.0%

Table 6.5: Results for Result Match

The results in this case highlight once again how the performance of the three models is comparable. This indicates that the agent-based architecture works almost in the same way, producing similar results regardless of the model.

However, a lower result for GPT-4o stands out. This is partly justified by its correlation with the very first metric evaluated, functional correctness. It had been highlighted that GPT-4o had a higher number of queries that were not executed and therefore did not extract a result. The lack of a result essentially led to an outcome that was not in line with the other models, as highlighted by the comparison made with this metric.

Here again, the non-matching results were marked as incorrect mainly due to differences in the SELECT statement. GPT-4o-mini, for example, had a tendency to merge the first and last names of the users pulled into a COALESCE function, whereas the target query was written with the two pieces of data separate.

Also, though Gemini and GPT-4o-mini had the same performance, a review of the SELECT statements indicated that though GPT-4o and Gemini both return an average of about 1.64 columns per query, GPT-4o-mini returns about 1.95 fields per query. This tendency would imply more accuracy in GPT-4o-mini as it returns more data on average for the same set of requests. It also highlights the possibility that, in some cases, the divergence from the target results is due to retrieving more, redundant, or unexpected fields.

6.4.4 Similarity Metric

The Similarity metric assesses the semantic closeness of the queries generated by each model compared to target queries. Results are summarized in Table 6.6.

Model	Similarity Score (average)
GPT 4o	85%
GPT 4o-mini	91%
Gemini 1.5-Flash	89%

Table 6.6: Results for Similarity

Model	High Similarity Results >90%
GPT 4o	55.6%
GPT 4o-mini	66.3%
Gemini 1.5-Flash	60.9%

Table 6.7: Percentage of Results with High Semantic Similarity (>90%)

Validation through semantic similarity is calculated by generating the embeddings of the target and predicted queries and computing the cosine similarity. This makes it possible to determine how much the generated query deviates from the ground truth. Thus, it allows considering results that are not exactly identical to the target but, with minor changes—such as column order inversion or the inclusion or exclusion of a field—do not affect the final outcome.

In this second table, the percentage of queries with a high level of similarity to the target has also been reported to highlight how many queries generated by the agent are almost identical to the target. Once again, the results show that GPT 4o performs worse than the two smaller models, while GPT 4o-mini outperforms the others in both indicators.

Overall, despite these differences between models, the agent proves capable of closely approximating the target queries written by a developer.

6.4.5 LLM-Based Qualitative Evaluation

Table 6.8 summarizes the qualitative evaluations performed by Claude 3.7, which rated the generated queries on a confidence scale from 1 to 10, based on adherence to the initial prompt, contextual relevance, syntactic validity, schema linking accuracy, and completeness.

The evaluation conducted by another LLM with greater capabilities than the models used confirms a decidedly positive average. This type of evaluation also provides written comments that further clarify the assessment criteria applied.

Model	Average LLM Confidence Score	Std. Dev.
GPT 4o	7.87	2.24
GPT 4o-mini	8.42	1.12
Gemini 1.5-Flash	8.5	1.45

Table 6.8: Results from an LLM evaluator

Table 6.9: Evaluation of SQL Query Generation

ID	58
Prompt	Retrieve employees who are currently working in both Sales and Marketing departments and have an average salary above $70,000. Ensure only active employees are considered, and order results by highest average salary.

Generated SQL	`SELECT e.first_name, e.last_name, AVG(s.salary) AS average_salary`
	`FROM employees e`
	`JOIN dept_emp de ON e.emp_no = de.emp_no`
	`JOIN departments d ON de.dept_no = d.dept_no`
	`JOIN salaries s ON e.emp_no = s.emp_no`
	`WHERE d.dept_name IN ('Sales', 'Marketing')`
	`AND de.to_date = '9999-01-01'`
	`AND s.to_date = '9999-01-01'`
	`GROUP BY e.emp_no`
	`HAVING AVG(s.salary) > 70000`
	`ORDER BY average_salary DESC LIMIT 10;`

Confidence	4
Explanation	The query attempts to retrieve employees working in both Sales and Marketing departments, but it only checks if an employee is in either department, not both. The use of `IN` with a single join on `departments` will not ensure employees are in both departments simultaneously. Additionally, the query does not ensure that only active employees are considered in both departments, as it only checks the `to_date` for the current department, not for both. The query correctly filters for active salaries and orders by average salary, but it does not fully meet the user's intent of finding employees in both specified departments.

Table 6.10: Evaluation of SQL Query Generation

ID	47
Prompt	Count active employees per department and classify them as short-term (<5 years), mid-term (5-10 years), or long-term (>10 years) based on their tenure. Sort by total employees in descending order.

Generated SQL	`SELECT d.dept_name,`
	`SUM(CASE WHEN TIMESTAMPDIFF(YEAR, e.hire_date,`
	`CURDATE()) < 5 THEN 1 ELSE 0 END) AS short_term,`
	`SUM(CASE WHEN TIMESTAMPDIFF(YEAR, e.hire_date,`
	`CURDATE()) BETWEEN 5 AND 10 THEN 1 ELSE 0 END) AS mid_term,`
	`SUM(CASE WHEN TIMESTAMPDIFF(YEAR, e.hire_date,`
	`CURDATE()) > 10 THEN 1 ELSE 0 END) AS long_term,`
	`COUNT(*) AS total_employees`
	`FROM employees e`
	`JOIN dept_emp de ON e.emp_no = de.emp_no`
	`JOIN departments d ON de.dept_no = d.dept_no`
	`WHERE de.to_date = '9999-01-01'`
	`GROUP BY d.dept_name`
	`ORDER BY total_employees DESC LIMIT 10;`

Confidence	9
Explanation	The query is syntactically valid and correctly uses the schema to join the `employees`, `dept_emp`, and `departments` tables. It accurately calculates the tenure of employees using `TIMESTAMPDIFF` and classifies them into short-term, mid-term, and long-term categories. The query filters for active employees by checking `de.to_date = '9999-01-01'`, which aligns with the prompt's requirement to count active employees. It groups the results by department and orders them by total employees in descending order, which matches the user's request. The only minor issue is the `LIMIT 10` clause, which is not mentioned in the prompt, but it does not affect the correctness of the query logic. Therefore, the confidence is high.

6.4.6 Discussion

The system's robust ability to interact with databases and generate functionally correct queries without humans in the loop is reflected in how the AI agent has been evaluated. The findings show that the agent performs well on a variety of measures without needing to be intervened upon while producing queries.

One of the most interesting findings is that the agent provides outputs that closely match the target questions and they also produce semantically correct questions. The high semantic similarity scores, which determine how close the predicted queries are to the ideal outputs, substantiate this. In the same way, the LLM-based assessor guaranteed the quality of the output questions, offering a second qualitative evaluation that takes into account not just correctness but also contextual appropriateness and compliance.

A remarkable observation is that the performance of the agent is comparatively decoupled from the specific model that powers it. Even though smaller models with a larger context window show advantage in tool calling, the general trend is consistent across models. This suggests that the agentic architecture itself rather than the specific model has an important effect on generating high-quality output. The ability to leverage tool calling effectively is crucial in agent-based systems because it allows enhanced schema adaptation and query construction with increased accuracy even in the absence of full schema data initially.

The second crucial learning point is that the ReAct (Reasoning + Acting) paradigm enhances the reasoning capability of the model apart from token effectiveness. Instead of loading the entire database schema into the prompt, wasting excessive token usage—the agent dynamically interacts with the environment and reduces unnecessary token usage. Empirical results show that the implementation typically uses around 500 tokens for the input, which could double or triple during execution depending on the iterations and steps required. Nevertheless, this remains a cost-effective solution compared to static prompt-based methods that preload the entire schema beforehand.

Besides query execution, the agent's memory capabilities create another layer of efficiency. The ability to store context provides fast adaptation to feedback from the user, so that iterative refinement may be done without reloading the entire execution process. This feature significantly contributes to usability, rendering AI agents valuable helpers for developers and analysts working with databases.

Limitations

One notable limitation is prompt variability. In fact, the way queries are phrased can impact and influence the agent's performance. As much as the system may demonstrate robustness, variation of input forms could still produce inconsistencies in query generation. But it can also facilitate task execution, when for instance

the request is more explicit and the user add a greater number of details to the prompt.

In addition, the test environment was designed for business use cases with small to medium-sized databases. Therefore, examination of the tool's scalability requires further testing on larger and more complex datasets. Although initial indications are of scalability as a feasible goal, inference of the results to more comprehensive use cases remains an open topic for research.

Future Work

Several directions could be explored to further improve and validate the approach:

Using the current implementation as a baseline to test alternative architectures. For example, replace the ReAct paradigm and try integrating LLM models with native reasoning to conduct a comparative test against the current implementation in which reasoning has been inducted through prompt engineering techniques.

Scientifically confirming relative performance gaps between GPT-4o and GPT-4o-mini. Initial findings suggest that GPT-4o is slightly more challenged with tool usage and function calling interactions, but further scientific confirmation is needed to determine this trend.

An investigation of a multi-agent solution to the same problem. A multi-agent system might further improve performance and provide more efficient task decomposition and cooperation, as indicated findings in the literature have found.

Chapter 7

Conclusion

This thesis explored the technical and scientific literature on Large Language Model architectures, datasets, and benchmarks, emphasizing the use of AI agents to augment or replace programming activity. This review, coupled with an examination of the productivity effect of AI, has revealed an evident and widespread consensus on the potential of these technologies to simplify many elements of software development.

A central aspect of this work was to establish the aforementioned theoretical possibilities in a practical application. To do so, an SQL AI agent was developed as a case study. The goal was to test whether an LLM-based system could understand natural language and turn it into working SQL queries. The case study made it possible to move from theory to practice, offering a concrete opportunity to observe how an LLM-powered agent performs when applied to a real-world task.

The findings confirmed many of the initial expectations. The agent combined reasoning, conversational memory, and task decomposition effectively. It performed well in generating accurate SQL queries, demonstrating its ability to handle and elaborate complex input and produce reliable outputs without human assistance. This clearly shows that LLM-based agents are capable of effectively addressing complex tasks, automating routine tasks, and substantially minimizing constant human intervention needs. However, challenges, such as linguistic ambiguity and hallucinations could still potentially influence such systems' reliability and robustness.

Moreover, practical results have proven the importance of conducting more research on scalability and adaptability of such systems to other scenarios. Future works may be focused on identifying the most effective technologies and methods to improve reliability and ensure seamless integration into everyday software development practice.

The empirical research also underlined the evolving nature of the human role in relation to such sophisticated systems. The SQL agent acts independently upon

receiving an initial request, eliminating the need for constant human intervention. Nevertheless, human interaction does not disappear; instead, it evolves into a strategic function focused on supervision, validation, and refinement, primarily through conversational interactions following automated processing. This indicates a new form of human-AI collaboration where repetitive tasks are automated, while complex cognitive tasks remain distinctly human responsibilities.

We are currently experiencing a significant technological shift. AI is progressing from being used for individual tasks to managing increasingly complex systems and potentially entire workflows. Given this evolution, we can reasonably question whether a fundamentally new approach to programming is becoming possible. In such a rapidly evolving landscape, it is not easy to define the trajectory of this change. Based on the observations and results presented in this thesis, it is reasonable to state that the future of programming is increasingly moving toward a form of technological symbiosis—one in which humans and AI agents collaborate side by side, each contributing their specific strengths: AI takes on complexity and automation, while humans offer guidance, critical thinking, and oversight.

Bibliography

[1] *8 year old girl codes with Cursor AI*. URL: https://www.youtube.com/watch?v=o5uvDZ8srHA (cit. on p. 1).

[2] Gordon Burtch and Dokyun Lee. «The consequences of generative AI for online knowledge communities». In: *Scientific Reports* (May 2024) (cit. on p. 1).

[3] Pei Wang. «What Do You Mean by "AI"?» In: (June 2008) (cit. on p. 6).

[4] Warren S. McCulloch and Walter Pitts. «A Logical Calculus of the Ideas Immanent in Nervous Activity». In: *The Bulletin of Mathematical Biophysics* (1943) (cit. on p. 6).

[5] A. M. Turing. «Computing Machinery and Intelligence». In: *Mind* (1950) (cit. on pp. 7, 15).

[6] John McCarthy, Marvin L. Minsky, Nathaniel Rochester, and Claude E. Shannon. *A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence*. Aug. 1955. URL: https://www.dartmouth.edu/~ai50/AI1955.pdf (cit. on p. 7).

[7] Murray Campbell, A. Joseph Hoane Jr., and Feng-hsiung Hsu. «Deep Blue». In: *Artificial Intelligence* (2002) (cit. on p. 8).

[8] Robert M. Solow. «Technical Change and the Aggregate Production Function». In: *Review of Economics and Statistics* (1957) (cit. on p. 11).

[9] Paul M. Romer. «Endogenous Technological Change». In: *Journal of Political Economy* (1990) (cit. on p. 11).

[10] Philippe Aghion and Peter Howitt. «A Model of Growth Through Creative Destruction». In: *Econometrica* (Mar. 1992) (cit. on p. 11).

[11] Erik Brynjolfsson. «The Productivity Paradox of Information Technology». In: *Communications of the ACM* (Dec. 1993) (cit. on p. 11).

[12] Accenture. *Generating Growth: How Generative AI Can Power the UK's Reinvention*. Accenture, Oct. 2024. URL: https://www.accenture.com/nz-en/insights/gen-ai/generating-growth (cit. on p. 12).

[13] Tom Coshow. «Intelligent Agents in AI Really Can Work Alone. Here's How.» In: *Gartner* (Oct. 2024). URL: https://www.gartner.com/en/articles/intelligent-agent-in-ai (cit. on p. 12).

[14] Muhammad Riyaz Hossain. «The Integration of AI in Small Enterprises and Its Impact on Productivity and Innovation». In: *International Journal For Multidisciplinary Research* (Oct. 2024) (cit. on p. 12).

[15] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. «The Impact of AI on Developer Productivity: Evidence from GitHub Copilot». In: (2023) (cit. on p. 14).

[16] *Bain Company - DeepSeek: A Game Changer in AI Efficiency*. 2025. URL: https://www.bain.com/insights/deepseek-a-game-changer-in-ai-efficiency (cit. on p. 14).

[17] Muennighoff et al. «s1: Simple Test-Time Scaling». In: (2025) (cit. on p. 15).

[18] Warren Weaver. *Translation*. 1949 (cit. on p. 15).

[19] Brown et al. «A Statistical Approach to Machine Translation». In: *Computational Linguistics* (1990) (cit. on p. 16).

[20] Franz Josef Och and Hermann Ney. «A Systematic Comparison of Various Statistical Alignment Models». In: *Computational Linguistics* (2003) (cit. on p. 16).

[21] Franz Josef Och and Hermann Ney. «The Alignment Template Approach to Statistical Machine Translation». In: *Computational Linguistics* (2004) (cit. on p. 16).

[22] Mikolov et al. «Efficient Estimation of Word Representations in Vector Space». In: (2013) (cit. on p. 17).

[23] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. «Enriching Word Vectors with Subword Information». In: (2017) (cit. on p. 17).

[24] Pennington et al. «GloVe: Global Vectors for Word Representation». In: 2014 (cit. on p. 18).

[25] Hochreiter et al. «Long Short-Term Memory». In: *Neural Computation* (1997) (cit. on p. 18).

[26] Vaswani et al. «Attention Is All You Need». In: 2017 (cit. on pp. 19, 24).

[27] Bengio et al. «Learning Long-Term Dependencies with Gradient Descent is Difficult». In: () (cit. on p. 19).

[28] Bahdanau et al. «Neural Machine Translation by Jointly Learning to Align and Translate». In: 2014 (cit. on p. 19).

[29] Anthropic. *Claude 2: Anthropic's Next-Generation Language Model*. 2023. URL: https://www.anthropic.com/news/claude-2 (cit. on p. 22).

[30] Brown et al. «Language Models are Few-Shot Learners». In: (May 2020) (cit. on pp. 22, 27, 35, 36).

[31] Radford et al. «Language Models are Unsupervised Multitask Learners». In: *OpenAI Blog* (2019) (cit. on pp. 24, 26).

[32] Raffel et al. «Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer». In: 2019. URL: https://arxiv.org/abs/1910.10683 (cit. on p. 25).

[33] Devlin et al. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». In: 2019 (cit. on p. 25).

[34] Radford et al. «Improving Language Understanding by Generative Pre-Training». In: *OpenAI Blog* (2018). URL: https://openai.com/research/language-unsupervised (cit. on pp. 25, 26).

[35] Zhu et al. «Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books». In: (2015) (cit. on pp. 26, 29, 30).

[36] OpenAI. *GPT-4 Technical Report*. Tech. rep. 2023 (cit. on p. 27).

[37] OpenAI. *GPT-4o System Card*. Tech. rep. Aug. 2024 (cit. on p. 27).

[38] Anthropic. *Introducing Claude 3.5 Sonnet*. June 2024. URL: https://www.anthropic.com/news/claude-3-5-sonnet (cit. on p. 28).

[39] Anthropic. *Claude 3.7 Sonnet and Claude Code*. Feb. 2025. URL: https://www.anthropic.com/news/claude-3-7-sonnet (cit. on pp. 28, 43).

[40] *The Llama 3 Herd of Models*. Tech. rep. Meta AI Research, 2024 (cit. on p. 28).

[41] Gemini Team, Google. «Gemini: A Family of Highly Capable Multimodal Models». In: *arXiv* (2024) (cit. on p. 28).

[42] Google DeepMind. *Google Gemini 2.0 AI Update*. Dec. 2024. URL: https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024/#gemini-2-0-flash (cit. on p. 28).

[43] Common Crawl. *Common Crawl Corpus*. URL: https://commoncrawl.org (cit. on p. 29).

[44] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. *WebText Dataset*. 2019 (cit. on p. 29).

[45] Kaplan et al. «Scaling Laws for Neural Language Models». In: *arXiv preprint arXiv:2001.08361* (2020) (cit. on p. 30).

[46] Hoffmann et al. «Training Compute-Optimal Large Language Models». In: *arXiv preprint arXiv:2203.15556* (2022) (cit. on p. 30).

[47] Blodgett et al. «A Taxonomy of Biases in Natural Language Processing». In: *arXiv preprint arXiv:2005.00614* (2020) (cit. on p. 30).

[48] Singh et al. «Evaluation Data Contamination in LLMs: How Do We Measure It and (When) Does It Matter?» In: (2024) (cit. on p. 31).

[49] Chunyuan Deng and et al. «Investigating Data Contamination in Modern Benchmarks for Large Language Models». In: (2024) (cit. on p. 31).

[50] Cheng Xu. «Benchmark Data Contamination of Large Language Models: A Survey». In: *arXiv preprint arXiv:2406.12345* (2024) (cit. on p. 31).

[51] Li et al. «An Open Source Data Contamination Report for Large Language Models». In: *arXiv preprint arXiv:2310.17845* (2023) (cit. on p. 31).

[52] Dubey et al. «The Llama 3 Herd of Models». In: (2024) (cit. on p. 31).

[53] Wooldridge Jennings. «Intelligent Agents: Theory and Practice». In: (1995) (cit. on p. 31).

[54] Lilian Weng. «LLM Powered Autonomous Agents». In: (2023). URL: `https://lilianweng.github.io/posts/2023-06-23-llm-autonomous-agents/` (cit. on p. 32).

[55] Noah Shinn. «Reflexion: Language Agents with Verbal Reinforcement Learning». In: (2023) (cit. on p. 33).

[56] Julia Wiesinger, Patrick Marlow, and Vladimir Vuskovic. *Agents*. Google AI Whitepaper. Sept. 2024 (cit. on p. 33).

[57] Lewis et al. «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks». In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020 (cit. on pp. 34, 37).

[58] IBM. *What is a Multiagent System?* 2024. URL: `https://www.ibm.com/think/topics/multiagent-system` (cit. on p. 35).

[59] Pranab Sahoo. «A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications». In: *arXiv preprint arXiv:2401.12345* (2024) (cit. on p. 35).

[60] Wei et al. «Chain-of-Thought Prompting Elicits Reasoning in Large Language Models». In: *arXiv preprint arXiv:2201.11903* (2022). URL: `https://arxiv.org/abs/2201.11903` (cit. on pp. 36, 37, 50).

[61] Wang and et al. «Self-Consistency Improves Chain of Thought Reasoning in Language Models». In: *arXiv preprint arXiv:2203.11171* (2022) (cit. on p. 36).

[62] Yao et al. «Tree of Thoughts: Deliberate Problem Solving with Large Language Models». In: *arXiv preprint arXiv:2305.10601* (2023). URL: https://arxiv.org/abs/2305.10601 (cit. on p. 36).

[63] Li et al. «Structured Chain-of-Thought Prompting for Code Generation». In: *arXiv preprint arXiv:2301.XXXXX* (2023). URL: https://arxiv.org/abs/2301.XXXXX (cit. on pp. 37, 50).

[64] Dina Genkina. «AI Prompt Engineering is Dead: Long Live AI Prompt Engineering». In: *IEEE Spectrum* (2024) (cit. on p. 37).

[65] Google Developers. *AI Image Solutions*. 2025. URL: https://developers.google.com/solutions/ai-images (cit. on p. 37).

[66] Sun et al. «AI Hallucination: Towards a Comprehensive Classification of Distorted Information in Artificial Intelligence-Generated Content». In: (2024). URL: https://www.nature.com/articles/s41599-024-03811-x (cit. on p. 37).

[67] Chen et al. «Evaluating Large Language Models Trained on Code». In: *arXiv preprint arXiv:2107.03374* (2021) (cit. on pp. 38, 41, 43).

[68] Hendrycks et al. «Measuring Coding Challenge Competence with APPS». In: *arXiv preprint arXiv:2105.09938* (2021) (cit. on pp. 38, 41).

[69] Lu et al. «CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation». In: *arXiv preprint arXiv:2102.04664* (2021). URL: https://arxiv.org/abs/2102.04664 (cit. on pp. 40, 43).

[70] Puri et al. *Project CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Programming Tasks*. IBM Research. 2021 (cit. on p. 40).

[71] Lai et al. «DS-1000: A Natural and Reliable Benchmark for Data Science Code Generation». In: (2022) (cit. on p. 41).

[72] Lajko et al. «MultiPL-E: A Scalable Benchmark for Evaluating Multilingual Programming Language Understanding». In: *arXiv preprint arXiv:2212.10450* (2022) (cit. on p. 41).

[73] Austin et al. «Program Synthesis with Large Language Models». In: (2021) (cit. on p. 41).

[74] Niklas Muennighoff. «OctoPack: Instruction Tuning Code Large Language Models». In: *arXiv preprint arXiv:2308.07469* (2023) (cit. on p. 42).

[75] Li et al. «StarCoder: May the Source Be with You!» In: (2023) (cit. on p. 42).

[76] Clement et al. «PyMT5: Multi-Tasking and Multi-Programming Languages with a Unified Text-to-Text Transformer». In: (2020) (cit. on p. 43).

[77] Ren et al. «CodeBLEU: a Method for Automatic Evaluation of Code Synthesis». In: *arXiv preprint arXiv:2009.10297* (2020) (cit. on p. 43).

[78] Chowdhery et al. «PaLM: Scaling Language Modeling with Pathways». In: *arXiv preprint arXiv:2204.02311* (2022) (cit. on p. 43).

[79] Li et al. «Competition-Level Code Generation with AlphaCode». In: *arXiv preprint arXiv:2203.07814* (2022) (cit. on p. 43).

[80] Fried et al. «InCoder: A Generative Model for Code Infilling and Synthesis». In: 2023 (cit. on p. 43).

[81] Touvron et al. «LLaMA: Open and Efficient Foundation Language Models». In: *arXiv preprint arXiv:2302.13971* (2023) (cit. on p. 43).

[82] Touvron et al. «Llama 2: Open Foundation and Fine-Tuned Chat Models». In: *arXiv preprint arXiv:2307.09288* (2023) (cit. on p. 43).

[83] OpenAI. *Learning to Reason with LLMs*. May 2024. URL: `https://openai.com/index/learning-to-reason-with-llms/` (cit. on p. 44).

[84] Fang et al. «Large Language Models for Code Analysis: Do LLMs Really Do Their Job?» In: (Mar. 2024) (cit. on p. 44).

[85] Alibaba Group Qwen Team. «CODEELO: Benchmarking Competition-level Code Generation of LLMs with Human-comparable Elo Ratings». In: (2025) (cit. on p. 44).

[86] Zheng et al. «A Survey of Large Language Models for Code: Evolution, Benchmarking, and Future Trends». In: (2024) (cit. on p. 44).

[87] Hou et al. «Comparing Large Language Models and Human Programmers for Generating Programming Code». In: *Advanced Science* (2024) (cit. on p. 44).

[88] Jay Peters. «OpenAI lays out plans for GPT-5». In: *The Verge* (2025). URL: `https://www.theverge.com/news/611365/openai-gpt-4-5-roadmap-sam-altman-orion` (cit. on p. 45).

[89] Pourreza Rafiei. «DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction». In: 2023 (cit. on p. 48).

[90] Tai et al. «Exploring Chain of Thought Style Prompting for Text-to-SQL». In: 2023 (cit. on p. 48).

[91] Gao et al. «Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation». In: (2023) (cit. on pp. 48, 50).

[92] An et al. «Skill-Based Few-Shot Selection for In-Context Learning». In: (2023) (cit. on p. 48).

[93] Rajkumar et al. «Evaluating the Text-to-SQL Capabilities of Large Language Models». In: (2022) (cit. on p. 49).

[94] Liu et al. «Benchmarking the Text-to-SQL Capability of Large Language Models: A Comprehensive Evaluation». In: (2024) (cit. on p. 49).

[95] Zhu et al. «Large Language Model Enhanced Text-to-SQL Generation: A Survey». In: (2024) (cit. on p. 49).

[96] Sarker et al. «Enhancing LLM Fine-tuning for Text-to-SQLs by SQL Quality Measurement». In: (2024) (cit. on p. 49).

[97] Qin et al. «A Survey on Text-to-SQL Parsing: Concepts, Methods, and Future Directions». In: (2022) (cit. on p. 49).

[98] IBM. *Prevent Prompt Injection*. 2025. URL: `https://www.ibm.com/think/insights/prevent-prompt-injection` (cit. on p. 49).

[99] Zhang et al. «Instruction Tuning for Large Language Models: A Survey». In: *arXiv preprint arXiv:2308.10792* (2023) (cit. on p. 50).

[100] Li et al. «Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Text-to-SQL Task». In: 2018 (cit. on p. 50).

[101] Li et al. «Can LLM Already Serve as A Database Interface? A Big Bench for Large-Scale Database Grounded Text-to-SQLs». In: 2023 (cit. on p. 51).

[102] Zhong et al. «Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning». In: 2018 (cit. on p. 51).

[103] Chen et al. «BEAVER: An Enterprise Benchmark for Text-to-SQL». In: 2023 (cit. on p. 51).

[104] Papineni et al. «BLEU: a Method for Automatic Evaluation of Machine Translation». In: 2002 (cit. on p. 52).

[105] Shuo Ren et al. *CodeBLEU: A Method for Automatic Evaluation of Code Synthesis*. 2020 (cit. on p. 52).

[106] He et al. «DeBERTa: Decoding-enhanced BERT with Disentangled Attention». In: *arXiv preprint arXiv:2006.03654* (2020) (cit. on p. 56).

[107] Yao et al. «ReAct: Synergizing Reasoning and Acting in Language Models». In: (2022) (cit. on p. 56).

[108] Zijin et al. «Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL». In: (2024) (cit. on p. 60).