VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
INSTITUTE OF COMPUTER SCIENCE
DEPARTMENT OF COMPUTATIONAL AND DATA MODELING

Master Thesis

Investigation of Process Automation with Large Language Models

Done by:
Manvydas Sokolovas signature

Supervisor:
dr. Linas Petkevičius

Vilnius
2024

Contents

Glossary of Terms

- **NLP (Natural Language Processing):** A branch of artificial intelligence focused on enabling computers to understand, interpret, and generate human language.
- **LLM (Large Language Model):** A type of machine learning model, often based on neural networks, designed to understand, generate, and translate human language at a large scale.
- **GPT (Generative Pre-trained Transformer):** A type of LLM model developed by OpenAI pre-trained on large data sets of unlabelled text and able to generate novel human-like content.
- **ChatGPT:** A specialized version of the GPT model by OpenAI, optimized for generating human-like conversational response.
- **RNN (Recurrent Neural Network):** A class of neural networks where connections between nodes form a directed graph along a temporal sequence, allowing it to exhibit temporal dynamic behavior.
- **LSTM (Long Short-Term Memory):** Advanced RNNs for learning long-term data sequences.
- **CLI (Command Line Interface):** Text-based interface for managing files and programs via commands.
- **UI (User Interface):** The interface for user interactions with computers and software.
- **Prompt:** User-provided input or instruction that triggers the system to process and generate a response or output.
- **Token:** In NLP it is a unit of text, such as a word, subword or character, treated as a single unit by model.

Abstract

This thesis investigates the role of Large Language Models (LLMs), specifically AutoGPT, in the automation of software development for data analysis, focusing on the impact of integrating functional and non-functional requirements on LLM decision quality. This study evaluates AutoGPT's ability to write Python scripts for data processing and to create R Shiny dashboards across three distinct datasets, using prompts of varying complexity. The experiment progresses from simple natural language prompts to structured ones, finally to prompts that integrate both functional and non-functional requirements. This progression enables an in-depth assessment of AutoGPT's efficiency, accuracy and challenges. Key findings show that AutoGPT can autonomously generate Python scripts and R Shiny dashboards. The study reveals that prompt complexity enhances AutoGPT's output quality and efficiency, although challenges such as the reproducibility of generated codes and sensitivity to the structure of prompts are observed. Concluding, the thesis underscores the potential and current limitations of AutoGPT in software development for data analysis. It suggests a strategic approach to prompt construction and indicates that while AutoGPT is promising, it does not yet match the capability of a skilled human data analyst.

Keywords: Large Language Models, AI agents, AutoGPT, Data Analysis, Software Development Automation

Santrauka

Didžiųjų kalbos modelių panaudojimo procesų automatizavimui tyrimas

Šiame darbe yra tiriamos didžiųjų kalbos modelių (LLM), konkrečiai AutoGPT, galimybės automatizuoti programinės įrangos kūrimą duomenų analizei. Esant sparčiai dirbtinio intelekto ir mašininio mokymo plėtrai, LLM tapo svarbiu elementu duomenų analitikoje. Tačiau jų galimybės automatizuoti programinės įrangos kūrimą iki šiol nebuvo pakankamai ištirtos. Šis tyrimas siekia užpildyti šią spragą, įvertinant AutoGPT efektyvumą rašant Python kodą duomenų apdorojimui ir išgavimui bei tuos duomenis atvaizduojant R Shiny programos pagalba.

Naudojant tris įvairius duomenų rinkinius, tyrimas metodologiškai įvertina AutoGPT veikimą naudojant įvairaus sudėtingumo užklausas. Tyrimas pradedamas nuo paprastų kasdienės kalbos užklausų, vėliau pereinama prie labiau strukturizuotų ir baigiama užklausomis, kurios integruoja tiek funkcinius, tiek nefunkcinius reikalavimus. Toks planas leidžia išsamiai įvertinti AutoGPT efektyvumą, tikslumą ir iššūkius įvairiose situacijose.

Išvadose atskleidžiama, kad AutoGPT gali generuoti Python kodą duomenų apdorojimui ir kurti R Shiny ataskaitas. Taip pat, tam tikriems duomenų rinkiniams AutoGPT pateikė duomenų analize paremtas įžvalgas bei dokumentacijas. Buvo pastebėta, kad AutoGPT rezultatų kokybė ir efektyvumas gerokai pagerėja didinant užklausų sudėtingumą ir specifiškumą, ypač į užklausą įtraukiant funkcinius ir nefunkcinius reikalavimus. Tačiau taip pat pastebėti iššūkiai bandant atkartoti LLM sugeneruotus išeities kodus bei didelė AutoGPT rezultatų priklausomybė nuo užklausos struktūros.

Šis darbas prisideda prie geresnio supratimo apie procesų automatizavimą naudojant LLM, ypač programinės įrangos kūrimo duomenų analizei srityje. Jis siūlo strukturizuotą planą užklausų kūrimui ir pabrėžia AutoGPT galimybes bei apribojimus praktinėse taikymo srityse. Tyrimas baigiamas išvada, kad nors dirbtinio intelekto agentai, tokie kaip AutoGPT, rodo potencialą automatizuojant programinės įrangos kūrimą, tačiau jų gebėjimai dar neprilygsta kvalifikuoto duomenų analitiko gebėjimams.

Raktiniai žodžiai: Didieji kalbos modeliai, DI agentai, AutoGPT, Duomenų analizė, Programinės įrangos kūrimo automatizavimas

Introduction

In the era of global digitalization, we observe an undeniable growth of data tracking mechanisms. As individuals we engage in a multitude of data generating activities, every click, transaction and interaction contributes to the generation of data points. The cumulative effect of these actions has resulted in the rapid increase in the volume of data generated [47]. The exponential growth of data in the digital age has introduced both challenges and opportunities for researchers. This provides perfect environment for the rapid evolution of data exploration techniques ranging from basic data-driven insights applicable to the fields like business [30], healthcare [4], education [26] and our everyday life [18], to the more sophisticated methods such as deep learning, artificial intelligence (AI) and large language models (LLM) for the more advanced applications in various domains.

Recent progress in natural language processing (NLP) led to the fast development of powerful large language models (LLMs) like ChatGPT. ChatGPT, in particular, has gained significant interest due to its impressive natural language processing capabilities and its accessibility to the general public. People have been actively investigating the potential uses of LLMs to enhance efficiency, accuracy and decision-making in a wide array of applications, from text generation to translation, summarization and question-answering [11, 2]. Widespread adoption of ChatGPT has ushered in a new era of automation, changing the way we approach complex processes in various fields and software development is no exception [52, 42, 43].

As software development become increasingly complex and demanding, our research strives to bridge the gap between cutting-edge natural language processing capabilities and the increasing demand of software engineering. The purpose of this master thesis is to analyze the potential of large language models in the context of automated software development. To achieve our objectives, we have outlined a comprehensive set of tasks. First, we perform a scientific literature review, examining a wealth of knowledge from previous research in the field of LLM and its applications in software development. Secondly, we collect 3 different test data sets and create test protocols. This not only ensures the reproducibility and measurability of our experiments but also creates a benchmark for evaluating the performance of different LLM prompting approaches. Through empirical studies, we further investigate how both functional and non-functional requirements can enhance the quality of automated software code generation using LLM. Finally, we provide actionable recommendations and insights for the creation of automated software development using LLM.

The aim of this thesis: To make a comparative and exploratory analysis in the context of automated software development by identifying the impact of functional and non-functional requirements on LLM decision quality.

Objectives:

1. **Literature Review:** Conduct a scientific review of the literature and identify relevant possible solutions.

2. **Collect Test Datasets:** Collect and process three distinct test datasets. Design and establish robust test protocols, ensuring the reproducibility, measurability and benchmarking capabilities of our experiments.

3. **Empirical Study:** Conduct an empirical study of how various functional/non-functional requirements can improve automated software code generation.

4. **Recommendations and Insights:** Provide actionable recommendations and conclusions for the creation of automated software code. Draw conclusions that could guide future research and practical applications in this field.

1 Related Work

This section reviews the literature and methods of software development automation using large language models. It begins with the evolution of large language models. Tracing the development from simple models to the most advanced today. Then it continues with process automation and its significance. Discussing existing automation applications like content generation, code writing and other. The chapter then continues about tools, techniques and challenges of data analysis today, followed by analysis of studies of using LLM for data analysis. Finally, the chapter ends with a summary and findings of literature analyzed.

1.1 Evolution of Large Language Models

Although the introduction of ChatGPT to public usage has brought significant attention to large language models in recent years, the evolution of natural language processing models has a long history dating back to the mid-1960s. with the introduction of the first language model - ELIZA, created by Joseph Weizenbaum. This was when the first language model - ELIZA, created by Joseph Weizenbaum [51]. It was a relatively simple program that used pattern recognition to simulate human-like conversation and generate a response from a set of pre-defined rules. While ELIZA was not "large" by today's standards, it was as an early example of human-computer interaction using natural language processing and it marked the beginning of research into natural language processing (NLP) and the development of more sophisticated LLMs.

As computational capabilities increased [12], the field of natural language processing (NLP) evolved with the emergence of statistical language models [31] and the early use of machine learning algorithms in the 1980s and 1990s. The transition from rule-based systems to probabilistic models, like n-gram [25] or hidden Markov models (HMMs) [37], marked a significant advancement in the machine's ability to predict and generate language.

Further NLP improvements emerged with the adoption of neural network based approaches, Recurrent Neural Networks (RNNs) and particularly Long Short-Term Memory (LSTM) architecture became transformative with their ability to handle sequential data and long-term dependencies, fundamental for tasks like speech recognition, language modeling or text generation [10, 32, 21]. These advancements, while monumental, were not without limitations. Issues such as vanishing gradients in standard RNNs [5] or lack of parallel computing were significant limitations, sparking further research and development into more robust architectures. The introduction of the attention mechanism in neural networks presented a solution to these problems, laying the groundwork for the next breakthrough in NLP – the Transformer model [50]. This architecture leverages self-attention to weigh the influence of different parts of the input data, which makes application of parallel processing easier.

The transformers model led to further reasearch of chatbots and NLP, which enabled the development of Bidirectional Encoder Representations from Transformers (BERT) [9] by researchers at Google capable to capture deep bidirectional contexts by pre-training on a large amount of unlabeled text and fine-tuning for various tasks with just one additional output layer. In parallel, the development of Generative Pre-trained Transformer (GPT) [38, 39, 8, 35] models by OpenAI has shown a leap in generative capabilities. These models, trained on large amount of diverse texts from the internet, are able to perform complex NLP tasks. With each iteration, from GPT-1 to the latest GPT-4, they showed significant improvements in machine translation, question answering, summarization and increased general application scope. The most advanced models have demonstrated unprecedented success in language comprehension and generation, opening new ways to automate complex processes, from writing and content creation to coding and data analysis using large language models. These developments have given rise to intelligent agents like AutoGPT [15], MetaGPT [22] and BabyAGI [34], which using OpenAI's GPT API and other tools are designed to autonomously carry out a diverse array of tasks.

1.2 Process Automation and Its Significance

Process automation, defined as the technology-enabled automation of various processes that reduces human intervention in them, has evolved significantly over the past several decades. It has fundamentally changed how work is completed in some industries, shifting from labor-intensive practices to efficient, technology-driven operations. Historically, process automation found its roots in the manufacturing, particularly in textile manufacturing [49], sector with the introduction of machinery to automate manual tasks. Since then, it has evolved to support a variety of applications across different sectors and this evolution shows a transition from manual, often error-prone processes to streamlined and efficient automated systems across industries such as manufacturing, telecommunications, banking and agriculture. The significance of this transformation lies in substantial gains in productivity, accuracy and efficiency it brings, empowering businesses to scale operations and innovate rapidly.

Development of Artificial Intelligence (AI) and Machine Learning (ML) has further pushed process automation into a new era, where automation systems are not just rule-based processes, but are able to learn and adapt, expanding their scope and scale to the new areas of application. In today's manufacturing landscape, robotics has become a key element executing tasks with a level of precision and stamina that far exceeds human limits. Apart from physical tasks, AI and ML have transformed the landscape of data processing and decision making, transitioning towards adaptive and continuously learning systems. These systems are good by learning from new data, identifying patterns and making intelligent decisions with little to no human input. There are many real-world applications, including recommendation systems in various industries, such as AI-driven content management in streaming services that improves viewing recommendations [48] or personalized shopping experiences in e-commerce which improve product discovery [46]. Also intelligent document processing which automates relevant data extraction from various types of documents [13]. These applications not only demonstrate a successful integration of AI and automation but also highlight significantly reduced time and costs for operations while simultaneously improving the quality of services.

In the field of content generation, automation is making significant advancements. This progress can be seen in a variety of applications, from simple text auto-completion to the writing of narrative content or chatbots [19, 29]. Here, the rapid processing and generative capabilities of AI have proved to be crucial. In software development, process automation has taken a more sophisticated turn with the emergence of tools like GitHub Copilot [14]. Leveraging the capabilities of Large Language Models (LLMs) to assist in code writing and debugging, significantly reducing the time and effort required for developing and maintaining software. Additionally, in the area of data analysis, LLMs could help with often labor-intensive process of data preparation, enabling more efficient data cleaning, feature engineering and train data generation [53, 7]. This not only speeds up the analysis process but also enhances the accuracy and reliability of the insights generated, marking a pivotal shift in how data-driven decisions are made.

In conclusion, while process automation, particularly through advancements in AI and ML, has revolutionized various industries by enhancing efficiency, accuracy and productivity, it is not without limitations. Large Language Models (LLMs), despite their sophisticated capabilities, often struggle with understanding context and create hallucination results [24, 3]. This shows the importance of human oversight, especially in critical decision-making processes. The integration of AI in automation has opened new horizons for innovation and operational excellence, but it also requires a balanced approach, where human expertise and ethical considerations are necessary to the successful implementation and evolution of these technologies. This balance is crucial to getting the full potential of process automation while mitigating its inherent challenges.

1.3 Modern Data Analytics: Tools, Techniques, Challenges and the Role of the LLM

The landscape of data analytics is constantly evolving, primarily due to increasing importance of data in decision-making across multiple industries. From healthcare to finance, data-driven insights have become crucial for strategic planning and operational efficiency [40, 45, 36]. This increased reliance on data analytics has created a strong need for skilled data professionals capable of extracting meaningful insights from complex datasets that are integral in guiding everything from business strategies to scientific research. However, actionable insights extraction from raw data not only requires comprehensive domain knowledge and advanced analytical techniques, but also data visualization skills and often some software development skills in programming languages like Python or R [6, 55, 1]. These tools, known for their powerful data manipulation and analytical capabilities, often require extensive training and expertise to use their full potential. In response to this growing demand, educational institutions and online learning platforms are rapidly expanding their course offerings in data science and analytics, equipping a new generation of professionals with the tools and knowledge needed to harness the power of data in our increasingly data-driven world [1].

However, the field of data analytics is not without its challenges. Some issues include concerns over data quality, privacy and the management of large and complex datasets [20, 41, 17]. Data quality issues can arise from various sources, such as incomplete data, inconsistent formats or errors in data collection and entry. Privacy concerns have also become increasingly significant, because data breaches and misuse can lead to severe consequences. Additionally, handling of large datasets requires specialized tools and techniques to process and analyze it effectively. Such issues not only slow down the analytical process but also can affect the quality of insights derived. These challenges highlight the need for advanced data analytics tools and systems that can assist data analysts in their work, making their processes not only faster but also more accurate and efficient. Solutions that can automate parts of the data cleaning, processing and analysis are in high demand, as they can significantly reduce the time and effort required to derive actionable insights or even replace some manual tasks.

Additionally, as technology advances, the role of artificial intelligence and machine learning in data analytics is expanding, offering new possibilities for deeper insights, predictive analytics and automated decision-making, thereby changing how data professionals work [16]. Large Language Models (LLMs) have begun to play a crucial role in data analysis. Case studies and recent research show that LLMs can help with data processing, interpretation, data quality checks, providing descriptive statistics, well interpreting unstructured data or even write code for data analysis [53, 54, 28, 16, 33]. In this context, the integration of Large Language Models (LLMs) into data analytics presents numerous opportunities. LLMs, with their advanced natural language processing capabilities, are beginning to play a role in various aspects of data analysis, from assisting in data cleaning and preprocessing to helping in the planning and initial stages of data analysis projects. They offer the potential to automate some of the more laborious parts of the data analyst's workflow, thereby speeding up the process, leaving more time for more valuable tasks and possibly uncovering insights that might be missed by traditional methods.

2 Methodology

This section covers the methodology used to investigate the integration of large language models (LLMs) in software development automation. It begins with an exploration of GPT models, briefly explaining their main concepts and how they work. Then it continues with the role and functionality of AI agents like AutoGPT, followed by an examination of effective prompt construction techniques for LLMs, which are essential for effective LLM utilization. Subsequently, it discusses the criteria for selecting appropriate datasets for this thesis. The chapter then continues discussing the formal requirements necessary for software development projects, delving into functional and non-functional requirements. It concludes with a detailed strategy for evaluating LLM performance in software development automation practical applications.

2.1 Generative Pre-trained Transformers (GPT)

This subsection discusses the essential concepts and techniques related to Generative Pre-trained Transformers (GPT), including the architecture of Transformers and their application in GPT models, as well as input tokenization. Understanding these foundational elements is crucial for our investigation into process automation using LLMs.

2.1.1 Transformers

Among the most known LLMs are the Generative Pre-trained Transformer (GPT) models, specifically GPT-3 and GPT-4, developed by OpenAI. These models represent the cutting edge in language processing and generation technology. GPT models learn to predict and generate language by being trained on a large corpus of text data, allowing them to respond to a wide range of prompts with high accuracy. Their versatility in handling various language tasks, from translation to code generation, makes them particularly relevant to the domain of software development automation.

GPT models use a transformer architecture [38] that handles sequential data well, making them very efficient for natural language based tasks. They operate using a mechanism of attention, allowing the model to focus on different parts of the input text to generate contextually relevant outputs.

Unlike standard transformers that use both encoders and decoders (Figure 1), GPT models use only decoders. These decoders are structured as a series of stacked layers, each consisting of self-attention mechanism and feed-forward neural network layer. This architecture is designed for text generation tasks, showing strong performance in predicting the next tokens in a sequence by effectively utilizing the context provided by the preceding tokens. This design choice makes GPT models good at processing and generating language, making them particularly effective for a wide range of generative language tasks.

Transformers revolutionized NLP with their unique architecture, centralizing around the concepts of attention and self-attention mechanisms. The key to their success lies in the ability to process entire sequences of data simultaneously, unlike traditional models that process data sequentially.

- **Positional Encoding.** Since Transformers don't have a recurrence mechanism like RNNs, positional encodings are used to give the model a sense of word order.

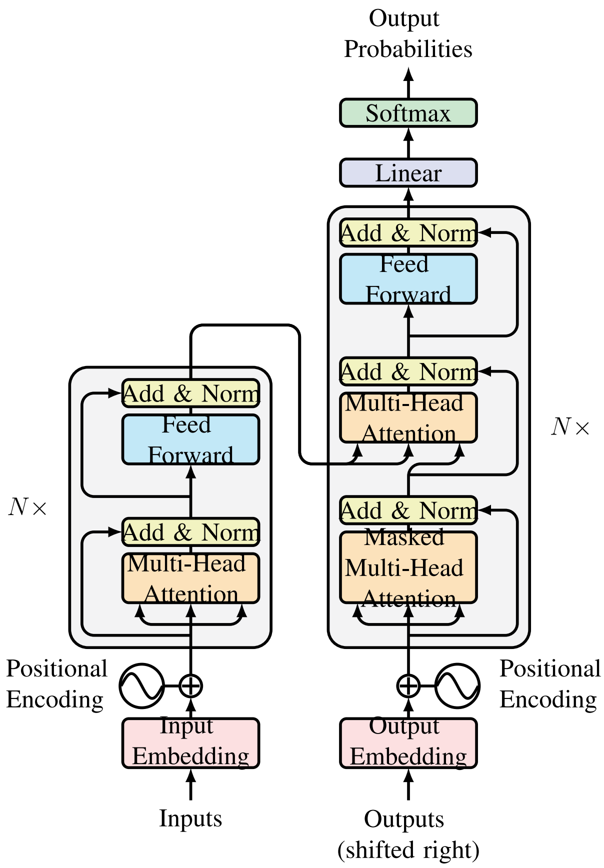


Figure 1. The Transformer architecture [50]. Encoder on the left and Decoder on the right.

- **Attention.** This mechanism allows the model to focus on different parts of the input sequence when performing a task, like humans pay attention to specific aspects of what they're reading or listening to.

- **Self-Attention.** A specialized form of attention, self-attention associate each element in the input sequence to every other element, giving a comprehensive understanding of the entire sequence.

- **Masked Attention.** Masked attention is a technique where certain positions in the input sequence are masked or hidden from the self-attention mechanism. This means that a model cannot see the masked positions, in this case it cannot attend future words in the sequence and it only has access to itself and the words before it.

- **Feed-Forward Network.** Feed-forward network is responsible for processing the information from the attention mechanism, applying additional transformations to the data. Each feed-forward network typically consists of two linear transformations with a non-linear activation function in between, allowing the network to learn complex patterns within the data.

GPT operates on an autoregressive principle, where it predicts the probability of each subsequent token (word or sub-word) based on the preceding sequence of tokens. During text generation, for each token prediction, GPT uses a softmax function to generate a probability distribution over its vocabulary, choosing the most likely next token.

2.1.2 Tokenization

The initial stage of text processing in GPT is tokenization, a critical step that segments the input text into manageable units, called tokens [31]. These tokens enable the model to understand and predict the statistical relationships in the sequence of words. This process is essential as neural networks are designed to interpret numerical data rather than raw text.

GPT models utilize a hybrid approach to tokenization, incorporating both word-level and subword-level strategies. This combination allows the models to capture linguistic nuances while effectively managing the vocabulary size. As a result, in GPT models, tokens may not align perfectly with word boundaries, they can include trailing spaces or even parts of words (Figure 2).

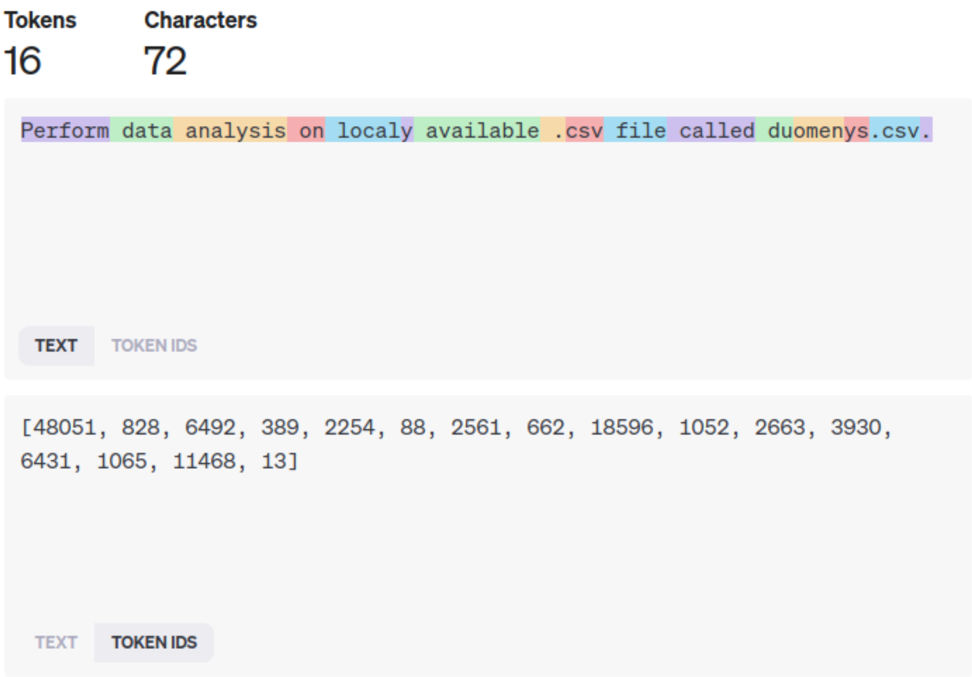


Figure 2. GPT-3.5 and GPT-4 Text Tokenization Example

To provide a sense of scale, a single token in English typically corresponds to about 4 characters, 100 tokens approximate 75 words and 1-2 sentences are approximately 30 tokens. This quantification helps in understanding the granularity and scale of tokenization in practice.

It is also important to mention that OpenAI's GPT API automatically convert text to tokens. Thus user does not have to do it manually.

Understanding the infrastructure of transformers and their adaptation in GPT models, along with the process of input tokenization, is crucial for our exploration of process automation using Large Language Models (LLMs).

2.2 AI Agents (AutoGPT)

This subsection explores the principal concepts of AI Agents, with a particular focus on AutoGPT [15]. It discusses their capabilities in autonomously executing tasks such as data analysis and software development by utilizing the advanced functionalities of Large Language Models (LLMs). The subsection also covers the main limitations, including cost considerations and the tendency to generate inaccurate or "hallucinated" content, which may affect their reliability and practical application. The subsection aims to provide a balanced view of AutoGPT's capabilities and challenges in the context of AI-driven process automation.

Currently, people interaction with AI involves a standard procedure: a user provides a prompt and the AI generates a response based on that input. This process requires a new prompt for each desired output. Thus, it requires continuous human intervention. However, AI agents function differently as they are designed for autonomous operation and decision-making. The innovation here is the shift from a passive to an active role in AI interaction. The user's role is primarily in setting objectives, which could range from conducting a market research to creating a personalized fitness program. Once a goal is created, AI agents autonomously formulate and execute their plan, dynamically adjusting their approach based on environmental feedback and internal assessment. This self-prompting capability allows AI agents to continuously refine their strategies to efficiently meet their goals. Unlike traditional automation, which relies on predefined triggers and actions, AI agents can adapt to complex and variable environments, demonstrating a sophisticated level of intelligent automation. Their capabilities extend to the ability to browse the internet, use applications, read and write files, execute payment transactions or even control various computer functions. The development of AI agents is a step closer to achieving Artificial General Intelligence (AGI), where machines can perform diverse tasks with a level of flexibility and efficiency comparable to human capabilities.

There are multiple AI agents available, but for this study, one of the most popular AI agents, AutoGPT, was selected. AutoGPT is and open-source Python application which can be run through CLI or using UI. At its core, it operates on a relatively straightforward concept: it uses self-prompting technique to make decisions and perform tasks autonomously. The results of each action are fed back into the system, enabling it to progressively work towards a defined objective. This recursive process distinguishes AutoGPT as an agent capable of executing actions on behalf of its users. Users are still required to authorize every action performed by AutoGPT or they can enter "continuous" mode, which lets to run the AI without user authorization. However, it is not yet recommended by authors of AutoGPT, due to potential risks, such as execution of unwanted actions or getting into an infinitive loop. The process ends by saving all its finding to the system's files for easy access and terminates the task when completed. It is important to mention that AutoGPT is designed as a generalist agent, meaning it is not restricted to specific tasks but can work on a wide range of tasks that are completable on a computer.

The core elements of AutoGPT's workflow are presented in the Figure (3) and described below:

1. **User Input:** The user inputs a sentence starting with "I want Auto-GPT to," or specifies the name, role and up to five goals for the AI.

2. **Input Preprocessing:** User's input is passed to OpenAI's API for input preprocessing and generating an internal monologue that outlines the AI's understanding by providing planned actions (*THOUGHTS*), strategy (*REASONING*), tasks to complete in order to reach provided goals (*PLAN*), optional *CRITICISM* and the next AI agent command (*SPEAK*).

3. **Task Queue:** All identified tasks are placed in a queue for execution.

4. **Task Execution:** The AI begins working on the tasks using the OpenAI's API, the internet and other applications, while interacting with its memory to save and retrieve information. The executed command returns a string value. For example, the *read_file* command would return a summary of file read from the system, the *execute_python_code* command would

return python code results in natural language, the browse_website command would return a summary of the scraped website contents, the *write_to_file* would return the status of writing to a file.

5. **Task Creation:** New tasks are created based on the results of previous executions, with continuous memory interaction for context and result storage.

6. **Task Prioritization:** Newly created tasks are added to the queue, where the "Task Prioritization" system reprioritizes the task list based on the new tasks generated and their priorities, using OpenAI's API to assist in the prioritization process.

The process loops through steps 3-6, continually generating and executing tasks and leveraging both internal and external resources, until the final goal is reached and the "Done" state is achieved.

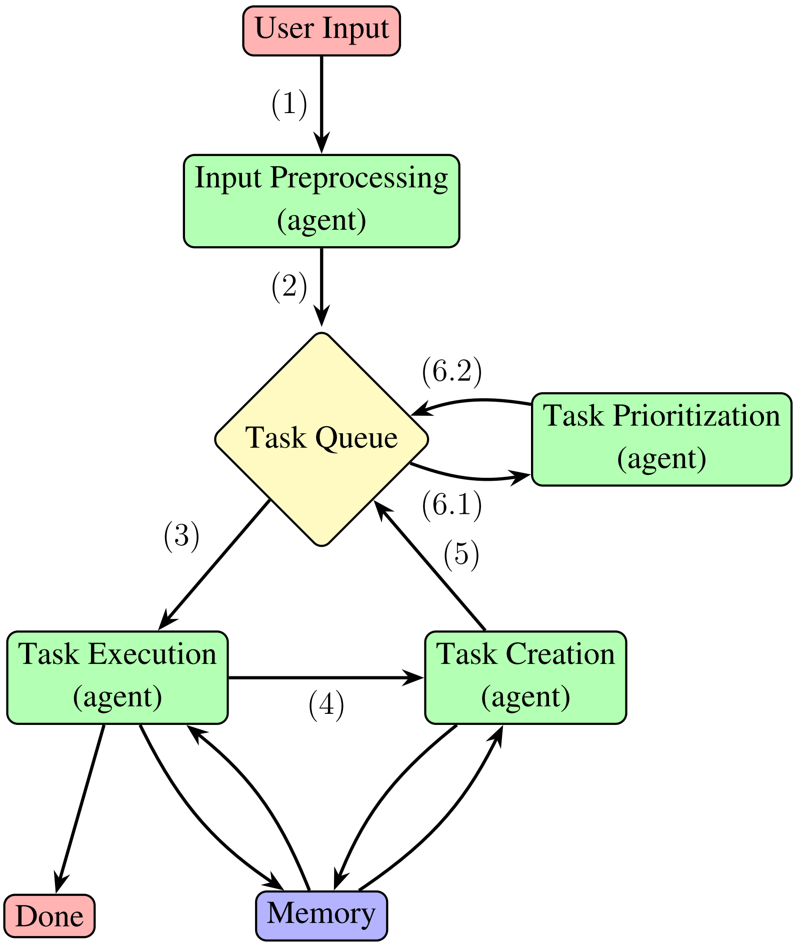


Figure 3. Conceptual Workflow of AutoGPT's Autonomous Task Processing Cycle. This diagram illustrates the AI agent's ability to independently process natural language input, prioritize tasks and execute a sequence of actions to achieve specified goals.

In conclusion, while utilizing AutoGPT presents significant advantages, it is important to acknowledge and address its limitations. There are multiple factors to consider when using AutoGPT, like cost considerations, particularly due to its reliance on OpenAI's GPT-4 API, known for its advanced capabilities but also its higher price. Every step working on the task requires a corresponding interaction with the API which incurs a token fee, accumulating costs with every loop required to reach the goal. Additionally, the risk of AutoGPT becoming stuck in the loop can further increase these expenses. Another concern is the chance to generate inaccurate or "hallucinated" content, which can compromise the reliability and effectiveness of the tool. Thus this AI agents advancement also shows the need for ongoing human oversight, particularly in complex decision-making scenarios, to ensure the ethical and effective use of such technology. The ultimate goal is to harness the potential of AI agents within a framework where they complement human skills, leading to greater efficiency and innovation across various sectors. Understanding and addressing these challenges is crucial for maximizing the benefits of AutoGPT in process automation.

2.3 Prompt Construction

This subsection continues with an examination of effective prompt construction techniques for AI agents, emphasizing the importance of skilled prompt engineering in automation using LLMs. Various strategies and best practices in prompt design are explored, integrating both functional and non-functional requirements from IT project management to formulate more precise prompts for AutoGPT. The subsection examine effective prompt construction techniques for LLMs, emphasizing the importance of skilled prompt engineering in automation using AI agents.

2.3.1 Overview of Prompt Engineering

Effective communication with AI chatbots, such as ChatGPT, relies heavily on the skill of prompt engineering. This process involves shaping a user's input or instruction in a way that guides the response of the large language model (LLM). In the context of code generation and software development, a prompt might vary from a broad description of the problem to a more detailed specification of input, output and functional requirements. The design of these prompts is crucial, as it directly influences the model's output, ensuring more accurate, relevant interactions and increasing the reliability of LLM results [44].

Even though AI agents like AutoGPT are built as a selft-prompting systems, but they still need initial steps from a human and it still require well-crafted prompts. Thus, majority of the core principles of prompt construction apply to both ChatGPT and AutoGPT, which utilize OpenAI's API. Prompt engineering, therefore, emerges as a crucial skill in the domain of LLMs, requiring a deep understanding of effective prompt design principles, including:

1. **Being Specific and Direct:** It's essential to ensure that prompts are clear and unambiguous. Clarity in prompts helps in reducing misunderstandings and misinterpretations by the LLM, leading to more accurate responses. This includes using straightforward language and avoiding complex jargon unless it's necessary for the context.

2. **Context Inclusion:** LLMs perform better with contextually rich prompts. Including relevant background information or specifying the task's context, together with guiding the LLM to understand the exact nature of the task and the information required, can enhance the model's understanding and response quality. This is especially important in complex tasks where the model needs to understand the broader context. It could be acomplished by including important details in the prompt, asking LLM to act in a relevant role, by specifying the steps required to complete the task or how the final output should look like.

3. **Providing Examples:** Incorporating examples within prompts, a technique known as "few-shot" prompting, can guide LLMs toward the desired output format or style. In coding tasks, providing code snippets as examples can clarify expected outcomes.

4. **Appropriate Formatting:** The format of a prompt can greatly influence the LLM's response. In software development, structured formats such as bullet points, numbered lists, delimiters like triple quotation marks, section titles or clear instructions can lead to more effective code generation.

5. **Task Decomposition:** Breaking down complex tasks into simpler sub-tasks can make it easier for the LLM to process and respond accurately. This approach is particularly beneficial in more complex projects, although AutoGPT attempts to address this aspect autonomously.

6. **Iterative Prompting:** Using an iterative approach, where responses are refined through subsequent prompts, can improve the model's output over time. This method is useful for gradually perfecting LLM's solutions, despite AI agents like AutoGPT already incorporating this feature to an extent.

Prompt engineering is not just about instructing an LLM, but engaging in a nuanced dialogue where each prompt is a blend of clarity, specificity and creativity. This subsubsection explores these principles in depth, providing a comprehensive understanding of how to effectively communicate with LLMs, thereby harnessing their full potential in diverse applications such as software development for data analysis automation.

2.3.2 Formal Requirements for Software Development Projects in Prompt Construction

The successful implementation of software development projects significantly depends on the accurate definition and consistent adherence to formal requirements [23]. These requirements are the cornerstone of project planning and execution, ensuring that the final product meets the end-user's needs and complies to quality and performance standards. Formal requirements defines what system needs to be built. Thus, the clarity, specificity and comprehensiveness of these requirements play a critical role in guiding the development process from the beginning to completion.

These requirements are broadly classified into functional and non-functional:

- **Functional Requirements:** Functional requirements define the specific behavior or functions of the software system, describing what the software should do, the actions it must perform, the data it processes and the expected outputs. They directly reflect the user's needs and are essential for defining the core functionalities of the software system. In the context of prompt construction for LLMs, functional requirements can be translated into explicit instructions or detailed specification that guide the model to achieve the final goal. In this case - perform automated software development for data analysis on selected datasets. For example, a prompt may include specific inputs and desired outputs, data processing methods, clearly outlining the functionality that needs to be developed. It should be clear what have to be done in order to comply with particular functional requirement, for example: *"Develop a Python script to import Lithuanian traffic accidents data from 'traffic_accidents.csv'. Ensure the script handles Lithuanian text encoding and verifies data integrity upon import."* To comply with the requirement, Python script must be developed to import data from *'traffic_accidents.csv'*, ensuring it correctly handles Lithuanian text encoding (likely UTF-8) and includes checks for data completeness and format consistency to maintain data integrity. This involves testing the script with the actual dataset to validate its functionality and effectiveness in data handling.

- **Non-Functional Requirements:** Non-functional requirements often define the quality attributes of the system. They focus on the operational criteria of a software system, such as performance, security, usability and reliability. They are essential for ensuring that the software operates efficiently and effectively within its intended environment. These requirements are not about what the software does, but how it does it. When constructing prompts for LLMs, these requirements can be incorporated by specifying constraints and standards the software needs to meet, such as budget, scalability, data privacy or user experience guidelines.

Together, functional and non-functional requirements form the framework for managing software development projects. Functional requirements drive the creation of the system's capabilities, directly impacting the system's functionality. Non-functional requirements ensure that the system is reliable, efficient and user-friendly, contributing to the overall quality and performance. A balanced focus on both is essential for developing a comprehensive and effective software solution.

Integrating formal requirements into the construction of prompts for AutoGPT could increase the reliability of LLMs in software development. Effective prompt construction requires balancing between detailed requirements and maintaining prompt clarity. A well-designed prompt should combine both functional and non-functional requirements. This approach ensures that prompts are not only technically accurate but also contextually rich, leading to more reliable and effective software development for data analysis.

2.4 Datasets Selection

This subsection discusses the criteria for selecting appropriate datasets for this thesis. The choice of datasets is crucial in ensuring the validity and relevance of the research findings, as they provide the foundation for testing and evaluating the AutoGPT's performance in practical software development for data analysis scenarios.

The primary objective of selecting datasets is to evaluate the capabilities of AutoGPT in software development for data analysis. The focus on collecting high-quality datasets that are diverse and from different domains, including traffic accidents, health data from a watch, sports and transactional data. This diversity ensures a comprehensive assessment of AutoGPT's adaptability and performance in different data contexts. It allows the study to cover a wide range of data analysis scenarios, from individual health tracking to broader societal issues.

The selection criteria prioritize non classic tutorial datasets but datasets that more likely were not part of the GPT-4 model's training data. This approach aims to test how effectively LLMs reason and adjust to new, unseen data. However, to ensure the datasets' relevance and quality, there is a focus on popular data topics widely analyzed in the field. This balance between novelty and familiarity is testing AutoGPT's versatility and robustness.

Three datasets are included in the analysis to provide a broad spectrum of data types and contexts. This number is optimal to ensure a thorough analysis without complicating the research process or inflating OpenAI's operational costs, allowing for a detailed exploration of AutoGPT's capabilities in each case. The datasets are collected from various sources:

- **Apple Health Data:** This dataset includes multiple dimensions such as steps, heart rate, hand-washing, sleep and workouts. The focus for this thesis is on steps and workout metrics. The choice of these metrics offers insights into daily health and activity patterns, providing a real-world application scenario for data analysis. It is personal data collected via Apple Watch and iPhone over multiple years.

- **Traffic Accidents in Lithuania:** This dataset provides data on traffic accidents, including year and month of occurrence, county, a flag indicating if an intoxicated person caused the accident and the accident count. The dataset presents a perspective on public safety and regional traffic issues. Lithuanian traffic accidents data from the Department of Statistics Lithuania (open data, downloaded in CSV format [27]).

- **Bitcoin Transaction Hashes:** Extracted from BigQuery public data (see Listing 1), this dataset involves transaction hashes from Bitcoin blockchain, adding a fintech dimension to the analysis. It also requires the use of more advanced data extraction and processing techniques.

Each dataset was manually exported from its source for reproducability and for AutoGPT to load them from the local machine where it runs. Preprocessing was minimal, primarily focusing on format conversion to ensure compatibility and readability for AutoGPT. Data was mainly kept raw to challenge AutoGPT's ability to handle and analyze unrefined data. Due to cost constraints and API usage limits, smaller subsets of the datasets were selected for the experiments.

The personal nature of the Apple Health data is ethically compliant, used with author's consent. Other datasets, being publicly available, do not pose privacy concerns. Although diverse and complex, the datasets are manageable in size, allowing detailed analysis without overwhelming the processing capabilities of a personal laptop or excessive use of the OpenAI API.

AutoGPT's performance will be evaluated on its ability to automate software development for data analysis across these datasets. This will include analyzing its effectiveness in interpreting data, generating insights and developing appropriate software solutions. A comparative analysis across different datasets will highlight AutoGPT's adaptability to varying data types and complexities.

While the chosen datasets are diverse, there is an acknowledgment of limitations in their scope and representation. The relevance of these datasets to specific domains may not comprehensively cover all possible scenarios in software development for data analysis.

The selection of these datasets is integral part in assessing the versatility and effectiveness of AutoGPT in software development for data analysis. The diversity of these datasets, combined with minimal preprocessing, provides a robust framework for assessing AutoGPT's capabilities across various domains and scenarios.

2.5 Performance Evaluation Strategy

In this subsection, a comprehensive strategy for evaluating the performance of Large Language Models (LLMs) in automating software development for data analysis is described. This evaluation strategy is critical for assessing the effectiveness, efficiency and adaptability of AI agents, such as AutoGPT, in automating software development processes within practical applications.

The primary objective of creating the evaluation framework is to appropriately evaluate the capabilities of AutoGPT in software development for data analysis. The focus is on creating a robust framework that would help to evaluate various practical aspects of LLM usage in process automation, helping to conduct a comparative analysis of its performance.

The experimental approach utilizes three distinct prompt construction techniques across three diverse datasets. For each experimental setup, data on several key metrics are recorded to comprehensively evaluate AutoGPT's performance. These metrics include:

- **Steps to Completion:** This metric shows a number of interactions between the user and AutoGPT required to reach the final goal. It assesses the level of user involvement needed for successful task completion, indicating how intuitive and autonomous AutoGPT is in software development tasks. It is important to mention, in the most common scenario, user has to provide only the first prompt and further is only required to authorize each action performed by AutoGPT. In other words, it calculates AI agent's steps required to complete the task.

- **Total Runtime:** This metric measures the total time from the initiation to the completion of a task by AutoGPT, essentially tracking the operational duration of the AI agent. It reflects the efficiency of AutoGPT in processing and completing assigned tasks, providing insight into its practical usability in terms of runtime.

- **API Request Count:** This measures the number of API requests made by AutoGPT during the execution of each task. It serves as an indicator of how frequently AutoGPT communicates with OpenAI's API.

- **Token Utilization:** This metric evaluates the amount of tokens used by AutoGPT in each session. It is important for understanding the AI agent's efficiency in process automation and for assessing the cost-effectiveness.

- **Operational Cost:** This metric calculates the associated costs in dollars of running AutoGPT experiments, consisting of API usage fees. It provides insights into the financial aspects of AutoGPT usage.

- **Requirement Fulfillment:** This metric evaluates if the final output generated by AutoGPT meets the predefined formal requirements. The analysis is categorized into three distinct levels:

  - *Not Met:* The output does not comply with the established formal requirement.
  - *Partially Met:* The output meets some parts of the formal requirement but has deficiencies.
  - *Fully Met:* The output completely adheres to the specified formal requirement.

  This metric serves as a measure of quality, ensuring that the outputs not only solve the task but also adhere to specific standards and protocols. It directly assesses the quality of the output in terms of specified standards and requirements completion. It helps to identify areas where AutoGPT needs improvement in understanding and complying to formal requirements.

Each metric is designed to provide a comprehensive understanding of AutoGPT's performance, highlighting its strengths and areas for improvement in automating software development for data analysis tasks. This set of metrics will help to create a comparable, multidimensional analysis of AutoGPT capabilities.

AutoGPT autonomously selects the most appropriate OpenAI model for each specific task, analyzing the task's nature and choosing the most suitable model from OpenAI's range. This process is automated, leveraging the strengths of different models to efficiently handle various tasks without requiring continuous user input. Consequently, the last three metrics ("API Request Count", "Token Used" and "Operational Cost") are differentiated based on the model versions used in the experiments:

- **GPT-3.5-turbo:** This version is an advanced iteration that precedes GPT-4, offering significant improvements in language understanding and generation capabilities compared to its predecessors. GPT-3.5-turbo provides a balance between computational efficiency and sophisticated language processing, making it suitable for a wide range of tasks that require a high level of language comprehension and output quality.

- **GPT-4:** It is the latest and most advanced model in the GPT series at the time of this study. GPT-4 is a large multimodal model that can solve difficult problems with greater accuracy than any of OpenAI's previous models. It has a broader general knowledge base and advanced reasoning capabilities, making it able to handle complex tasks and providing more accurate responses across various domains.

- **Text-embedding:** A model focused on generating text embeddings, which are numerical representations of text data. It is part of OpenAI's models that emphasize efficient text representation for various NLP tasks. Unlike GPT-3.5 and GPT-4, which are more focused on language generation and interaction, Text-embedding is specialized in converting text into a form that can be easily processed and analyzed by AI systems.

This differentiation not only allows an evaluation of AutoGPT's overall performance but also an understanding of how different model versions impact accuracy and operational costs. By comparing these metrics across models and datasets, insights are gained into each model's efficiency and cost-effectiveness in real-world applications.

This evaluation strategy is essential for validating AutoGPT's capabilities together with identifying areas of improvement and exploring its potential applications in real-world scenarios.

3 Experiment Setup

3.1 Preprocessing of Datasets

This subsection outlines the preprocessing steps applied to each dataset used in this study. With a focus on evaluating AutoGPT's ability to process and analyze data in a relatively raw state, the preprocessing efforts were intentionally minimal, primarily focusing on format conversion and essential data cleaning.

Each dataset was manually exported from its source to ensure reproducibility and to allow AutoGPT to load them from the local machine where it operates. Preprocessing was minimal, primarily focusing on format conversion to ensure compatibility and readability for AutoGPT. Data was mainly kept raw to challenge AutoGPT's ability to handle and analyze raw data.

- **Apple Health Data:** Data was manually extracted from a personal iPhone in a single export.xml file. A custom script `extract_apple_health_from_xml.py` was then run, which was specifically designed to parse and extract hand-washing, step count and workout data from the XML file. Subsequently, these data points were saved into three distinct files: *hand_washing.csv*, *step_count.csv* and *workout.csv*. This process ensured that the data was structured and formatted in a manner that was easily accessible by AutoGPT. Step count and workout datasets schemas can be found in Table 10 and Table 11 correspondingly.

- **Traffic Accidents in Lithuania:** Road accidents data has been extracted from Lithuanian State Data Agency and saved as a CSV file on December 12, 2023 [27]. The specific indicator chosen was "Number of road accidents in which people were injured," found under "Indicators of road accidents" in the "Road transport" category of the "Transport and communication" section. Following dimensions filters were applied:

  – Time Period: The entire range from January 2005 to October 2023.

  – Road Accidents: Both available categories were chosen, including "Number of road traffic accidents" and "Number of road traffic accidents by drivers under alcoholic affect".

  – Administrative Territory: Focused solely on the county level, including all 10 counties in Lithuania.

  Full dataset schema can be found in Table 12.

- **Bitcoin Transaction Hashes:** The Bitcoin transaction hashes were extracted from a public BigQuery dataset. To manage the scope and relevance of the data, only the 1000 most recent observations were selected from October 2023 (Appendix C.3). During preprocessing, only the transaction hashes were kept, which serve as a unique identifier for each transaction on the blockchain. This focused approach ensured the dataset was manageable and suitable for analysis while preserving the essential characteristics of the blockchain data. Dataset schema can be found in Table 13.

In all cases, the chosen datasets were capped to 100 samples to manage computational and cost constraints effectively. This sample size was considered sufficient for preliminary analysis and for testing the functionality of AutoGPT generated codes, although it may not capture the full diversity or represent the entire dataset comprehensively. It is also important to mention that the main goal for each dataset is to automate software development process for data analysis. Thus, majority of AutoGPT generated codes were also rerun on full datasets in order to test their effectiveness.

The preprocessing steps for each dataset have been designed to make them ready for analysis with AutoGPT while maintaining their raw characteristics. The minimal preprocessing approach was chosen to challenge and assess AutoGPT's capabilities in handling and analyzing data that is not fully prepared for analysis. This aligns with thesis objective to evaluate AutoGPT's performance in practical, real-world scenarios and helps to perform a comprehensive assessment of AutoGPT's adaptability and performance across a range of datasets.

3.2 Setting up AutoGPT for Data Analysis

This subsection describes the configuration process of AutoGPT, outlining the technical setup and model parameters used in the experiments.

Most of the data extraction and preparation related code was written in Python. Only "Bitcoin Transaction Hashes" were extracted through BigQuery in the Google Cloud console by running SQL query (shown in Appendix C.3).

All datasets were manually exported from their respective sources and stored locally in the *auto_gpt_workspace* subfolder, located within the main AutoGPT directory. This subfolder serves as the default location for AutoGPT to save its results.

Basic setup requires to have Python and Docker installed on the system. AutoGPT has been setup following the instructions provided in the official AutoGPT GitHub page, selecting the recommended setup with Docker. The use of Docker, a containerization technology, allows for the creation, deployment and execution of applications in a controlled and isolated environment, which is crucial for maintaining consistency across experiments.

In this thesis, version v0.4.7 of AutoGPT was used, which mainly uses OpenAI's GPT-4 API. This version was the latest AutoGPT stable version during the thesis writing. One of the prerequisites for using this system involves setting up an OpenAI billing account, which is necessary due to the limitations imposed on API calls. This setup ensures uninterrupted access to the OpenAI's most recent GPT models. AutoGPT is designed to autonomously select the most suitable OpenAI model for each specific task, based on an analysis of the task's nature.

Considering the primary goal of the thesis, which is to automate the process of software development for data analysis, emphasis was placed on model reproducibility, accuracy and consistency over creative output. Thus, the "temperature" setting for the GPT models was adjusted to 0.2. This parameter in the ChatGPT API influences the randomness or creativity of the model's responses, with a value range between 0 and 1. A lower temperature value, like 0.2, leads to more focused and deterministic outputs, aligning with the need for precision in automated software development tasks.

3.3 Executing Prompt Construction

This subsection explores three different complexities of prompt construction: simple natural language prompts, advanced structured prompts and prompts integrating both functional and non-functional requirements. This exploration is crucial to understanding how varying levels of specificity in prompts influence the performance of AutoGPT in automated software development, providing insights into the balance between guiding precision and creative freedom in AI interactions.

Additionally, this subsection includes a comprehensive presentation of all prompts and formal requirements for each dataset used in Section 4.

In this thesis, three different complexities of prompt construction are explored:

- **Simple Natural Language Prompts:** These prompts are straightforward and phrased in everyday language. They typically require a basic level of understanding and do not involve complex instructions or multiple steps. The focus is is on assessing how AI agents like AutoGPT respond to user-friendly, simple, conversational prompts. These prompts are easy and quick to compose, requiring minimal effort.

- **Advanced Structured Prompts:** These prompts are more detailed and follow a specific structure by providing clearer goals for AutoGPT. They often involve multiple steps or require the AI to process and integrate multiple pieces of information. This category aims to analyze AutoGPT's performance in handling tasks by providing more complex, structured and informative prompts. Writing these prompts demands a moderate level of effort, as they require a clear understanding of the task at hand and the ability to articulate it in a structured and detailed manner.

- **Prompts Integrating Both Functional and Non-Functional Requirements:** These prompts combine functional requirements (what needs to be done) with non-functional requirements (how it should be done, under what constraints or conditions). This type of prompt is designed to assess the AI's proficiency in delivering complete solutions that meet both the technical and practical needs of a project. Creating these prompts involves a high level of effort and expertise. The writer must not only define the formal requirements but also seamlessly incorporate then into prompt construction.

Each category represents a different level of sophistication in prompt engineering. Moving from simple natural language prompts to prompts integrating both functional and non-functional requirements, the complexity increases, requiring more advanced understanding and specificity. This approach help to perform a comparative and exploratory analysis in the context of automated software development, focusing on how functional and non-functional requirements impact the decision quality of AI agents like AutoGPT.

In the following subsubsections, the exact prompts used in the experiments and the formal requirements for each dataset are provided. This inclusion ensures a transparent and detailed view into the experimental setup. Given that the thesis has the same purpose for each dataset, it is important to mention that a majority of non-functional requirements overlap.

3.3.1 Formal Requirements and Prompts for Apple Health Data

In this subsubsection, the focus is on the Apple Health Data, specifically detailing the formal requirements and the three types of prompts used in the analysis. This dataset, including key health metrics such as steps and workouts, offers valuable insights into personal health patterns and physical activity trends. The primary goal with the Apple Health Data is to conduct an analysis of daily step counts, organized by day of the week and calculate the number of workouts by year and workout type. This approach aims to uncover patterns and correlations in the data, providing a nuanced understanding of personal health habits across time and activity type.

Formal Requirements This part outlines the formal requirements for the Apple Health Data analysis, including both functional and non-functional aspects. These requirements are essential to ensure that the analysis is complete and comply with the stated objectives. They are also especially crucial for the development of the third type of prompt.

Functional Requirements For the analysis of the Apple Health Data, the following functional requirements were established:

- The system must import and process data from separate *.csv* files: *step_count.csv* and *workout.csv*. Use the *creationDate* as the date reference for all analyses.
- All data preparation and analyses should be performed using Python.
- Calculate daily step count together with 7 days rolling average and create a graph from it where x axis shows date and y axis shows steps count.
- Calculate daily average step count grouped by the day of the week. Provide bar chart where x axis shows day of the week and y axis shows daily average step count.
- Count number of workouts grouped by year and type of workout (*workoutActivityType* column). Provide bar chart where x axis shows year, y axis shows workouts count and bars are split and colored by the type of workout.
- Create a dashboard / R Shiny app including performed analysis. Add all created graphs to it.
- Provide 2-4 actionable insights.
- Write a user guide and documentation.

Non-Functional Requirements The non-functional requirements for the system are as follows:

- Easy to understand graphs and tables. Dashboard should be easy to navigate, intuitive.
- Ability to handle large amounts of data efficiently.
- The project should be developed within a predefined budget and ongoing operational costs should be reasonable.

These requirements describe the analysis approach for the Apple Health Data and are integral to achieving the research objectives. Furthermore, they play a crucial role in the development of the third type of prompt, which integrates both functional and non-functional requirements.

Simple Natural Language Prompt This prompt for the Apple Health Data was designed to be intuitive and easy to understand. It is phrased in everyday language and aim to initiate basic data analysis tasks.

> "Analyze Apple Health Data for step count and workout frequency using Python, from respective .csv files available in the workspace and present the findings in an R Shiny report."

Advanced Structured Prompt This prompt represent a more sophisticated level of interaction. It is more detailed and structured, providing specific instructions and clear objectives for data analysis.

> "Analyze Apple Health Data using Python, from respective .csv files: 'data_to_read/workout_100.csv' and 'data_to_read/step_count_100.csv'. Perform all data preparation and analyses using Python and compile a final report using R Shiny. Use the 'creationDate' as the date reference for all analyses. The tasks include computing the daily step count and its 7-day rolling average, visualizing these trends in a graph, summarizing average step counts by day of the week in a bar chart, counting workouts annually by type (workoutActivityType) in a bar chart."

Prompt Integrating Both Functional and Non-Functional Requirements This prompt is the most complex. It not only specify what analytical tasks need to be performed (functional requirements) but also consider how these tasks should be executed (non-functional requirements).

> "Using Python, analyze Apple Health Data from 'data_to_read/workout_100.csv' and 'data_to_read/step_count_100.csv', focusing on step count and workouts, with 'creationDate' as the date reference. Calculate the daily step count and 7-day rolling average, presenting it in a graph with dates on the x-axis and step counts on the y-axis. Summarize daily average step count by weekday in a bar chart and count workouts by year and type in another bar chart, with years on the x-axis and counts on the y-axis, differentiated by workout type (workoutActivityType). Create an intuitive, easy-to-navigate R Shiny dashboard to display these analyses and provide 2-4 actionable insights. Include efficient data handling, even with large datasets and ensure the project stays within budget, with reasonable operational costs. Accompany with a user-friendly guide and documentation."

3.3.2 Formal Requirements and Prompts for Traffic Accidents in Lithuania

In this subsubsection focuses on the Traffic Accidents in Lithuania dataset, detailing the formal requirements and the three types of prompts used in the thesis. This dataset provides valuable insights into traffic accidents in Lithuania, categorizing them by year, month, county and whether they were caused by drivers under the influence of alcohol. The primary goal with this dataset is to conduct a comprehensive analysis of monthly traffic accidents and to compare accident counts by county, along with assessing the proportion of accidents caused by drivers under the influence of alcohol. This analytical approach is intended to reveal any observable trends and patterns within the data.

Formal Requirements This part defines the formal requirements for the analysis of the Traffic Accidents in Lithuania dataset, including both functional and non-functional aspects. These requirements are essential not only for ensuring the completeness of the analysis but also for the development of the third type of prompt.

Functional Requirements For the analysis of the Traffic Accidents in Lithuania, the following functional requirements were established:

- The system should be able to import Lithuanian traffic accidents data provided in a single *.csv* file *data_to_read/traffic_accidents_100.csv*.
- All data preparation and analyses should be performed using Python.
- Calculate the total number of road traffic accidents every month to see if there's any trend or seasonality. Plot it like a curve where x axis shows year and month, and y axis shows the number of road traffic accidents.
- Compare the total number of road traffic accidents to those caused by drivers under the influence of alcohol over time. Plot it like a curve where x axis shows year and month, and y axis shows the number of road traffic accidents split by if accident was caused by driver under the influence of alcohol or not.
- Calculate the total number of accidents in each county and the proportion of road traffic accidents by drivers under alcoholic affect in each county. Visualize it using a bar chart where x axis shows county and y axis shows number of accidents colored by if accidents caused by drivers under alcoholic affect.
- Create a dashboard / R Shiny app. Add all graphs and tables described in functional requirements.
- Provide 2-4 insights from analysis performed.
- Write a user guide and documentation.

Non-Functional Requirements The non-functional requirements for the system are as follows:

- Easy to understand graphs and tables. Dashboard should be easy to navigate, intuitive.
- Ability to handle large amounts of data efficiently.
- The project should be developed within a predefined budget and ongoing operational costs should be reasonable.

These requirements describe the analysis approach for the Traffic Accidents in Lithuania and are integral to achieving the research objectives. Additionally, they are crucial in the development of the third type of prompt, which integrates both functional and non-functional requirements.

Simple Natural Language Prompt This prompt for the Traffic Accidents in Lithuania was designed to be intuitive and easy to understand. It is phrased in everyday language and aim to initiate basic data analysis tasks.

> "Analyze Traffic Accidents in Lithuania data from 'data_to_read/traffic_accidents_100.csv' available in the workspace for monthly trends in road accidents and distribution between counties taking into account the impact of alcohol on these accidents using Python and present the findings in an R Shiny report."

Advanced Structured Prompt This prompt represent a more sophisticated level of interaction. It is more detailed and structured, providing specific instructions and clear objectives for data analysis.

> "Analyze Lithuanian traffic accidents using Python from 'data_to_read/traffic_accidents_100.csv' file. Transform data to a wide format using "Administrative territory" and "Road accidents" columns. Perform all data preparation and analyses using Python and compile a final report using R Shiny. Calculate monthly accident count and compare accidents involving alcohol, display in a curve graph. Calculate the total number of accidents in each county and the proportion of road traffic accidents by drivers under alcoholic affect in each county and show in bar charts. Provide key insights from the analysis."

Prompt Integrating Both Functional and Non-Functional Requirements This prompt is the most complex. It not only specify what analytical tasks need to be performed (functional requirements) but also consider how these tasks should be executed (non-functional requirements).

> "Analyze the Lithuanian traffic accidents data from 'data_to_read/traffic_accidents_100.csv' using Python. Focus on identifying monthly and yearly trends in road accidents, including a comparison with alcohol-related incidents. Present these trends in curve graphs, with the x-axis showing year and month and the y-axis the number of accidents. Additionally, analyze and display in a curve the total number of accidents split by alcohol-related and not. Calculate the total number of accidents in each county and the proportion of road traffic accidents by drivers under alcoholic affect in each county. Visualize it using a bar chart. Create an R Shiny dashboard to display these analyses, ensuring the dashboard is intuitive and can handle large datasets efficiently. The project should stay within budget, with reasonable operational costs. Include a user guide and documentation with 2-4 actionable insights derived from the data."

3.3.3 Formal Requirements and Prompts for Bitcoin Transaction Hashes

This subsubsection outlines the formal requirements and the three types of prompts for analyzing Bitcoin Transaction Hashes. The dataset includes only Bitcoin transaction hashes, necessitating the extraction of additional details such as timestamps, amounts and other relevant information from a provided API. Moreover, the data retrieved from the API requires processing to become human-readable. As is well-known in data analysis, the process of data processing plays a very significant role and results significantly depend on efficient and correct data extraction and preparation. Thus, this approach emphasizes more on data extraction from the API and the preparation step than on the analytical insights part.

Formal Requirements This part specifies the formal requirements for analyzing Bitcoin Transaction Hashes, emphasizing the extraction and preparation of data from an API, followed by basic statistical analysis. The importance of data processing is underscored, acknowledging its significant role in influencing the outcomes of the analysis.

Functional Requirements The key functional requirements for the analysis of Bitcoin Transaction Hashes are:

- The system must import and process data from *data_to_read/bitcoin_transactions_100.csv*.
- All data preparation and analyses should be performed using Python.
- Extract available information for each Bitcoin transaction hash using *https://blockchain.info/rawtx/$tx* API, where *tx_hash* is transaction hash provided in dataset.
- Transform unix timestamps extracted from the API to human-readable datetime (YYYY-MM-DD hh:mm:ss).
- Convert transaction values to BTC currency. blockchain.info API provide transaction values in Satoshi, given that 1 BTC is equal to 100 million Satoshis.
- Analyze data by extracting the first and last timestamps in given dataset, provide in scorecards.
- Calculate the total count of transactions together with total revenue in BTC and provide in scorecards.
- Calculate the total count of transactions together with total revenue in BTC daily and provide in a curve chart where x axis represent date and first y axis shows transaction count and second y axis shows revenue.
- Create a dashboard / R Shiny app including performed analysis. Add all created graphs to it.
- Provide 2-4 actionable insights.
- Write a user guide and documentation.

Non-Functional Requirements The non-functional requirements for the analysis system include:

- Easy to understand graphs and tables. Dashboard should be easy to navigate, intuitive.
- Ability to handle large amounts of data efficiently.

Given the nature of the data and the analysis process, these requirements are crucial in ensuring that the extraction and preparation of data from the API are executed efficiently and correctly, setting a foundation for meaningful statistical analysis. Additionally, they are crucial in the development of the third type of prompt, which integrates both functional and non-functional requirements.

Simple Natural Language Prompt This prompt for the Bitcoin Transaction Hashes was designed to be intuitive and easy to understand. It is phrased in everyday language and aim to initiate basic data analysis tasks.

> "Analyze the Bitcoin transaction data from 'data_to_read/bitcoin_transactions_100.csv' file. Use the blockchain.info API to extract details like timestamps and transaction values. Summarize analysis timeframe, main trends and daily transaction activities. Present the findings in a R Shiny report."

Advanced Structured Prompt This prompt represent a more sophisticated level of interaction. It is more detailed and structured, providing specific instructions and clear objectives for data analysis.

> "Analyze the Bitcoin transaction data from 'data_to_read/bitcoin_transactions_100.csv' using Python. Extract transaction information using the blockchain.info API, including timestamps (convert from Unix format to YYYY-MM-DD hh:mm:ss) and values (convert from Satoshi to BTC). Provide a basic summary of the analysis timeframe, total transactions and revenue in BTC, along with a daily breakdown of transactions and revenue. Present this data in an R Shiny dashboard."

Prompt Integrating Both Functional and Non-Functional Requirements This prompt is the most complex. It not only specify what analytical tasks need to be performed (functional requirements) but also consider how these tasks should be executed (non-functional requirements).

> "Develop a Python script to process 'data_to_read/bitcoin_transactions_100.csv'. Use https://blockchain.info/rawtx/$tx_hash API for extracting information from each Bitcoin transaction hash, where tx_hash is the transaction hash listed in the dataset. Transform Unix timestamps to human-readable format (YYYY-MM-DD hh:mm:ss) and convert transaction values from Satoshi to Bitcoin (1 BTC equals 100 million Satoshis). Analyze the dataset to identify the first and last timestamps and display these in scorecards. Compute the total number of transactions and the overall revenue in Bitcoin, showcasing these figures in scorecards. Additionally, create a curve chart to display the daily transaction count and revenue in Bitcoin, with dates on the x-axis and transaction counts and revenue figures on separate y-axes. Build an intuitive and efficient R Shiny dashboard to present these analyses. Include a user guide and documentation with 2-4 actionable insights derived from the data."

4 Experiment

This section presents the results and findings from experiments conducted with AutoGPT, focusing on its ability to automate the software development process for data analysis across three distinct datasets: Apple Health Data, Traffic Accidents in Lithuania and Bitcoin Transaction Hashes. Each experiment leverages the methodologies and prompts described in previous sections, exploring specific data analysis questions and unique data processing approaches applied to each dataset. The aim is to demonstrate AutoGPT's capabilities in real-world scenarios, testing its proficiency in writing Python code for data preparation and analysis, as well as creating R Shiny reports. The experiments progress from simple natural language prompts to more structured ones and finally to prompts integrating both functional and non-functional requirements. This comprehensive approach provides a detailed comparative review of the AI agent's performance across varying prompt complexities and datasets, concluding with the identification of limitations and challenges. The results offer valuable insights into the impact of functional and non-functional requirements on the decision quality of Large Language Models (LLMs) and their effectiveness in process automation tasks.

4.1 Analysis and Findings

This subsection presents the experimental results and findings on AutoGPT's ability to automate software development for data analysis. It is organized based on the three types of prompts — simple natural language prompts, advanced structured prompts and prompts integrating both functional and non-functional requirements — followed by a comparative analysis of these three categories. It provides a comparative review of AutoGPT's performance across various prompt types and datasets.

4.1.1 Simple Natural Language Prompts

The first part of the experiment focused on testing AutoGPT's performance using simple natural language prompts, which require significantly less effort to write compared to other types of prompts. Despite this simplicity, it is crucial that the prompts are informative enough for the AI agent to understand the main requirements and conduct the appropriate analysis. This section presents the execution of simple natural language prompts, highlighting how effectively AutoGPT interpreted and responded to these prompts. It concludes with an analysis of the results, emphasizing AutoGPT's proficiency and limitations in handling basic, conversational prompts.

Quantitative results for each dataset describing main metrics are presented in Table 1. Each metric description provided in Subsection 2.5.

Apple Health Data In the initial step, AutoGPT successfully identified the correct files. It wrote a Python script to read these files into a Python dictionary and then saved the output in a new JSON file. However, the R Shiny report generated by AutoGPT was empty, it only loaded the previously generated JSON file without displaying any content.

It took 16 steps to completion, mainly due to two reasons: debugging, which added four additional steps to fix an error and retry the script execution, and information search, which included three *web_search* commands to gather information on how to build an R Shiny dashboard. The task's complexity was further increased as it involved processing two CSV files, requiring more steps and additional API calls to OpenAI, potentially leading to higher operational costs.

AutoGPT did not provide any analytical insights or documentation and was unable to test the R script directly. However, it did provide guidance on script execution:

> "INFO SPEAK: I cannot execute the R script directly. Please execute the 'generate_shiny_report.R' script using R to create the R Shiny report."

In summary, while AutoGPT using a simple natural language prompt for Apple Health Data successfully generated appropriate goals (see Listing 3), the final output was not sufficient to derive any meaningful insights and the generated analysis lacked reproducibility.

Traffic Accidents in Lithuania In this experiment, AutoGPT completed the initial steps of loading the data and writing a Python script for data aggregation in 2 steps. However, instead of loading the data file directly into the Python script, AutoGPT read the provided file and inserted the data into the script. The Python script successfully aggregated the data and printed the results, which AutoGPT then copied and saved to a JSON file. This approach deviated from the required approach of writing a script to handle these tasks programmatically.

One significant challenge was AutoGPT's tendency to perform the analysis directly, rather than focusing on developing software for this purpose. This issue led to the prompt being run multiple times with adjustments, but each attempt resulted in an error due to exceeding the context length limit. It also made AI agent to fail creating a dashboard, providing instructions and insights.

In summary, while AutoGPT was able to set appropriate goals for the Traffic Accidents in Lithuania dataset using a simple natural language prompt (detailed in Listing 6), it faced difficulties in creating a reproducible and efficient workflow. Although it managed to save the analysis results in a JSON file and write a Python script for data aggregation, it was unable to develop a functional R Shiny report and inserted data into the scripts reduced reproducibility.

Bitcoin Transaction Hashes In this part of the experiment, AutoGPT correctly identified the required blockchain.info API and successfully extracted the necessary information, saving the raw data to a JSON file. However, instead of scripting this extraction process, AutoGPT performed theses steps manually, leading to results that cannot be reproduced or applied to the full dataset.

Then AutoGPT wrote separate Python scripts for each simple descriptive statistic, executed these scripts and stored results in text files. However, rather than integrating these results programmatically into the R Shiny report, AutoGPT manually inserted the text file contents into the report. As a result, the report, as shown in Figure 4, lacked visualizations and the Key Performance Indicators (KPIs) were not formatted into a human-readable format, remaining in the same format as extracted from the API.

To summarize, while AutoGPT set appropriate goals for the Bitcoin transaction hashes dataset (see Listing 9) and managed to write Python scripts for statistical analysis, the lack of reproducibility and direct data manipulation limited the usefulness of the results. The major drawback was AutoGPT's approach of directly extracting and handling data, which restricted potential scripts scalability and reusability for larger datasets.

Summary The experiments with simple natural language prompts revealed that AutoGPT's performance is notably sensitive to the precise wording of the prompt. Simple prompts, requiring

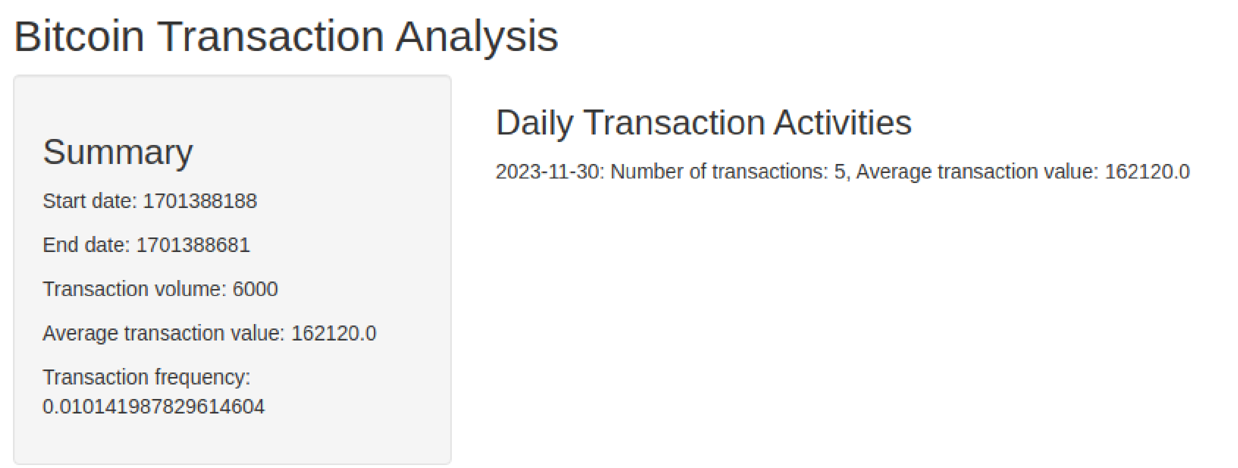


Figure 4. R Shiny Dashboard Created by AutoGPT Using Simple Natural Language Prompt for Bitcoin Transaction Hashes

minimal effort to create, led to varying results based on subtle changes in language. On average, tasks took about 7 minutes to run, with operational costs ranging between 1.4 to 1.69 dollars. The Bitcoin Transaction Hashes dataset showed the best results in both data processing and final report generation. However, a common limitation across all datasets was the lack of reproducible code and the absence of instructions or insights. Additionally, AutoGPT often attempted to conduct analyses directly rather than writing scripts for them as instructed, affecting the reproducibility and scalability of the solutions.

Table 1. AutoGPT Results Using Simple Natural Language Prompts

Metric		Apple Health Data	Traffic Accidents in Lithuania	Bitcoin Transaction Hashes
Steps to Completion		16	6	13
Total Runtime		7m3,466s	6m36,543s	6m43,710s
API Request Count	GPT-3.5	19	11	12
	GPT-4	15	10	14
	Text Embedding	6	8	10
Token Utilization	GPT-3.5	27561	17178	15722
	GPT-4	50843	41069	45763
	Text Embedding	20867	11945	8318
Operational Cost	GPT-3.5	0.04	0.03	0.02
	GPT-4	1.64	1.37	1.48
	Text Embedding	0.01	0	0

4.1.2 Advanced Structured Prompts

The second part of experiment was to test AutoGPT performance when shifting from simple natural language prompts to more advanced structured prompts. These structured prompts provide additional context and information, requiring greater effort to construct but potentially leading to higher quality results. This part presents the execution process of advanced structured prompts, detailing how effectively AutoGPT interpreted and responded to them. The following paragraphs explore the outcomes of using these prompts, highlighting AutoGPT's proficiency and limitations in handling software development tasks for data analysis.

Quantitative results for each dataset describing main metrics are presented in Table 2. Each metric description provided in Subsection 2.5.

Apple Health Data AutoGPT created multiple Python scripts: two for data processing and one for each graph, saving the generated graphs as PNG files. Subsequently, it developed an R Shiny dashboard containing three tabs, each corresponding to a specific analysis. However, the dashboard was non-functional initially, as AutoGPT did not test it and manual intervention was needed to move the PNG files to the appropriate folder for R Shiny. The final dashboard is provided in Figure 5.

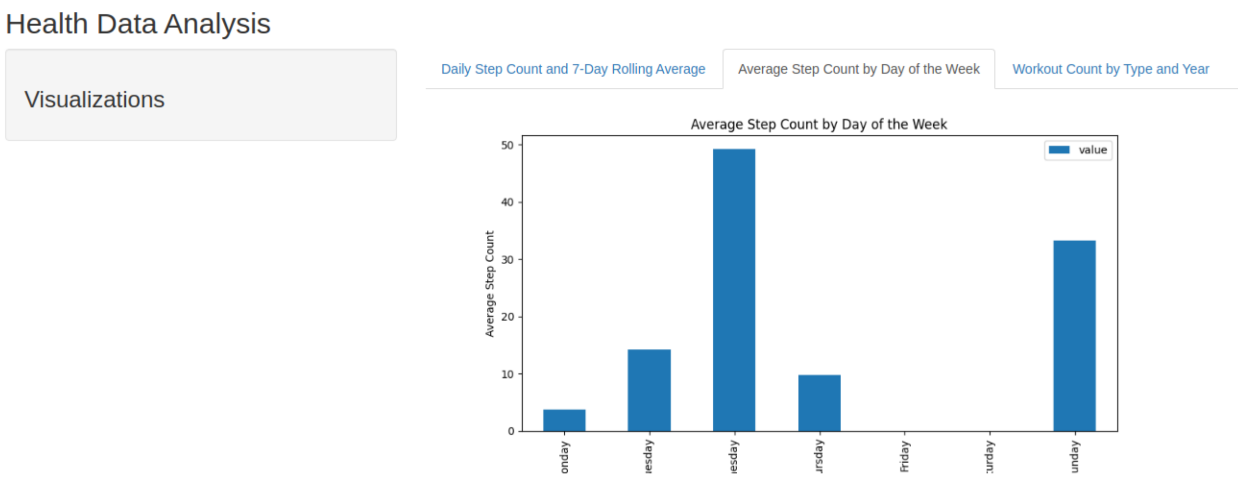


Figure 5. R Shiny Dashboard Created by AutoGPT Using Advanced Structured Prompt for Apple Health Data

While AutoGPT did not provide explicit insights or documentation, it did include a comment on how to execute the created R Shiny file. Notably, the software developed by AutoGPT was tested on the full dataset, where it functioned correctly, demonstrating that all analyses were reproducible and could be applied to the full dataset scope.

In summary, AutoGPT successfully generated the necessary Python scripts for data processing and analysis and created a final report that included all requested graphs. The scripts were reusable on the full dataset. However, a drawback was that the graphs were generated in Python and saved as PNG files, rather than being directly created in R, which would have been a more integrated approach for the R Shiny dashboard.

Traffic Accidents in Lithuania In this task, AutoGPT correctly identified the file to be processed but directly inserted the file data into the Python script rather than loading it from the file. This approach reduced the reproducibility of the code, as editing would be required to rerun the experiment with the full dataset.

The Python script transformed the data as needed and saved it to a new CSV file. This file was then used in an R Shiny report to generate the graphs. The report, illustrated in Figure 6, feature three separate tabs for each graph.

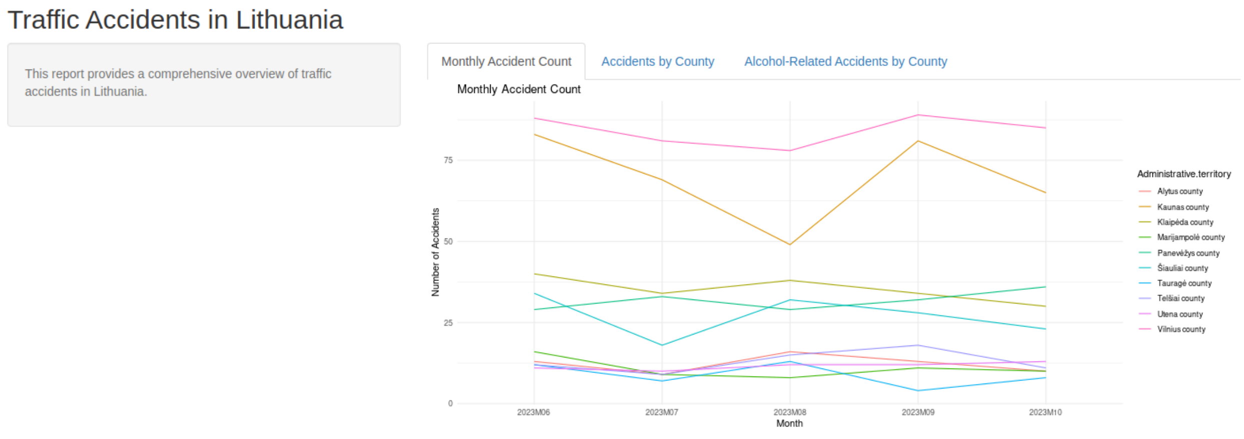


Figure 6. R Shiny Dashboard Created by AutoGPT Using Advanced Structured Prompt for Traffic Accidents in Lithuania

While AutoGPT developed a correct plan based on the provided prompt (see Listing 7), the analyses in the final report slightly deviated from the original request. No additional insights or specific instructions were provided, except for a brief comment on executing the R Shiny file.

In summary, although AutoGPT successfully set appropriate goals for the Traffic Accidents in Lithuania dataset with an advanced structured prompt (see Listing 7), it faced challenges in fully adhering to these goals. The primary issues were the lack of a fully reproducible workflow ,due to direct insertion of data into the Python script, and the deviation of the generated graphs from the requested analyses.

Bitcoin Transaction Hashes In this experiment, AutoGPT started by reading the provided file and directly inserting its data into the Python script, rather than specifying data load in the script itself. This approach reduce reproducibility, as editing the script would be necessary to apply it to the full dataset.

Subsequently, AutoGPT correctly identified and utilized the blockchain.info API to extract the required information, which was then saved into a new CSV file. The AI agent also wrote three additional Python scripts for data processing and aggregation, successfully converting values as requested.

The creation of the R Shiny dashboard involved several steps: three for searching for instructions and one for the actual dashboard development. The resulting dashboard, as shown in Figure 7, met the requested criteria, displaying well-formatted and accurate KPIs. However, these KPI's data were inserted into the R script as text by AutoGPT, with only the graph being generated from the data file created by AutoGPT.

In conclusion, AutoGPT demonstrated the ability to write well-structured Python scripts for data processing and handled the more complex data preparation required for the Bitcoin Transaction Hashes dataset. It successfully created R Shiny dashboard containing all required information. However, the primary drawback was AutoGPT's tendency to hardcode data into the scripts, which significantly reduced the reproducibility, reusability, and scalability of the generated code.

Summary for Advanced Structured Prompts The transition from simple natural language prompts to advanced structured prompts demonstrated significant improvements in the quality

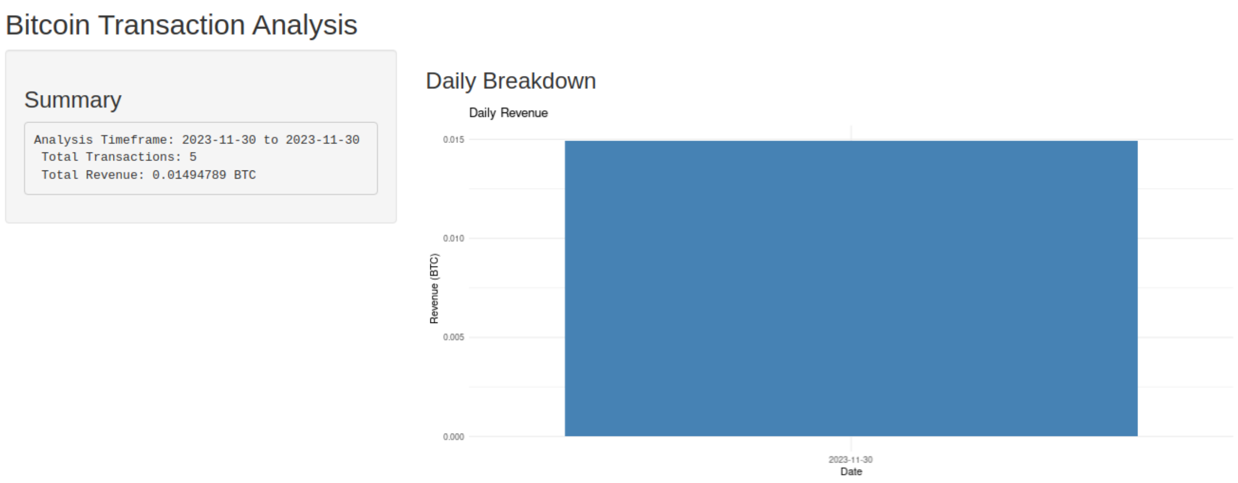


Figure 7. R Shiny Dashboard Created by AutoGPT Using Advanced Structured Prompt for Bitcoin Transaction Hashes

of analysis. AutoGPT was able to write higher-quality data processing scripts and create more consistent and accurate dashboards for each dataset.

On average, tasks with advanced structured prompts took approximately 6.5 minutes to run, with operational costs ranging between 1.11 to 1.87 dollars. Notably, the Apple Health Data experiment required the most steps and API requests, resulting in higher operational costs. This was primarily due to AutoGPT inefficiency by performing and testing each analysis separately. Despite these complexities, AutoGPT autonomously debugged one script.

On the other hand, the Traffic Accidents in Lithuania dataset reduced costs and required the fewest steps and incurred the lowest operational costs of all the datasets. This suggests increased efficiency in task execution for this specific dataset.

However, a recurring limitation across datasets was the lack of reproducible code and the absence of detailed instructions or insights. AutoGPT frequently conducted analyses directly, bypassing the creation of scripts, which impacted the reproducibility and scalability of the solutions. Notably, only the Apple Health Data experiment produced a fully reproducible outcome on the full dataset without any modifications except moving generated images to a folder where R Shiny can load them.

In conclusion, while advanced structured prompts led to more refined and accurate analyses, the experiments highlighted the need for further advancements in AutoGPT's ability to generate reproducible and scalable code, particularly in the context of complex data processing tasks.

4.1.3 Prompts Integrating Both Functional and Non-Functional Requirements

In the third part of the experiment, the focus shifted to evaluating AutoGPT's performance with prompts that integrate both functional and non-functional requirements. Given the increased complexity and specificity of these prompts, they demand even more effort to construct. Thus, the expectation was that such detailed prompts would lead to faster, more cost-effective and higher-quality results compared to the simpler and structured prompt types previously tested. The assumption was that a highly specific prompt, giving clear instructions to the Large Language Model (LLM), would yield outputs that are more precise, accurate and better aligned with the user's expectations. This section examines how these comprehensive prompts were executed and discusses the outcomes of these tests.

Table 2. AutoGPT Results Using Advanced Structured Prompts

Metric		Apple Health Data	Traffic Accidents in Lithuania	Bitcoin Transaction Hashes
Steps to Completion		16	7	13
Total Runtime		5m57,947s	8m14,552s	6m38,277s
API Request Count	GPT-3.5	10	7	8
	GPT-4	16	7	13
	Text Embedding	2	4	10
Token Utilization	GPT-3.5	14530	12916	4225
	GPT-4	57701	29264	48402
	Text Embedding	7884	3332	1801
Operational Cost	GPT-3.5	0.02	0.02	0.01
	GPT-4	1.85	1.09	1.57
	Text Embedding	0	0	0

Quantitative results for each dataset, highlighting the main metrics, are presented in Table 6. Detailed descriptions of each metric can be found in Subsection 2.5.

Apple Health Data AutoGPT created several Python scripts: one for data renaming and saving, two for chart creation and three for data processing and aggregation. Similar to the advanced structured prompts, it generated graphs within the Python scripts and saved them as PNG files. However, as a significant improvement, the R Shiny dashboard this time utilized graphs created in R, not just the PNG images. The generated report is shown in Figure 8.

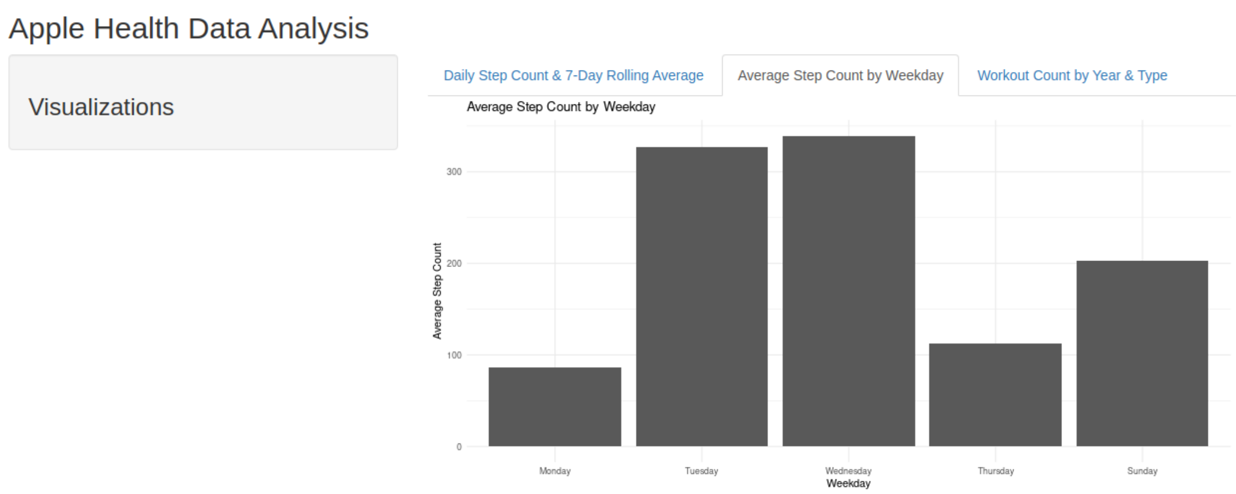


Figure 8. R Shiny Dashboard Created by AutoGPT Using Prompt Integrating Both Functional and Non-Functional Requirements for Apple Health Data

Although AutoGPT could not run the R Shiny dashboard itself, it provided clear instructions for dashboard execution and interpretation of the graphs. This guidance included steps for installing necessary software and packages, running the dashboard and insights based on the visualizations.

Furthermore, the software created by AutoGPT was tested on the full dataset, confirming that analysis is reproducible and could be applied on full dataset.

In assessing the fulfillment of requirements, AutoGPT's performance with the Apple Health Data *Fully Met* 6 out of 8 functional requirements and 2 out of 3 non-functional requirements, as shown in Table 3. This demonstrates a high level of completion, though it fell short in providing insights and user documentation. While the graphs were labeled, well structured in separate tabs, there was room for improvement in their readability.

Table 3. Requirement Fulfillment for Apple Health Data

	Completion Status		
Requirement	Not Met	Partially Met	Fully Met
Functional Requirements			
The system must import and process data from separate .csv files: step_count.csv and workout.csv. Use the creationDate as the date reference for all analyses.			+
All data preparation and analyses should be performed using Python.			+
Calculate daily step count together with 7 days rolling average and create a graph from it where x axis shows date and y axis shows steps count.			+
Calculate daily average step count grouped by the day of the week. Provide bar chart where x axis shows day of the week and y axis shows daily average step count.			+
Count number of workouts grouped by year and type of workout (workoutActivityType column). Provide bar chart where x axis shows year, y axis shows workouts count and bars are split and colored by the type of workout.			+
Create a dashboard / R Shiny app including performed analysis. Add all created graphs to it.			+
Provide 2-4 actionable insights.	+		
Write a user guide and documentation.		+	
Non-Functional Requirements			
Easy to understand graphs and tables. Dashboard should be easy to navigate, intuitive.		+	
Ability to handle large amounts of data efficiently.			+
The project should be developed within a predefined budget and ongoing operational costs should be reasonable.			+

In summary, AutoGPT showed notable progress with prompts integrating both functional and non-functional requirements. It successfully developed Python scripts that efficiently processed data and generated a comprehensive final report, including all requested graphs. A significant advancement was that these scripts were fully reusable with the complete dataset, and all graphs were generated directly in R, accompanied by brief execution instructions from AutoGPT. Furthermore, both the steps required for completion and the overall operational costs were reduced compared to earlier prompts, indicating an improvement in efficiency.

However, despite these advancements, AutoGPT did not produce reliable form of documentation or actionable insights and there was still room for improvement in the readability of the generated graphs. These areas represent potential opportunities for further enhancement in AutoGPT's capabilities for comprehensive data analysis tasks.

Traffic Accidents in Lithuania In this experiment, AutoGPT directly inserted data from the provided file into the Python script, rather than dynamically loading it, thus limiting the reproducibility of the experiment. Modifications to the code would be necessary to apply it to the full dataset. The Python script initially replaced missing values and saved the processed data into a new CSV file.

Subsequently, another Python script was developed to analyze the data, producing a PNG file that depicted monthly trends in traffic accidents, categorized by alcohol-related and non-alcohol-related accidents. AutoGPT also created an R Shiny dashboard, which utilized the files it generated to display a dashboard with two tabs. However, the data representation in the report lacked clarity and readability. The generated report is showcased in Figure 9.

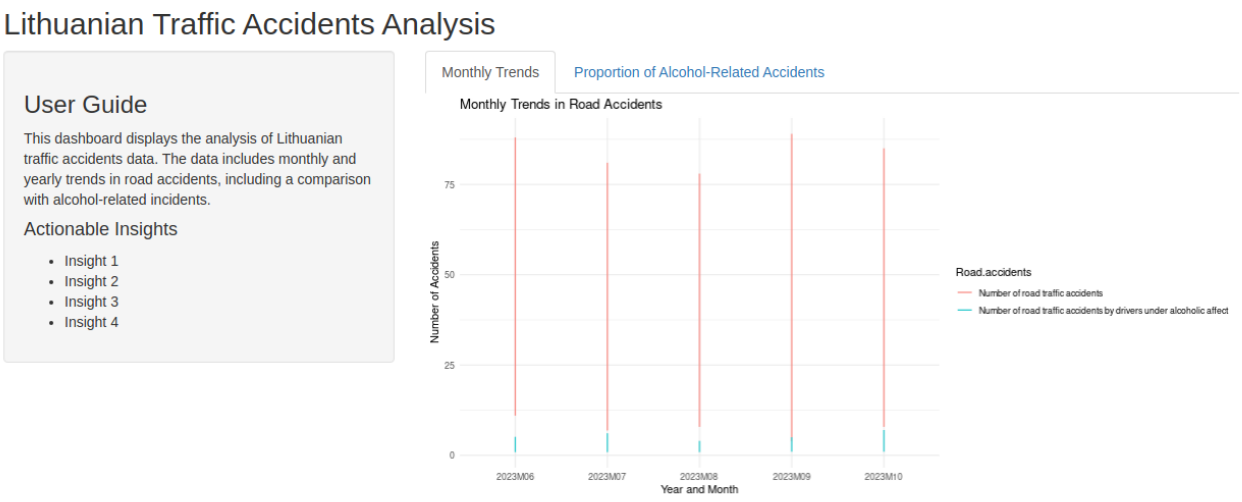


Figure 9. R Shiny Dashboard Created by AutoGPT Using Prompt Integrating Both Functional and Non-Functional Requirements for Traffic Accidents in Lithuania

AutoGPT included a user guide within the report, providing instructions on how to interpret the graphs.

The assessment of requirement fulfillment revealed that AutoGPT's performance with the Traffic Accidents in Lithuania dataset was lacking. As detailed in Table 4, it fully met only 2 out of 8 functional requirements and failed to meet 3, with mixed results for the non-functional requirements. While it successfully created a dashboard, the inserted data within the Python script and the failure to include all requested graphs, along with issues in graph readability, marked significant areas for improvement.

In summary, AutoGPT's handling of the Traffic Accidents in Lithuania dataset with prompt integrating both functional and non-functional requirements demonstrated mixed outcomes. While it successfully processed the data and created an R Shiny dashboard, the approach to data handling in Python scripts was a major limitation, affecting the reproducibility of the analysis. The generated dashboard, although operational, exhibited issues with the clarity and readability of the data presentation. Additionally, AutoGPT's performance in fulfilling the specified requirements was

Table 4. Requirement Fulfillment for Traffic Accidents in Lithuania

	Completion Status		
Requirement	Not Met	Partially Met	Fully Met
Functional Requirements			
The system should be able to import Lithuanian traffic accidents data provided in a single .csv file data_to_read/traffic_accidents_100.csv.			+
All data preparation and analyses should be performed using Python.		+	
Calculate the total number of road traffic accidents every month to see if there's any trend or seasonality. Plot it like a curve where x axis shows year and month, and y axis shows the number of road traffic accidents.	+		
Compare the total number of road traffic accidents to those caused by drivers under the influence of alcohol over time. Plot it like a curve where x axis shows year and month, and y axis shows the number of road traffic accidents split by if accident was caused by driver under the influence of alcohol or not.		+	
Calculate the total number of accidents in each county and the proportion of road traffic accidents by drivers under alcoholic affect in each county. Visualize it using a bar chart where x axis shows county and y axis shows number of accidents colored by if accidents caused by drivers under alcoholic affect.	+		
Create a dashboard / R Shiny app. Add all graphs and tables described in functional requirements.			+
Provide 2-4 insights from analysis performed.	+		
Write a user guide and documentation.		+	
Non-Functional Requirements			
Easy to understand graphs and tables. Dashboard should be easy to navigate, intuitive.	+		
Ability to handle large amounts of data efficiently.		+	
The project should be developed within a predefined budget and ongoing operational costs should be reasonable.			+

suboptimal, as it met only a fraction of the functional requirements and showed varied results for non-functional requirements. The inclusion of a user guide for graph interpretation was a positive aspect, yet the overall execution indicated a need for improvements.

Bitcoin Transaction Hashes In this task, AutoGPT demonstrated its proficiency by writing effective Python scripts for data processing and analysis. The script accurately loaded data from the file, utilized the provided API correctly, and performed the requested transformations.

AutoGPT also succeeded in creating an aesthetically pleasing R Shiny report, which featured well-formatted Key Performance Indicators (KPIs) at the top and the required graph at the bottom. However, the R script was not executable and required minimal user corrections due to an incorrect separator and a missing library.

A notable accomplishment of AutoGPT in this instance was its ability to generate well-structured documentation and real data-based insights, marking a significant improvement over the other prompts.

The generated report is showcased in Figure 10.

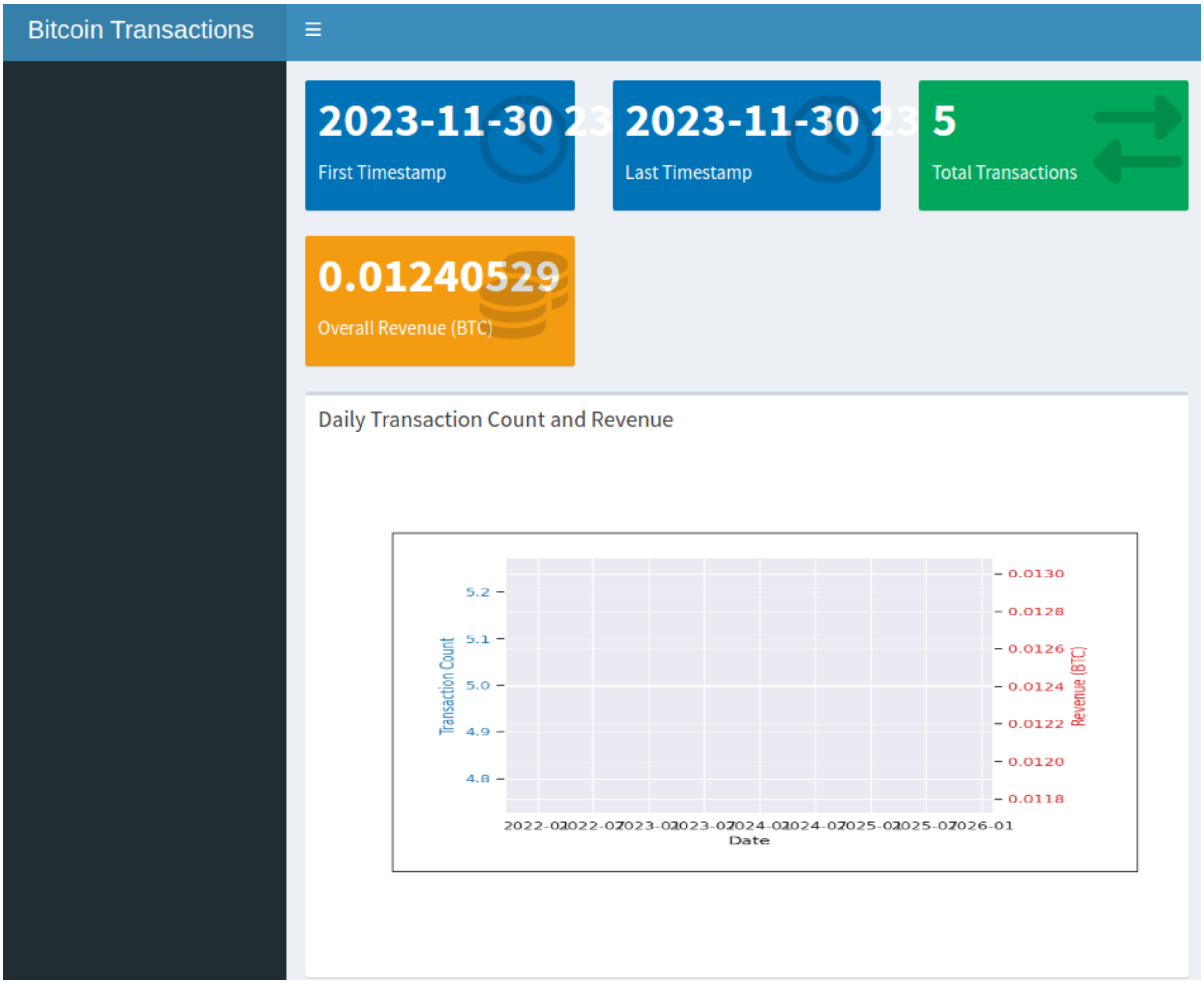


Figure 10. R Shiny Dashboard Created by AutoGPT Using Prompt Integrating Both Functional and Non-Functional Requirements for Bitcoin Transaction Hashes

In assessing the fulfillment of requirements, AutoGPT's performance with the Bitcoin Transaction Hashes dataset *Partially Met* only 1 functional requirement due to the need of minimal correction for R Shiny dashboard to work. Other requirements are treated as *Fully Met* (see Table 5). This demonstrates a high level of completion.

In summary, AutoGPT's handling of the Bitcoin Transaction Hashes dataset with prompt integrating both functional and non-functional requirements was notably effective, especially in the areas of data processing and analysis. The Python scripts developed by AutoGPT were proficient in loading and transforming data and the AI agent's use of the provided API was accurate. A major improvement was observed in the R Shiny report, which was better designed, featuring well-formatted KPIs and a required graph.

Table 5. Requirement Fulfillment for Bitcoin Transaction Hashes

	Completion Status		
Requirement	Not Met	Partially Met	Fully Met
Functional Requirements			
The system must import and process data from data_to_read/bitcoin_transactions_100.csv.			+
All data preparation and analyses should be performed using Python.			+
Extract available information for each Bitcoin transaction hash using https://blockchain.info/rawtx/$tx_hash API, where tx_hash is transaction hash provided in dataset.			+
Transform unix timestamps extracted from the API to human-readable datetime (YYYY-MM-DD hh:mm:ss).			+
Convert transaction values to BTC currency. blockchain.info API provide transaction values in Satoshi, given that 1 BTC is equal to 100 million Satoshis.			+
Analyze data by extracting the first and last timestamps in given dataset, provide in scorecards.			+
Calculate the total count of transactions together with total revenue in BTC and provide in scorecards.			+
Calculate the total count of transactions together with total revenue in BTC daily and provide in a curve chart where x axis represent date and first y axis shows transaction count and second y axis shows revenue.			+
Create a dashboard / R Shiny app including performed analysis. Add all created graphs to it.		+	
Provide 2-4 actionable insights.			+
Write a user guide and documentation.			+
Non-Functional Requirements			
Easy to understand graphs and tables. Dashboard should be easy to navigate, intuitive.			+
Ability to handle large amounts of data efficiently.			+

However, the R Shiny script required minimal user corrections to function correctly, indicating a slight gap in AutoGPT's testing and execution capabilities. Despite this, AutoGPT excelled in providing well-structured documentation and real data-based insights, a significant advancement over its performance with other datasets. This achievement demonstrates AutoGPT's potential in generating comprehensive and insightful analysis, aligning closely with user expectations and requirements. The overall performance showed a high level of requirement fulfillment, marking a substantial step forward in the AI's ability to handle complex data analysis tasks with precision and depth.

Summary In summary, the use of prompts integrating both functional and non-functional requirements marked an advancement in AutoGPT's performance across all datasets. These comprehensive prompts led to more detailed and precise Python scripts for data processing and analysis, indicating a higher level of understanding and execution by AutoGPT. Notably, there was a marked improvement in the creation of R Shiny dashboards, which featured better formatting and more insightful data representations compared to earlier prompts.

With the Apple Health Data, AutoGPT successfully generated multiple Python scripts and an improved R Shiny dashboard, showing an enhanced ability to integrate analyses directly into the dashboard. For the Traffic Accidents in Lithuania dataset, despite facing challenges with direct data insertion in Python scripts, AutoGPT managed to produce a dashboard, although with some issues in data readability. The Bitcoin Transaction Hashes dataset saw the most significant improvement, with AutoGPT not only generating effective scripts and a visually appealing dashboard but also providing valuable data-based insights, setting a new benchmark for its performance.

However, across all datasets, certain limitations persisted. The most notable was the inability to create fully reproducible and scalable code due to the tendency of inserting data into scripts.

Overall, this part of the experiment demonstrated that more specific and detailed prompts lead to more accurate and useful outcomes from AutoGPT, though attention to enhancing code reproducibility and scalability remains a crucial area for future development.

Table 6. AutoGPT Results Using Prompts Integrating Both Functional and Non-Functional Requirements

Metric		Apple Health Data	Traffic Accidents in Lithuania	Bitcoin Transaction Hashes
Steps to Completion		12	6	13
Total Runtime		6m4,523s	5m28,235s	8m52,616s
API Request Count	GPT-3.5	8	3	7
	GPT-4	12	6	13
	Text Embedding	4	2	7
Token Utilization	GPT-3.5	13748	5633	3808
	GPT-4	38779	25979	46331
	Text Embedding	8469	2376	1504
Operational Cost	GPT-3.5	0.02	0.01	0
	GPT-4	1.29	0.74	1.5
	Text Embedding	0	0	0

4.1.4 Prompt Types Comparison

This subsubsection presents a comparative review of AutoGPT's performance across the different types of prompts and datasets. This comparison will highlight how varying levels of prompt specificity and complexity influenced the quality, accuracy and efficiency of AutoGPT's responses and its ability to fulfill the specified requirements.

Among the different prompt types, prompts integrating both functional and non-functional requirements consistently led to the most effective results. They created better quality scripts for data processing and analysis, produced more refined dashboards and even demonstrated potential in generating meaningful insights and documentation.

Table 7. Requirement Fulfillment Analysis

	Completion Status Count		
Dataset	Not Met	Partially Met	Fully Met
Functional Requirements			
Apple Health Data	1	1	6
Traffic Accidents in Lithuania	3	3	2
Bitcoin Transaction Hashes	0	1	10
Non-Functional Requirements			
Apple Health Data	0	1	2
Traffic Accidents in Lithuania	1	1	1
Bitcoin Transaction Hashes	0	0	2

For the Traffic Accidents in Lithuania dataset, a consistent trend was observed: it utilized the least amount of OpenAI's API resources across all prompt types (see Table 8). This trend highlights the efficiency of AutoGPT in handling this particular dataset, regardless of the prompt complexity. Also, When comparing simple natural language prompt with a prompt integrating both functional and non-functional requiremnets operational costs decreased almost twice: from 1.5 to 0.75 dollars.

In the case of the Apple Health Data, not only did the quality of results improve with the most complex prompts, but there was also a significant decrease in operational costs. This improvement underscores the effectiveness of highly specific prompts in guiding AutoGPT to deliver more accurate and cost-efficient outcomes.

Notably, in the case of the Bitcoin Transaction Hashes dataset, the operational costs did not significantly fluctuate across the different types of prompts. This consistency suggests that the complexity of the prompts did not significantly impact the resource utilization for this particular dataset, indicating a level of stability in AutoGPT's performance regardless of prompt specificity.

The trend of decreasing API request counts with increasing prompt specificity and complexity was consistent across all datasets (see Table 9). This trend suggests that more specific prompts enable AutoGPT to execute tasks with greater precision and less resource utilization.

This decrease is more visible in GPT-3.5 API requests, suggesting its higher sensitivity to prompt specificity. In contrast, while GPT-4 also showed a decreasing trend, the reduction in API requests was not as significant. This difference may indicate variations in how AutoGPT differentiate GPT model selection based on prompt complexity and specificity.

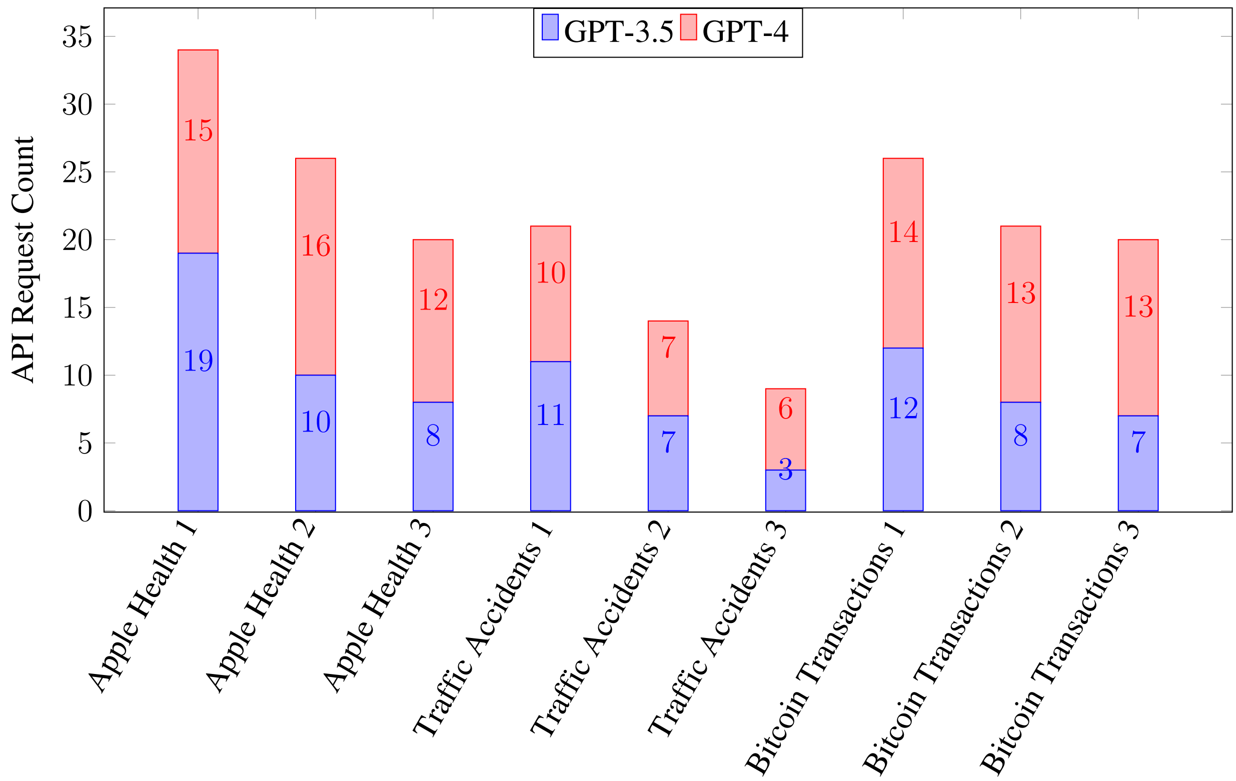
Overall, this comparative analysis across different datasets and prompt types shows a general trend: as the complexity of prompts increases, so does the efficiency and quality of the outputs produced by AutoGPT. However, the relatively stable operational costs for the Bitcoin Transaction Hashes dataset across different prompts, suggest that the influence of prompt complexity might differ depending on the nature of the dataset. Such insights show the importance of adjusting prompts to the specific characteristics and requirements of the dataset to optimize the performance of AI tools like AutoGPT in software development for data analysis tasks.

Given that creating prompts with formal requirements can demand a significant investment of time, it becomes crucial to balance this effort against the potential improvements in output quality. When compared to advanced structured prompts, the decision to use more complex prompts should be carefully considered, ensuring that the improvement in output quality justifies the ad-

Table 8. Results Aggregated (Prompt 1 - Simple Natural Language Prompts; Prompt 2 - Advanced Structured Prompts; Prompt 3 - Prompts Integrating Both Functional and Non-Functional Requirements

Dataset Name	Metric	Prompt 1	Prompt 2	Prompt 3
Apple Health Data	Steps to Completion	16	16	12
	Total Runtime	7m3,466s	5m57,947s	6m4,523s
	API Request Count	40	28	24
	Token Utilization	99271	80115	60996
	Operational Cost	1.69	1.87	1.31
Traffic Accidents in Lithuania	Steps to Completion	6	7	6
	Total Runtime	6m36,543s	8m14,552s	5m28,235s
	API Request Count	29	18	11
	Token Utilization	70192	45512	33988
	Operational Cost	1.4	1.11	0.75
Bitcoin Transaction Hashes	Steps to Completion	13	13	13
	Total Runtime	6m43,710s	6m38,277s	8m52,616s
	API Request Count	36	31	27
	Token Utilization	69803	54428	51643
	Operational Cost	1.5	1.58	1.5

Table 9. AutoGPT API Request Count Grouped by Dataset, Prompt and GPT Model Used (Number by dataset name indicates prompt number: Prompt 1 - Simple Natural Language Prompts; Prompt 2 - Advanced Structured Prompts; Prompt 3 - Prompts Integrating Both Functional and Non-Functional Requirements



ditional time and effort required. These insights emphasize the need for a strategic approach in prompt construction, balancing complexity with practicality, to use the full potential of LLM in data analysis.

4.2 Limitations and Challenges

This subsection delves into the various limitations and challenges encountered throughout the experiments with AutoGPT. Recognizing these limitations is significant for understanding the current boundaries of LLM capabilities in software development for data analysis tasks and for guiding future improvements and research directions. The following paragraphs will describe key areas where AutoGPT faced difficulties or where the experimental design could be refined.

Technical Limitations of AutoGPT In our experiments, AutoGPT faced a range of technical limitations that impacted its overall performance. One significant challenge was the context window limitation, which occasionally led to breakdowns in processing, especially when dealing with large datasets or loading extensive pages. This limitation reflects a need for more efficient data handling capabilities in AI systems. Additionally, AutoGPT sometimes struggled with understanding and efficiently executing complex prompts, leading to incomplete or inaccurate task fulfillment. Another notable limitation was AutoGPT's inability to generate comprehensive documentation and actionable insights. This shortfall indicates a gap in the AI's capability to not only process data but also to meaningfully interpret and communicate complex analytical tasks. Additionally, AutoGPT was unable to test and run R scripts, despite being deployed in a custom Docker image with R installed. This issue points to a limitation in its scripting capabilities. There were also instances where AutoGPT became stuck in repetitive loops, working on the same task without yielding productive results, which highlights a need for better task management and execution strategies within the AI's operational framework.

Data Handling and Reproducibility Throughout our experimentation, data handling and script reproducibility emerged as significant challenges. AutoGPT frequently attempted to execute analyses directly and insert data into scripts rather than creating reusable scripts. While this direct approach might yield immediate results, it significantly reduces the reproducibility and scalability of the solutions. This tendency towards inefficient data handling points to a broader issue within the current AI agents capabilities. It underscores the need for more sophisticated methodologies that can generate versatile and robust scripts, enabling more dynamic data handling. Enhancing these capabilities is critical for ensuring that AI solutions can be effectively scaled and adapted to various data analysis scenarios, meeting the diverse needs of real-world applications.

Operational Costs and Efficiency Throughout the experiments with AutoGPT, it was observed a consistent trend that the API request count decreased as the specificity and complexity of the prompts increased. This pattern indicates that more detailed prompts lead to more precise task execution by AutoGPT, ultimately resulting in lower resource utilization and potentially more cost-effective operations. However, the use of GPT-4 API, which is relatively expensive, posed certain limitations. Inefficiencies such as the need for debugging and addressing repetitive tasks can increase the operational costs. Consequently, due to budgetary constraints, it was not possible to run the experiments on full datasets, more datasets or in continuous mode. At the beginning of this thesis, the options for running Large Language Models (LLMs) locally were limited, necessitating reliance on costly LLM APIs. This reliance on external APIs introduces significant considerations for the practical deployment of AI technologies like AutoGPT in data analysis tasks, especially in resource-constrained environments. Finding a balance between the complexity of tasks and operational efficiency remains a critical challenge for the deployment of AI agents like AutoGPT.

Sensitivity to Prompt Structure and Order A notable limitation observed in the experiments with AutoGPT was its significant sensitivity to the structure and order of words in the prompts. This characteristic has critical implications for how prompts should be constructed to guide AutoGPT effectively. It was discovered that properties or tasks meant to be universally applicable in an AutoGPT task need to be stated at the beginning of the prompt. AutoGPT tends to create tasks in the order presented in the prompt. For example, if a file name is mentioned towards the end of the prompt, AutoGPT is likely to interpret it as an instruction to load data at a later stage in the task sequence. This behavior underscores the need for careful and strategic prompt design, ensuring that the most critical elements are positioned early in the prompt to direct AutoGPT's task prioritization and execution order appropriately.

Summary and Future Research Directions To summarize, our experiments with AutoGPT highlighted several key limitations and challenges, ranging from technical limitations to efficiency concerns. These findings pave the way for future research directions focused on enhancing AI agent's understanding of complex tasks, improving code reproducibility and scalability and optimizing resource utilization. Future improvements in AI algorithms could address these limitations, leading to better solutions for automated software development for data analysis. Continued exploration in this field is essential to unlock the full potential of AI agents in automating and streamlining complex analytical processes.

Results and Conclusions

This section presents the results and draws conclusions from comprehensive experiments with AutoGPT, conducted across various datasets and using three different prompt types. Central to this thesis is the exploration of the impact that integrating functional and non-functional requirements into prompt construction has on the automation of software development for data analysis.

Extensive literature review highlighted a growing trend: the increasing role of artificial intelligence and machine learning in data analytics, with Large Language Models (LLMs) playing a pivotal role. Despite this trend, there exists a gap in academic research, particularly regarding the capabilities of AI agents in software development for data analysis and the influence of formal requirements on LLM results.

To address this gap, three diverse datasets were carefully selected and robust testing protocols were established. This setup allowed us to conduct a detailed comparative analysis of different prompt complexities on distinct datasets. The experiment was structured to progressively assess AutoGPT's capabilities, starting from simple natural language prompts, advancing to more structured prompts and finally constructing prompts that integrated both functional and non-functional requirements.

Results

1. Performed extensive experiments on testing AutoGPT capabilities in automation of software development for data analysis. AutoGPT was able to generate well-written Python scripts for data processing and create informative R Shiny dashboards with visual data representations. Notably, for one dataset, AutoGPT even produced actionable insights along with documentation.

2. The operational efficiency and associated costs of using AutoGPT were influenced by the complexity of the prompts. More sophisticated prompts resulted in fewer API requests, implying greater task execution efficiency. Comparing simple natural language prompts with those integrating formal requirements, the average operational cost decreased by $0.34 (from $1.53 to $1.19). However, the cost associated with OpenAI's API calls remained a limiting factor, influencing the scope of the experiments.

3. Sensitivity to the structure and wording of prompts has been observed in AutoGPT. This aspect is crucial for prompt construction, as it significantly impacts how AutoGPT interprets and executes tasks.

4. The experiments revealed AutoGPT's effectiveness in meeting formal requirements with prompts tailored for this purpose:

    - For Apple Health Data, AutoGPT fully met 6 out of 8 functional requirements and 2 out of 3 non-functional requirements.
    - With Traffic Accidents in Lithuania, it met 2 out of 8 functional requirements and 1 out of 3 non-functional requirements.
    - For Bitcoin Transaction Hashes, it successfully fulfilled 10 out of 11 functional requirements and both non-functional requirements.

Conclusions

1. Our experiments demonstrated AutoGPT's capability to perform basic data analysis tasks. It successfully wrote scripts for data processing, extracted data from APIs and presented the results in R Shiny dashboards. Additionally, AutoGPT showed potential in generating actionable insights and creating documentation, marking a significant step in AI-assisted data analysis.

2. The effort involved in constructing prompts with formal requirements must be weighed against the expected improvement in output quality. Complex prompts can enhance output precision but require more time and effort to create. This necessitates a strategic approach in prompt construction, optimizing the balance between complexity and practical utility to fully leverage LLM capabilities in data analysis.

3. Strategic prompt design is crucial, with the placement of critical elements early in the prompt playing a key role in directing AutoGPT's task execution. This approach ensures that important tasks are prioritized and executed first.

4. While not all formal requirements were fully met in our experiments, the inclusion of these requirements could significantly improve the efficiency and quality of AutoGPT's results, showing their value in prompt construction.

AutoGPT demonstrates potential as an autonomous AI agent, capable of improving efficiency or performing certain tasks. However the tasks it handled were relatively simple and within the capabilities of a junior data analyst. Although AutoGPT completed these tasks, its current capabilities are not yet suited for production-level solutions. Crafting effective prompts for AutoGPT, even for simple tasks, can be time-consuming. Additionally, the cost-effectiveness of using the GPT-4 API compared to hiring a data analyst needs careful consideration. While AutoGPT shows promise in certain areas, its role as a replacement for human analysts is still limited.

References

[1] Mohamad Almgerbi, Andrea De Mauro, Adham Kahlawi, and Valentina Poggioni. A systematic review of data analytics job requirements and online-courses. *Journal of Computer Information Systems*, 62:1--13, 09 2021.

[2] Amos Azaria, Rina Azoulay, and Shulamit Reches. Chatgpt is a remarkable tool -- for experts, 2023.

[3] Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, Quyet V. Do, Yan Xu, and Pascale Fung. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity, 2023.

[4] Kornelia Batko and Ślęzak Andrzej. The use of big data analytics in healthcare. *Journal of Big Data*, 9, 01 2022.

[5] Y. Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 5:157--66, 02 1994.

[6] Sridevi Bonthu and Hima Bindu. Review of leading data analytics tools, 08 2018.

[7] Vadim Borisov, Kathrin Seßler, Tobias Leemann, Martin Pawelczyk, and Gjergji Kasneci. Language models are realistic tabular data generators, 2023.

[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[10] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179--211, 1990.

[11] Tyna Eloundou, Sam Manning, Pamela Mishkin, and Daniel Rock. Gpts are gpts: An early look at the labor market impact potential of large language models, 2023.

[12] Pablo García-Risueño and Pablo Ibáñez. A review of high performance computing foundations for scientists. *International Journal of Modern Physics C*, 23, 07 2012.

[13] Christopher Gerling and Stefan Lessmann. Multimodal document analytics for banking process automation, 2023.

[14] GitHub. Github copilot, 2021. Accessed: 2023-11-11.

[15] Significant Gravitas. Autogpt, 2023. Accessed: 2023-11-04.

[16] Ken Gu, Madeleine Grunde-McLaughlin, Andrew M. McNutt, Jeffrey Heer, and Tim Althoff. How do data analysts respond to ai assistance? a wizard-of-oz study, 2023.

[17] Ishu Gupta and Ashutosh Kumar Singh. A holistic view on data protection for sharing, communicating, and computing environments: Taxonomy and future directions, 2022.

[18] Shishir Gupta, Ahmad Mahmoud, and Michael Massoomi. A clinician's guide to smartwatch "interrogation". *Current Cardiology Reports*, 24, 08 2022.

[19] Erik E. Guzik, Christian Byrge, and Christian Gilde. The originality of machines: Ai takes the torrance test. *Journal of Creativity*, 33(3):100065, 2023.

[20] Andreas Heinrich and Eduard Klein. Challenges for the management of qualitative and quantitative data: The example of social policy–related data collections. *Global Social Policy*, 21(1):138--143, 2021.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735--1780, 11 1997.

[22] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, and Chenglin Wu. Metagpt: Meta programming for multi-agent collaborative framework, 2023.

[23] Azham Hussain, Emmanuel Mkpojiogu, and Fazillah Kamal. The role of requirements in the success or failure of software projects. *EJ Econjournals*, 6:6--7, 10 2016.

[24] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12), mar 2023.

[25] R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181--184 vol.1, May 1995.

[26] Eugenijus Kurilovas. On data-driven decision-making for quality education. *Computers in Human Behavior*, 107:105774, 2020.

[27] Lithuanian State Data Agency. Road accidents in lithuania: January 2005 to october 2023. https://osp.stat.gov.lt/en/statistiniu-rodikliu-analize?hash=12fb56bf-21a6-4901-b31d-88a84f1ef9f1, 2023. Accessed: 2023-12-12.

[28] Chang Liu and Bo Wu. Evaluating large language models on graphs: Performance insights and comparative analysis, 2023.

[29] Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, Zihao Wu, Lin Zhao, Dajiang Zhu, Xiang Li, Ning Qiang, Dingang Shen, Tianming Liu, and Bao Ge. Summary of chatgpt-related research and perspective towards the future of large language models. *Meta-Radiology*, 1(2):100017, 2023.

[30] Mehrdad Maghsoudi and Navid Nezafati. Navigating the acceptance of implementing business intelligence in organizations: A system dynamics approach. *Telematics and Informatics Reports*, 11:100070, sep 2023.

[31] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1999.

[32] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. *Proceedings of the 11th Annual Conference of the International Speech Communication Association, INTERSPEECH 2010*, 2:1045--1048, 01 2010.

[33] Lincoln Murr, Morgan Grainger, and David Gao. Testing llms on code generation with varying levels of prompt specificity, 2023.

[34] Yohei Nakajima. Babyagi, 2023. Accessed: 2023-11-04.

[35] OpenAI. Gpt-4 technical report, 2023.

[36] F. Provost and T. Fawcett. *Data Science for Business*. O'Reilly, 2013.

[37] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257--286, Feb 1989.

[38] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training, 2018.

[39] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.

[40] Iqbal Sarker. Data science and analytics: An overview from data-driven smart computing, decision-making and applications perspective. *SN Computer Science*, 2, 07 2021.

[41] Iqbal Sarker. Data science and analytics: An overview from data-driven smart computing, decision-making and applications perspective. *SN Computer Science*, 2, 07 2021.

[42] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. An empirical evaluation of using large language models for automated unit test generation, 2023.

[43] Bo Shen, Jiaxin Zhang, Taihong Chen, Daoguang Zan, Bing Geng, An Fu, Muhan Zeng, Ailun Yu, Jichuan Ji, Jingyang Zhao, Yuenan Guo, and Qianxiang Wang. Pangu-coder2: Boosting large language models for code with ranking feedback, 2023.

[44] Chenglei Si, Zhe Gan, Zhengyuan Yang, Shuohang Wang, Jianfeng Wang, Jordan Boyd-Graber, and Lijuan Wang. Prompting gpt-3 to be reliable, 2023.

[45] Gökhan Silahtaroğlu and Nevin Yılmaztürk. Data analysis in health and big data: A machine learning medical diagnosis model based on patients' complaints. *Communications in Statistics - Theory and Methods*, 50(7):1547--1556, 2021.

[46] Sanjeevan Sivapalan, Alireza Sadeghian, Hossein Rahanam, and Asad Madni. Recommender systems in e-commerce, 08 2014.

[47] Statista. Worldwide data created 2010-2025, 2021. Accessed on 2023-10-15.

[48] Harald Steck, Linas Baltrunas, Ehtsham Elahi, Dawen Liang, Yves Raimond, and Justin Basilico. Deep learning for recommender systems: A netflix case study. *AI Magazine*, 42(3):7--18, Nov. 2021.

[49] The Editors of Encyclopaedia Britannica. William lee. Encyclopedia Britannica, 8 2023. Accessed 5 November 2023.

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[51] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, jan 1966.

[52] Danning Xie, Byungwoo Yoo, Nan Jiang, Mijung Kim, Lin Tan, Xiangyu Zhang, and Judy S. Lee. Impact of large language models on generating software specifications, 2023.

[53] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. Large language models as data preprocessors, 2023.

[54] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. Large language models as data preprocessors, 2023.

[55] Jin Zhang, Taowen Le, and Jianyao Chen. Investigation of essential skills for data analysts: An analysis based on linkedin. *Journal of Global Information Management*, 31:1--21, 07 2023.

Appendices

A Bitcoin Transaction Hashes Query

Listing 1. SQL query for "Bitcoin Transaction Hashes" extraction from Bigqeuery

```
SELECT transactions.hash
FROM `bigquery-public-data.crypto_bitcoin.transactions` AS
    transactions
WHERE transactions.block_timestamp_month = "2023-11-01"
ORDER BY transactions.block_timestamp DESC
LIMIT 1000
```

B AutoGPT Example

Listing 2. Example terminal message for initial user's input for AutoGPT

```
Welcome to Auto-GPT!  run with '--help' for more information
    .
Create an AI-Assistant:  input '--manual' to enter manual
    mode.
  Asking user via keyboard...
I want Auto-GPT to: I want Auto-GPT to: Perform data
    analysis on localy available .csv file called duomenys.
    csv.
NOTE:All files/directories created by this agent can be
    found inside its workspace at:  /app/auto_gpt_workspace
DataGPT  has been created with the following details:
Name:  DataGPT
Role:  an autonomous data analysis AI that specializes in
    performing advanced data analysis on locally available .
    csv files to provide valuable insights and help make data
    -driven decisions.
Goals:
-  Utilize advanced data analysis techniques to extract
    meaningful insights from the duomenys.csv file and
    present them in a clear and understandable manner.
-  Identify patterns, trends, and correlations within the
    data to uncover hidden relationships and provide
    actionable recommendations based on the analysis.
-  Perform thorough data cleansing and preprocessing to
    ensure the accuracy and reliability of the analysis
    results.
```

```
-  Offer customizable analysis options, allowing the user to
     specify the desired analysis techniques, such as
    descriptive statistics, data visualization, regression
    analysis, or clustering.
-  Provide efficient and timely analysis results, ensuring
    that the user can make informed decisions quickly based
    on the insights gained from the duomenys.csv file.
Using memory of type:  JSONFileMemory
Using Browser:  chrome
DATAGPT THOUGHTS:  I need to read the duomenys.csv file to
    start analyzing the data.
REASONING:  Reading the file is the first step in the data
    analysis process. Once I have the file contents, I can
    proceed with data cleansing, preprocessing, and analysis.
PLAN:
-  Read duomenys.csv
-  Clean and preprocess data
-  Perform advanced data analysis
-  Provide insights and recommendations
CRITICISM:  I should be cautious not to make assumptions
    about the data before reading and analyzing it.
SPEAK:  I will start by reading the duomenys.csv file.

NEXT ACTION:  COMMAND = read_file ARGUMENTS = {'filename': '
    duomenys.csv'}
  Enter 'y' to authorise command, 'y -N' to run N continuous
       commands, 'n' to exit program, or enter feedback for
     DataGPT...
  Asking user via keyboard...
Input:
```

C Dataset Schemas

This appendix provides detailed schemas for all datasets utilized in this study. Each schema includes comprehensive information like column names, data types and descriptions of fields within the datasets. The purpose is to offer a clear understanding of the datasets used in the experiments.

C.1 Apple Health Data

This subsection presents the schema for the Apple Health Data used in the study, it consists from two tables: step count and workout.

C.1.1 Step Count

This part describes the schema for the Step Count dataset extracted from Apple Health.

Table 10. Schema for Apple Health table - *step_count.csv*

Field name	Type	Description
sourceName	String	The name of the data source, typically the owner's device.
sourceVersion	String	The version of the software from which data is collected.
device	String	A string representation of the device including identifier, name, manufacturer, model, hardware, and software details.
unit	String	The measurement unit for the recorded data, typically 'count'.
creationDate	DateTime	The date and time when the data record was created, including timezone.
startDate	DateTime	The starting date and time for the data measurement period, including timezone.
endDate	DateTime	The ending date and time for the data measurement period, including timezone.
value	Integer	The numerical value recorded or measured for the data.

C.1.2 Workout

This part describes the schema for the Workout dataset extracted from Apple Health.

Table 11. Schema for Apple Health table - *workout.csv*

Field name	Type	Description
workoutActivityType	String	The type of workout activity, e.g., Cycling, Rowing, Strength Training.
duration	Float	The duration of the workout activity.
durationUnit	String	The unit of measurement for the duration, typically 'min' for minutes.
sourceName	String	The name of the data source, typically the owner's device.
sourceVersion	String	The version of the software from which data is collected.
creationDate	DateTime	The date and time when the data record was created, including timezone.
startDate	DateTime	The starting date and time for the data measurement period, including timezone.
endDate	DateTime	The ending date and time for the data measurement period, including timezone.

C.2 Traffic Accidents in Lithuania

This part describes the schema for the Traffic Accidents in Lithuania dataset.

Table 12. Schema for Traffic Accidents in Lithuania dataset

Field name	Type	Description
Time	String	The time period when the data was recorded in year and month format (YYYYMM).
Indicator	String	A description of the indicator being measured, such as the number of road accidents.
Administrative territory	String	The name of the administrative region or county where the data applies.
Road accidents	String	The category of road accidents being reported, such as accidents involving injuries or those involving alcohol.
Unit	String	The unit of measurement for the data, typically 'units'.
Value	Integer	The numerical value representing the count of road accidents; may be empty if data is not available.

C.3 Bitcoin Transaction Hashes

This part describes the schema for the Bitcoin Transaction Hashes dataset.

Table 13. Schema for Bitcoin Transaction Hashes dataset

Field name	Type	Description
hash	String	A unique identifier for a Bitcoin transaction, typically a 64-character hexadecimal string.

D AutoGPT Generated Goals

This appendix presents initial information generated by AutoGPT from natural language prompts for each dataset and prompt type. It includes AI agent's name, role and goals.

D.1 Apple Health Data

D.1.1 Simple Natural Language Prompts

Listing 3. Goals Generated by AutoGPT Using Simple Natural Language Prompt for Apple Health Data.

```
INFO HealthDataAnalyzerGPT has been created with the
   following details:
INFO Name: HealthDataAnalyzerGPT
INFO Role: an AI assistant specialized in analyzing health
   data using Python and presenting the findings in an R
   Shiny report. It helps users make sense of their step
   count and workout frequency data from Apple Health,
   providing valuable insights for better health and fitness
    management.
```

```
INFO Goals:
INFO - Develop a Python script to read and analyze step
   count and workout frequency data from the respective .csv
    files in the workspace.
INFO - Extract meaningful insights from the data, such as
   average step count per day, weekly workout frequency, and
    trends over time.
INFO - Generate an R Shiny report that presents the findings
    in a visually appealing and interactive manner.
INFO - Ensure the accuracy and reliability of the analysis
   by implementing robust data validation and error handling
    mechanisms.
INFO - Provide clear and concise documentation on how to use
    the Python script and interpret the findings in the R
   Shiny report.
```

D.1.2 Advanced Structured Prompts

Listing 4. Goals Generated by AutoGPT Using Advanced Structured Prompt for Apple Health Data.

```
INFO DataAnalysisGPT has been created with the following
   details:
INFO Name: DataAnalysisGPT
INFO Role: an AI data analyst that specializes in analyzing
   health data using Python and R Shiny. It assists in
   performing data preparation, analysis, and report
   generation for health-related datasets.
INFO Goals:
INFO - Utilize Python to read and analyze the 'workout_100.
   csv' and 'step_count_100.csv' files, using the '
   creationDate' as the date reference for all analyses.
INFO - Compute the daily step count and its 7-day rolling
   average, and visualize these trends in a graph.
INFO - Summarize the average step counts by day of the week
   in a bar chart.
INFO - Count workouts annually by type (workoutActivityType)
    and present the results in a bar chart.
INFO - Compile a final report using R Shiny, incorporating
   the analyzed data and visualizations to provide a
   comprehensive overview of the health data analysis.
```

D.1.3 Prompts Integrating Both Functional and Non-Functional Requirements

Listing 5. Goals Generated by AutoGPT Using Prompt Integrating Both Functional and Non-Functional Requirements for Apple Health Data.

```
INFO DataAnalysisGPT has been created with the following details:
INFO Name: DataAnalysisGPT
INFO Role: an AI data analyst that specializes in analyzing Apple Health Data using Python. It provides comprehensive insights into step count and workouts, with a focus on efficient data handling and cost-effective operations. The AI also creates an intuitive R Shiny dashboard for easy navigation and includes a user-friendly guide and documentation.
INFO Goals:
INFO - Perform efficient data handling and analysis of Apple Health Data from 'data_to_read/workout_100.csv' and 'data_to_read/step_count_100.csv' using Python, with 'creationDate' as the date reference.
INFO - Calculate the daily step count and 7-day rolling average, and present it in a graph with dates on the x-axis and step counts on the y-axis.
INFO - Summarize the daily average step count by weekday in a bar chart.
INFO - Count workouts by year and type, and present the data in a bar chart with years on the x-axis and counts on the y-axis, differentiated by workout type (workoutActivityType).
INFO - Create an intuitive and easy-to-navigate R Shiny dashboard to display the analyses and provide 2-4 actionable insights.
INFO - Ensure efficient data handling, even with large datasets, while staying within budget and maintaining reasonable operational costs.
INFO - Include a user-friendly guide and documentation to assist users in utilizing the AI's capabilities effectively.
```

D.2 Traffic Accidents in Lithuania

D.2.1 Simple Natural Language Prompts

Listing 6. Goals Generated by AutoGPT Using Simple Natural Language Prompt for Traffic Accidents in Lithuania.

```
INFO TrafficAnalyzerGPT has been created with the following details:
```

```
INFO Name: TrafficAnalyzerGPT
INFO Role: an autonomous agent specialized in analyzing traffic accidents data and providing insights on monthly trends and distribution between counties in Lithuania. It is proficient in Python and can generate an R Shiny report to present the findings.
INFO Goals:
INFO - Analyze the traffic accidents data from 'data_to_read/traffic_accidents_100.csv' to identify monthly trends in road accidents in Lithuania.
INFO - Determine the distribution of traffic accidents between counties in Lithuania using the provided data.
INFO - Investigate the impact of alcohol on traffic accidents and analyze its correlation with accident severity.
INFO - Utilize Python to perform data analysis and generate meaningful insights from the traffic accidents data.
INFO - Create an R Shiny report to present the findings of the analysis in an interactive and visually appealing manner.
```

D.2.2 Advanced Structured Prompts

Listing 7. Goals Generated by AutoGPT Using Advanced Structured Prompt for Traffic Accidents in Lithuania.

```
INFO TrafficAnalyzerGPT has been created with the following details:
INFO Name: TrafficAnalyzerGPT
INFO Role: an AI data analyst that specializes in analyzing traffic accidents using Python and R Shiny. It provides comprehensive insights and visualizations to help understand the patterns and trends in traffic accidents in Lithuania.
INFO Goals:
INFO - Perform data preparation and analysis using Python to transform the data from the 'data_to_read/traffic_accidents_100.csv' file into a wide format, using the "Administrative territory" and "Road accidents" columns.
INFO - Compile a final report using R Shiny, incorporating the Python analysis results, to provide a comprehensive overview of the traffic accidents in Lithuania.
INFO - Calculate the monthly accident count and compare accidents involving alcohol, displaying the results in a curve graph to visualize the trends over time.
```

```
INFO - Calculate the total number of accidents in each
    county and the proportion of road traffic accidents by
    drivers under alcoholic affect in each county, and
    present the results in bar charts for easy comparison.
INFO - Provide key insights from the analysis to help
    understand the patterns, trends, and potential factors
    contributing to traffic accidents in Lithuania.
```

D.2.3 Prompts Integrating Both Functional and Non-Functional Requirements

Listing 8. Goals Generated by AutoGPT Using Prompt Integrating Both Functional and Non-Functional Requirements for Traffic Accidents in Lithuania.

```
INFO DataAnalysisGPT has been created with the following
    details:
INFO Name: DataAnalysisGPT
INFO Role: an autonomous agent specialized in data analysis
    and visualization using Python and R. It assists in
    analyzing and interpreting large datasets, providing
    actionable insights and creating intuitive dashboards for
     efficient data exploration.
INFO Goals:
INFO - Analyze the Lithuanian traffic accidents data from '
    data_to_read/traffic_accidents_100.csv' using Python,
    focusing on identifying monthly and yearly trends in road
     accidents, including a comparison with alcohol-related
    incidents.
INFO - Present the identified trends in curve graphs, with
    the x-axis showing year and month and the y-axis showing
    the number of accidents.
INFO - Analyze and display in a curve the total number of
    accidents split by alcohol-related and non-alcohol-
    related incidents.
INFO - Calculate the total number of accidents in each
    county and the proportion of road traffic accidents by
    drivers under alcoholic affect in each county.
INFO - Create an intuitive R Shiny dashboard to display the
    analyses, ensuring efficient handling of large datasets
    and staying within budget with reasonable operational
    costs.
INFO - Include a user guide and documentation with 2-4
    actionable insights derived from the data.
```

D.3 Bitcoin Transaction Hashes

D.3.1 Simple Natural Language Prompts

Listing 9. Goals Generated by AutoGPT Using Simple Natural Language Prompt for Bitcoin Transaction Hashes.

```
INFO BitcoinAnalyzerGPT has been created with the following
    details:
INFO Name: BitcoinAnalyzerGPT
INFO Role: an AI data analyst that specializes in analyzing
    Bitcoin transaction data to provide valuable insights and
     trends. It utilizes the blockchain.info API to extract
    details such as timestamps and transaction values, and
    presents the findings in a comprehensive R Shiny report.
INFO Goals:
INFO - Analyze the Bitcoin transaction data from the '
    data_to_read/bitcoin_transactions_100.csv' file using the
     blockchain.info API.
INFO - Summarize the analysis timeframe, including the start
     and end dates of the data.
INFO - Identify and analyze the main trends in the Bitcoin
    transaction data, such as transaction volume, transaction
     values, and transaction frequency.
INFO - Analyze the daily transaction activities, including
    the number of transactions per day, average transaction
    values, and any significant fluctuations or patterns.
INFO - Present the findings in a visually appealing and
    informative R Shiny report for easy interpretation and
    decision-making.
```

D.3.2 Advanced Structured Prompts

Listing 10. Goals Generated by AutoGPT Using Advanced Structured Prompt for Bitcoin Transaction Hashes.

```
INFO CryptoAnalyzerGPT has been created with the following
    details:
INFO Name: CryptoAnalyzerGPT
INFO Role: an autonomous agent specialized in analyzing
    cryptocurrency transaction data using Python and
    extracting information from the blockchain.info API. It
    provides a comprehensive analysis of Bitcoin transaction
    data, including timestamps and values, and presents the
    results in an R Shiny dashboard.
INFO Goals:
```

```
INFO - Utilize Python to analyze the Bitcoin transaction data from 'data_to_read/bitcoin_transactions_100.csv' and extract transaction information using the blockchain.info API.
INFO - Convert the timestamps from Unix format to YYYY-MM-DD hh:mm:ss and the values from Satoshi to BTC.
INFO - Provide a basic summary of the analysis timeframe, total transactions, and revenue in BTC.
INFO - Generate a daily breakdown of transactions and revenue.
INFO - Present the analyzed data in an R Shiny dashboard for easy visualization and exploration.
```

D.3.3 Prompts Integrating Both Functional and Non-Functional Requirements

Listing 11. Goals Generated by AutoGPT Using Prompt Integrating Both Functional and Non-Functional Requirements for Bitcoin Transaction Hashes.

```
INFO DataProcessingGPT has been created with the following details:
INFO Name: DataProcessingGPT
INFO Role: an autonomous agent specialized in data processing and analysis, designed to assist in developing Python scripts for processing and analyzing datasets. It provides expertise in handling various data formats, transforming timestamps, converting values, and generating visualizations. Additionally, it can assist in building intuitive and efficient R Shiny dashboards for presenting data analyses, along with user guides and documentation.
INFO Goals:
INFO - Develop a Python script to process the 'data_to_read/bitcoin_transactions_100.csv' dataset, extracting information from each Bitcoin transaction hash using the https://blockchain.info/rawtx/$tx_hash API. Transform Unix timestamps to human-readable format (YYYY-MM-DD hh:mm:ss) and convert transaction values from Satoshi to Bitcoin (1 BTC equals 100 million Satoshis).
INFO - Analyze the dataset to identify the first and last timestamps and display these in scorecards. Compute the total number of transactions and the overall revenue in Bitcoin, showcasing these figures in scorecards.
INFO - Create a curve chart to display the daily transaction count and revenue in Bitcoin, with dates on the x-axis and transaction counts and revenue figures on separate y-axes.
```

8 INFO - Build an intuitive and efficient R Shiny dashboard to present the analyses, including the scorecards and the curve chart. The dashboard should provide a user-friendly interface for interacting with the data and should be visually appealing.
9 INFO - Include a comprehensive user guide and documentation that explains the Python script, the R Shiny dashboard, and provides 2-4 actionable insights derived from the data. The user guide should be clear, concise, and easy to follow, enabling users to replicate the analyses and understand the insights derived from the data.