A Work Project, presented as part of the requirements for the Award of a Master's degree in Management/Business Analytics from the Nova School of Business and Economics.

AI-Driven Software Development:

How has the rise of ChatGPT impacted GitHub users?

-

Identification of Control and Treatment Group

Niklas Mundt

Work project carried out under the supervision of:

Michail Batikas

25/02/2024

Abstract

This thesis analyses AI-driven Software Development practices, investigating how the ban of ChatGPT in Italy affected GitHub activity using a Difference in Difference analysis. It covers the process of retrieving and processing data from GitHub Archive to form a four-week dataset spanning the period from 17/03/2023 to 14/04/2023. A scoring and selection approach to identify users from Italy (Treatment Group) and Germany (Control Group) resulted in a dataset comprising 244,401 commits from 10,520 individual GitHub users. Results suggest that users affiliated with organizations show a 12.12% increase in GitHub events, implying a decrease in coding efficiency after the ban of ChatGPT. In contrast, individual users' activities remain largely unaffected by the ban. Coding errors rose by 8.91% on business days, further indicating a reduction in code quality, while results for weekends and public holidays were insignificant. Lastly, organization-related users active on business days exhibited a 13.39% increase in GitHub events post-ban, suggesting a reduction in coding efficiency, and a 20.6% increase in coding errors, pointing to a decline in code quality. No empirical evidence is found for the bans' effects on collaboration practices. These findings suggest that ChatGPT is well integrated into the daily software development workflow and actively used to assist in writing and debugging code, especially in professional settings.

Keywords: ChatGPT; Software Development; GitHub; Code Quality; AI; LLM; Impact; Code Efficiency; DID

Acknowledgments

We would like to express our deep appreciation to our advisor, Michail Batikas, for his advice, feedback, and great disposition while helping us in this journey. We would also like to extend a warm thank you to our families, for their continued support and inspiration during these months.

Table of Contents

Figures

Tables

I. Introduction

Recent developments in Natural Language Processing (NLP) have revolutionized the field of Artificial Intelligence (AI). The combination of Large Language Models (LLMs) with user-friendly interfaces such as ChatGPT, Google's Bard, or Bing Chat enable entirely new ways of human-AI interaction, with tremendous opportunities for Business and Software Engineering (Teubner et al. 2023). Their exceptional ability to understand nuanced contexts and provide largely accurate responses even in complex scenarios will potentially transform the way we interact with machines, allowing for more human-like and intuitive communication (Roumeliotis and Tselikas 2023). As LLMs advance, so does the community surrounding them. Access to LLMs becomes more democratic, allowing independent developers and researchers to be involved instead of restricting access to private institutions. This leads to increased activity on collaboration platforms such as GitHub, arXiv, and Stackoverflow, especially regarding LLM topics (Barua, Thomas, and Hassan 2014; Son and Kim 2023).

The implications of AI in software development are, therefore, plenty, and the recent advances come with both opportunities and challenges. One of them lies in striking a balance between leveraging the capabilities of such models for increased efficiency while keeping the human aspect that makes software development what it is (Rahmaniar 2023).

The following work aims to provide new insights into the field of AI-driven software development, specifically, how the rise of ChatGPT impacted Software Development. As the adaptation to ChatGPT was gradual and the new technology omnipresent, measuring its impact was challenging, especially due to the lack of a natural control group without access to the new Chatbot. This thesis therefore applies a Difference in Difference analysis, leveraging the ban of ChatGPT in Italy from 01. April 2023 to 28. April 2024 to measure how the restricted access to the Chatbot affected Italian developers (Privacy Laws & Business 2023). More precisely, it

focuses on the dimensions of collaboration, code quality, and coding efficiency among developers.

This work comprises nine parts. The first introduces software development and ChatGPT, and further explores AI's impact. The second chapter outlines the research design, objectives, and methodology of the Differences-in-Differences analysis. The third chapter explains the technical aspects of data processing, server and infrastructure setup. Section four then covers the approach to identify treatment and control groups from the raw data. The following three chapters five to seven address ChatGPT's impact on collaboration, code quality and coding efficiency. Next, the eighth part examines the previously studied effects on subsamples for organizational and individual users, as well as business days and weekends. Finally, chapter nine concludes with a discussion on findings, their significance, the limitations of this study and recommendations for future research.

The thesis contributes to the literature on AI-driven software development in the following ways. Firstly, by utilizing a large dataset of daily-level GitHub user data, we perform a comprehensive analysis of the communities' activity during this timeframe. Furthermore, we perform an in-depth analysis of subsamples for organizational software projects and business days to gain further insights into the affected groups, showing whether individual users react differently to the ban of ChatGPT compared to organizations, and whether the effect is different for business days compared to weekends and public holidays. This analysis, especially considering the fast developments in the field, can provide interesting insights and implications for business practices and the software development research in general.

1. ChatGPT & Software Development

1.1 Software Development

Software development is a multifaceted field focused on discovering and implementing effective methods for developing and enhancing software. This comprehensive process involves various activities such as coding, designing, and producing documentation that guides and records the creation and modification of software. In parallel, there are analytical tasks involved in assessing and measuring various aspects of software systems, ranging from their functionality to non-functional attributes like performance and reliability (Shaw, n.d.).

General software development research or life cycle seeks to address a diverse array of questions, from refining methods and analyses to exploring the intricacies of designing and evaluating specific software instances. It also delves into broader concepts, like overarching principles that apply to entire categories of systems or methodologies or even more exploratory aspects, such as the existence or feasibility of innovative ideas and solutions. It then must test these designs and developments to ensure their quality before implementation (Gurung, Shah, and Jaiswal 2020). The tangible outcomes of software development research may take the form of practical techniques for development and analysis. They may manifest as comprehensive models that draw insights from specific instances, or as specialized tools, solutions, or findings

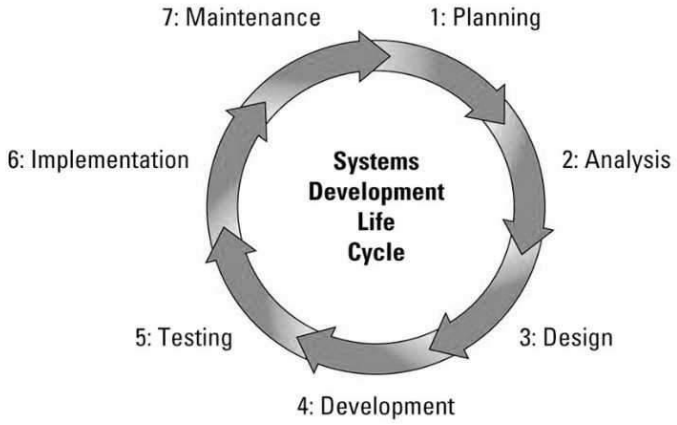

Figure 1: Seven Stages of System Development Life Cycle (Kukhnavets 2019)

tailored to specific software systems. Lastly, software development also encompasses all the maintenance work that comes after outcomes have been reached (Hossain 2023).

The software development landscape has witnessed significant transformations, with new technologies playing a pivotal role in the orchestration of software development. This evolution can be categorized into two main avenues: technology as an orchestrator and technology as a deployment environment (Maruping & Matook 2020).

Regarding the first avenue, historically, technology has been used to manage code contributions in software development, ensuring that multiple developers can collaborate without overwriting each other's work. Concurrent version systems emerged in the late 1980s to log changes made to the code base, primarily benefiting open-source software communities. These rule-based technologies supported large-scale development efforts. Today, a new set of actors, such as Application Programming Interfaces (APIs) have gained prominence, enabling developers to access pre-built functionalities from various software applications. This shift reduces the need to code functions from scratch, allowing developers to focus on combining various APIs to create innovative software solutions. However, the implications of APIs on software development orchestration remain relatively unexplored (Maruping & Matook 2020).

The second main avenue, the deployment environment for software, has also evolved with the advent of the Internet of Things (IoT). Everyday objects, such as sensors, wearables, and mechanical parts, have become imbued with computational capabilities. This convergence of software and autonomous equipment presents new challenges and opportunities for developers, having to come up with new, innovative ways of solving everyday issues with coding (Fahmideh et al. 2021). Digital platforms, such as GitHub, have also had a profound impact on software development. They have facilitated collaboration between organizations with commercial interests and open-source communities, accelerating the convergence of these two distinct software development approaches. Additionally, digital platforms have opened the door

to large-scale user involvement in software development, allowing a vast number of users to provide input, identify defects, and suggest features. This represented a very significant shift in user engagement and communication, but its full implications for software development orchestration are yet to be fully understood (Maruping & Matook 2020).

The biggest transformation factor to software development in the last decade may be, however, Artificial Intelligence and the models and tools spanned from it. Presenting a modicum of new ways of automating and interacting with code, AI has impacted software in many ways since its inception, a few of which will be explored below.

1.2 Artificial Intelligence and Software Development

Defining artificial intelligence proves to be a complex endeavour, with various interpretations in circulation, leading to potential confusion. The absence of a universally accepted definition further complicates the matter. It is essential to clarify the terminology to gain a comprehensive understanding. The diversity of definitions is not due to carelessness but intrinsic to the multifaceted nature of AI (Sheikh et al. 2023).

At its broadest, AI is often linked with algorithms, which is not particularly informative for analytical purposes, as algorithms have been used extensively outside the realm of AI. Artificial Intelligence, if solely defined as the use of algorithms, would encompass activities such as operating a pocket calculator or following instructions in a cookbook, which are clearly distinct from AI's scope, that being of a more advanced simulation of human intelligence in some definitions (Xu et al. 2021).

This strict interpretation, however, may not be suitable for a comprehensive assessment since many contemporary applications fall short of imitating complex human skills. A common definition describes AI as technology enabling machines to mimic a range of complex human abilities. However, without specifying these "complex human skills," this definition leaves AI's nature ambiguous. Some definitions delve further into these skills and tasks, such as the ability

to function appropriately and foresee outcomes in a given environment, the capacity to perceive, pursue goals, initiate actions, and learn from feedback loops. These task-oriented definitions offer better insight into AI but may still appear broad, encompassing phenomena that would not typically be associated with AI. The challenge in defining AI arises from its emulation of human intelligence, a subject under continuous exploration by various disciplines, without a definitive understanding of what constitutes human intelligence (Sheikh et al. 2023).

In essence, AI embodies the imitation of an aspect of human capabilities that remains incompletely understood, making it a continually evolving field with evolving definitions. Its impact in Software development, however, can be somewhat more clearly defined, with concrete applications and models put to work in a variety of tasks.

In the realm of software development, AI automation plays a crucial role in two key phases of the development process: integrating specific pieces of code into a larger framework and problem-solving during the coding phase (Latinovic & Pammer-Schindler 2021).

The integration phase comes into play when a software engineer has devised a functional solution that needs to be integrated into the existing codebase and validated. The conventional approach involves leveraging continuous integration, which is the practice of merging developer's working copies to a mainline code base, configured according to project-specific requirements, and executed through an automated build process designed to detect errors at the earliest possible stage (Latinovic & Pammer-Schindler 2021). The build process involves the execution of predefined build modules defined by a Continuous Integration pipeline, resulting in either a successful or failed build, often accompanied by a report detailing the causes of failure, such as syntax errors or failed tests. In a study encompassing 20 open-source projects, various AI Static Code Analysis Tools have been identified as components of Continuous Integration (Zampetti et al. 2017). These tools serve purposes such as checking coding style, identifying bugs leading to noticeable errors, detecting anti-patterns like unused objects and

superfluous code blocks, examining license validity and the presence of license headers, analyzing dependencies, and recognizing compatibility issues. While Continuous Integration successfully pinpointed issues in the analyzed code, there were instances where the application and configuration of CI tools led to false errors (Zampetti et al. 2017). This underscores the dual nature of automation tools, which can be immensely beneficial when used correctly but may also yield adverse effects when misapplied.

The phase of problem-solving and coding revolves around a software developer's endeavor to comprehend and address a particular problem, focusing on the implementation of the resulting algorithm being designed (Gurung, Shah, and Jaiswal 2020). During this phase, AI tools like a programming assistant, known as PAPA, powered by IBM Watson's cognitive computing technology, have proven to be valuable. PAPA excels at responding to theoretical questions using Natural Language Processing and a predefined knowledge base, although it does not support debugging code through program analysis techniques. Nevertheless, recent research into software engineer motivations and challenges in utilizing Machine Learning frameworks has emphasized the significance of having pre-built models that showcase best practices, offer practical examples, and facilitate hands-on learning (Latinovic & Pammer-Schindler 2021).

In this case, traditional Language Models (LMs) have commonly served as the foundation for text and code generation and comprehension, as previously mentioned with examples like PAPA. However, recent advancements in computational power, machine learning techniques, and access to extensive datasets have given rise to Large Language Models. LLMs are equipped with vast and diverse training data, enabling them to effectively mimic human linguistic abilities. These models can learn from extensive corpora and produce coherent text, bridging the gap between human and machine-generated language. As a result, LLMs have become powerful tools for researchers and developers, revolutionizing the field of language processing and coding, bringing about revolutionary coding processes (Hou et al. 2023).

One of such LLMs is the now highly recognized ChatGPT. It has incited numerous discussions and research papers regarding the influence of AI across a multitude of domains, primarily attributable to its pioneering functionalities and its rise to general public prominence. The ensuing section will undertake an examination of the nature of this Large Language Model and offer an exploration of its ramifications in the domain of software development.

1.3 ChatGPT

ChatGPT is an LLM, created and designed by OpenAI to comprehend and respond to prompts with detailed answers. This model's core is rooted in Reinforcement Learning from Human Feedback (RLHF). ChatGPT has gone through many iterations, currently working at version 4.0, far and improved from where it began in 2018 with GPT-1.

1.4 Model History

Built on the Transformer architecture, this first model focused on language modelling and translation tasks. It was trained on diverse text sources like books and web content to mainly predict subsequent words in a sequence. Despite its 117 million parameters, relatively small compared to later versions, GPT-1 showcased good performances across language-related tasks. GPT-2 marked a significant leap with 1.5 billion parameters, distinguishing itself as one of the largest language models upon release. Similar to its predecessor, it excelled in predicting text sequences but demonstrated superior coherence and the ability to adapt to various tasks. Notably, GPT-2's capacity to generate highly convincing, human-like text raised concerns about potential misuse and OpenAI even initially withheld the full model due to fears of misinformation, releasing a scaled-down version to mitigate these risks (Ray 2023). GPT-3 pushed the envelope in size and capability, boasting 175 billion parameters. Trained extensively on diverse textual sources, it excelled in generating coherent, high-quality language. What set GPT-3 apart from the two previous models was its versatility in tackling an array of language processing tasks without specific task-based training data. Innovations like multi-task learning

and few-shot learning contributed to its adaptability, enabling simultaneous multitasking, and learning new tasks from minimal examples. GPT-4 is the latest and current version of the model, now accepting both text and image prompts and also demonstrating human-level performance on professional and academic settings. This model's development focused on fixing the past versions issues, providing better results in factuality, steerability and staying focused on user-given boundaries and inputs. The next section will focus on explaining how the latest models are constructed and provide examples of their usage and performance.

1.4.1 Model Inner Workings and Applications

Initially, ChatGPT goes through supervised fine-tuning, where human AI trainers assume both user and AI assistant roles in conversations. They benefit from model-generated suggestions to enhance their responses. To enable reinforcement learning, a reward model is implemented, necessitating a collection of comparison data, consisting of multiple model responses ranked by their quality. This is accomplished by selecting model-written messages from real trainer-chatbot interactions, generating alternative completions, and having AI trainers rank them.

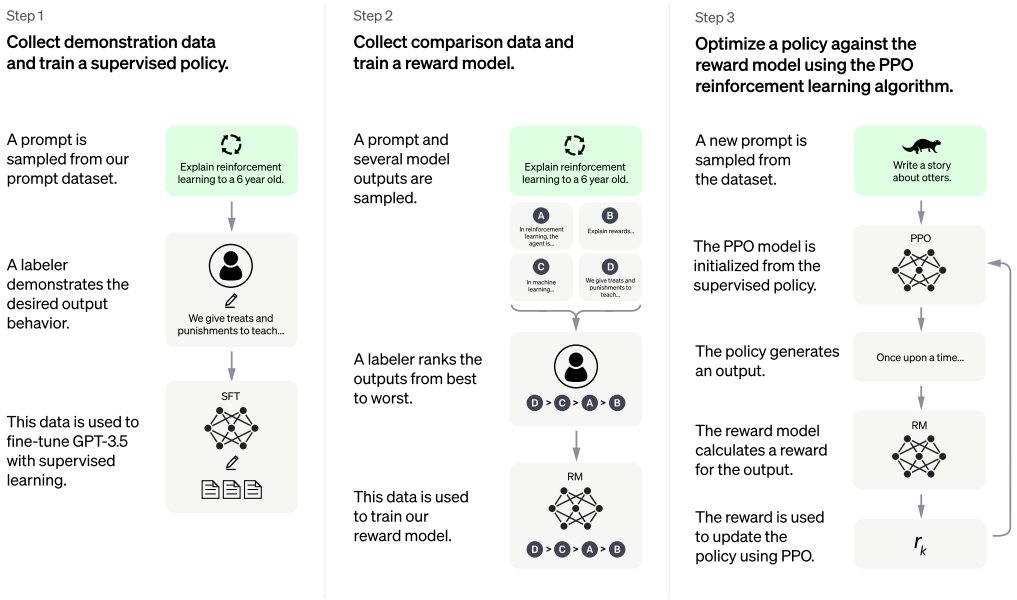

Figure 2: ChatGPT training steps and inner workings (Ray 2023)

Proximal Policy Optimization (PPO) is utilized for fine-tuning, a process iterated multiple times to enhance the model's performance (Kalla and Kuraku 2023).

Despite these processes, ChatGPT has certain limitations. It occasionally generates answers that, while sounding plausible, are factually incorrect or nonsensical. This challenge is compounded by the absence of a definitive source of truth during reinforcement learning training, the fine balance between cautiousness and accuracy, and the pitfalls of supervised training. Additionally, the model is sensitive to variations in input phrasing and can produce differing responses to slightly rephrased prompts. Some of the model's biggest downsides also include presenting training biases that may perpetuate discrimination or stereotypes that were inserted in the model's training data and its limited knowledge when regarding less-common topics or highly specialized data (Kalla and Kuraku 2023).

In essence, ChatGPT is an AI language model that provides answers and assistance through a combination of supervised and reinforcement learning techniques. On the realm concrete applications, the model has shown to be very adaptable, with multiple possible uses and successful tests and processes in fields like text classification, text generation, translation and, more importantly, software development.

One of the model's key applications mentioned above is in the realm of text classification. Text classification is a fundamental NLP task that involves assigning text data to predefined categories. This has wide-ranging applications such as sentiment analysis, spam detection, and topic modelling. Traditionally, text classification relied on specific machine learning algorithms. However, recent advances in NLP have brought forth more advanced techniques, with ChatGPT presenting itself as a great option. Its ability to accurately classify text, flexibility in handling various classification tasks, and potential for customization make it a valuable tool for text classification (Liu et al. 2023). Several studies in the literature have reaffirmed ChatGPT's remarkable potential in text classification. For example, Kuzman et al. (2023)

employed ChatGPT for automated genre recognition, simplifying the text classification task. ChatGPT demonstrated impressive results, achieving high Micro F1, Macro F1, and Accuracy scores, highlighting its efficiency in this genre recognition task. Such scores are a way to measure a Machine Learning models' results effectively and will be further mentioned and explained in this research.

Another study conducted by Amin et al. (2023) evaluated ChatGPT's text classification ability in affective computing, encompassing tasks like personality prediction, sentiment analysis, and suicide ideation detection. Their findings revealed that ChatGPT's accuracy and UAR (User Accuracy Rate) for these tasks varied across datasets, often outperforming baseline methods in some scenarios. Furthermore, ChatGPT can also be applied in stance detection, which includes identifying support and opposition in text. In a study by Zhang et al. (2022), ChatGPT classified the political stance of tweets in datasets such as SemEval-2016 and P-Stance. The model achieved impressive F1-m scores, demonstrating its capabilities in this complex classification task.

These examples underscore ChatGPT's potential in text classification tasks. However, it's crucial to recognize its failings in this domain. It might struggle in classification tasks with rare or out-of-vocabulary words since its performance heavily relies on the distribution of training data. Additionally, the substantial computational resources required for training and utilizing ChatGPT may limit its use in some applications (Liu et al. 2023).

Expanding beyond text classification, ChatGPT finds its true potential in the realm of text generation. Text generation is a versatile NLP task that has far-reaching applications, from automated content generation to enhancing human-computer interactions. As we delve deeper into this aspect of ChatGPT's capabilities, it becomes apparent that this model represents a step forward in generating text with remarkable proficiency (Liu et al. 2023). Researchers have harnessed ChatGPT to generate diverse forms of text, ranging from short phrases to complete

paragraphs. The LLM's ability to generate phrases is exemplified by the work of Zhang et al. (2022), who leveraged it to simplify motion recognition. They introduced a semantic augmentation approach, where ChatGPT effectively generated labels for datasets that initially lacked shared tokens, thereby simplifying the whole recognition task.

Another study by Fu et al. (2023) demonstrated ChatGPT's utility in generating Bash commands, a language designed for users to navigate through Linux's system and manage files and directories, from natural language commands. They used the model to generate a candidate list of Bash commands based on user input and then employed heuristic and machine learning techniques to rank and select the most likely candidates, with positive results showcasing the model's efficacy in this task. In the domain of generating sentences, the model can also be utilized. For instance, N. Chen et al. (2022) created a dialogue dataset, HPD, and used ChatGPT as a conversational agent to generate dialogue. While ChatGPT exhibited promise in this endeavour, it also revealed room for improvement and potential for refinement.

In another intriguing application, ChatGPT demonstrated its ability to simplify complex text. Complex radiology reports were simplified using ChatGPT, with feedback from radiologists indicating that the simplified reports were both accurate and complete, albeit with some identified errors (Liu et al. 2023). Further explorations by Xia & Zhang (2023) delved into automated program repair, where ChatGPT outperformed other models in repairing datasets, showing positive results in this realm.

In the context of translation, ChatGPT was evaluated against commercial products, demonstrating competitiveness with high-resource languages, and showcasing innovation through strategies like pivot prompts. Moreover, ChatGPT played a role in developing automated construction schedules based on natural language prompts, albeit with identified limitations that require further refinement (Liu et al. 2023).

2. Research Design & Objectives

Within the software development domain, one of ChatGPT's most notable contributions is code generation. Code generation involves automatically producing computer code from high-level descriptions or specifications, an area where ChatGPT's advanced NLP capabilities are used. The model's ability to analyze such high-level requirements and translate them into code snippets capable of executing the intended functionality generally presents a paradigm shift in software development: This automation not only reduces manual coding effort but also minimizes the risk of human errors often inherent in the coding process (Liu et al. 2023).

Megahed et al. (2023) highlighted ChatGPT's potential in code explanation, suggesting alternative problem-solving methods with code, and translating code between programming languages. Such applications open new possibilities for developers to streamline and optimize their work. Treude (2023) introduced GPTCOMCARE, a ChatGPT-based prototype that assists programmers in generating multiple solutions for programming problems and highlights the differences between these solutions, empowering developers to explore multiple avenues, a process that was traditionally especially time-consuming and labor-intensive. In addition to code generation, ChatGPT's versatility in dealing with different programming languages and frameworks is a testament to its readiness for complex programming tasks.

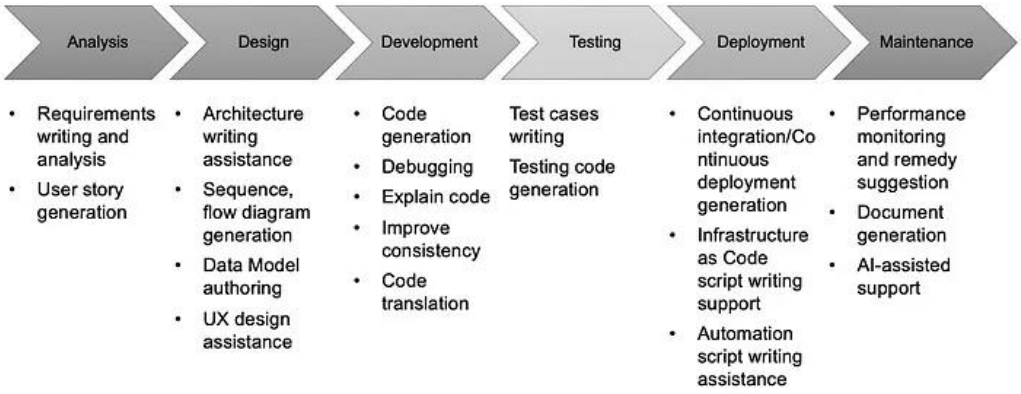

Figure 3: Example of ChatGPT and generative AI's applications in the Software Development Life Cycle (Cheng 2023)

Its ability to access structured datasets and perform standard database operations such as creating, reading, updating, and deleting (CRUD) is indicative of its potential in developing data-centric applications (Liu et al. 2023).

However, it is essential to acknowledge that while ChatGPT offers significant contributions to code generation, challenges persist. The model's training data is predominantly skewed toward popular programming languages such as Python, C++, and Java. This training bias may limit its suitability for less popular languages and coding styles (Liu et al. 2023). Additionally, developers may need to manually optimize code formatting, performance, and best coding practices in the generated code. The quality of generated code is also highly contingent on the quality of the natural language input, with errors, ambiguities, or inconsistencies in input potentially affecting the accuracy and reliability of the generated code (Liu et al. 2023).

However, the general question of the matter of how, in a quantifiable way, ChatGPT has affected the realm of Software development with the implementation of all these new methods and technologies remains largely unanswered. Has the employment of such techniques as Treude (2023) applied, or the model's potential in generating code explanations, evidenced by Megahed et al. (2023), improved developer's work quality, efficiency, and collaboration?

The goal of the following work is to quantify this effect and the influence of the widespread use of ChatGPT, with a specific focus on software development. Specifically, three areas will be analysed, resulting in the following research questions:

RQ1: *How has the introduction of ChatGPT affected collaboration in Software Development?*

RQ2: *How has the introduction of ChatGPT affected code quality in Software Development?*

RQ3: *How has the introduction of ChatGPT affected coding efficiency in Software Development?*

However, this work aims to go even further. While the primary research questions address the overall effects of ChatGPT's on collaboration, code quality, and coding efficiency, the diverse

nature of the software development community and the event circumstances call for a more nuanced approach. Therefore, we will further analyse ChatGPT's impact on distinct subsets, to capture potentially heterogenous effects that might otherwise be hidden.

Firstly, the software development community includes both individual developers and larger organizations. Organizations with structured teams and larger resources might be able to use ChatGPT differently compared to individual developers using it for personal and independent projects. Adding to this, the activity can also fluctuate within the week, with business days potentially showing a different pattern of ChatGPT activity compared to weekends and public holidays. These reflections led to the following sub-questions, aiming to reveal the nuances of ChatGPT's impact on software development.

RQ4: *How has the introduction of ChatGPT affected collaboration, code quality and coding efficiency differently in organizational settings compared to individual user environments?*

RQ5: *How has the introduction of ChatGPT affected collaboration, code quality and coding efficiency differently on bufsiness days compared to weekends and public holidays?*

RQ6: *Has the introduction of ChatGPT affected collaboration, code quality, and coding efficiency differently in organizational settings on business days compared to weekends and public holidays?*

By addressing these sub-research questions, the study intends to provide a layered understanding of how advanced language models like ChatGPT are influencing the domain of software development. However, answering these questions in a real-world setting is challenging, as an ideal evaluation would require a randomized controlled trial, with a group of individuals being randomly chosen to use ChatGPT, while others are not. Such experimental conditions are hard to find, as deliberately denying some businesses access to the latest beneficial technology is neither practical nor ethical.

Unexpectedly, the regulatory decision by Italy on April 1st, 2023, to ban ChatGPT over concerns of consent and privacy of personal data has provided just such an opportunity. The ban's implementation was abrupt, enforced by Italy's privacy watchdog without previous public indication or dialogue with OpenAI, the creators of ChatGPT. This ensures that there were no anticipatory behavior changes by users due to forewarning of the event and it can be considered an exogenous shock to the system. Furthermore, the fact that the ban was initiated due to regulatory concerns, independent of economic motivations related to ChatGPT, strengthens the argument that the observed effects can be linked to the technology's absence rather than other economic factors. Starting on April 1st, 2023, the one-month ban, which remained in effect until April 28th, 2023, provides a clearly defined timeframe for assessing its impact (Privacy Laws & Business 2023). Further, measuring the effects of the ban on Italian software developers requires for a control group, a group of users unaffected by the ban to which the effects in Italy can be compared to. Germany was selected to be the control group due to several reasons. Primarily, Germany's economic and technological landscape is closely aligned with Italy's, providing a similar context in terms of development practices and industry standards. Both countries have mature software development industries and share comparable levels of technological infrastructure and access to digital tools, which is crucial for such a comparison. Additionally, the cultural and regulatory environments in Germany offer a parallel to Italy, which ensures that any observed differences in software development practices are more likely due to the absence of ChatGPT rather than differing national characteristics.

2.1 Data Source – GitHub

Real-world time-based data is commonly used in software engineering research, as it is a convenient way to obtain information on when certain actions or events occurred, indicated by a respective timestamp (Flint et al. 2022). GitHub presented a suitable data basis for our research, being described as a platform that goes beyond hosting source code, fostering

collaboration and contribution to open-source projects (Braga et al. 2023).

The community-driven approach of GitHub is highlighted, encouraging sharing, learning, and collective problem-solving, thus serving as an incubator for innovation and a repository of a vast array of software solutions (Borges et al. 2016). Additionally, GitHub has significantly shaped modern software development, especially in the realm of open-source projects. It's widely hailed as a key platform for fostering collaborative and reproducible research in various areas of study (Bhattacharjee et al. 2020). The platform's role in facilitating diverse collaboration and its influence on the software development community are also acknowledged, with GitHub being embraced as an essential platform for managing software projects (Zagalsky et al. 2015).

2.1.1 GitHub – Definition and Features

To understand how we are going to use this platform and the data provided by it, one must first understand what GitHub itself is, its nuances and how it affects general users and organizations. GitHub is a collaborative software platform that serves as a hub, hence the name, for developers to work together, either by sharing code or managing version control for software projects.

It provides a centralized space where individuals and teams can host their code, track changes, and collaborate seamlessly (GitHub, n.d.). At its core, GitHub leverages Git, a distributed version control system that enables users to keep track of changes made to their codebase over time, allowing developers to work simultaneously on different aspects of a project, merge their contributions, and maintain a coherent and up-to-date codebase (GitHub, n.d.).

In our analysis, we will be looking specifically at commits and repositories. Starting with the latter, one can say that GitHub's key feature may be its repository function, as it allows users or organizations to create repositories to store their projects, manage access control, and contribute code, suggesting changes, or reviewing code via pull requests (Figure 4). This collaborative nature extends beyond code; GitHub also offers issue tracking and project

management tools that enable teams to organize tasks, track bugs, and coordinate their workflow efficiently (GitHub, n.d.).

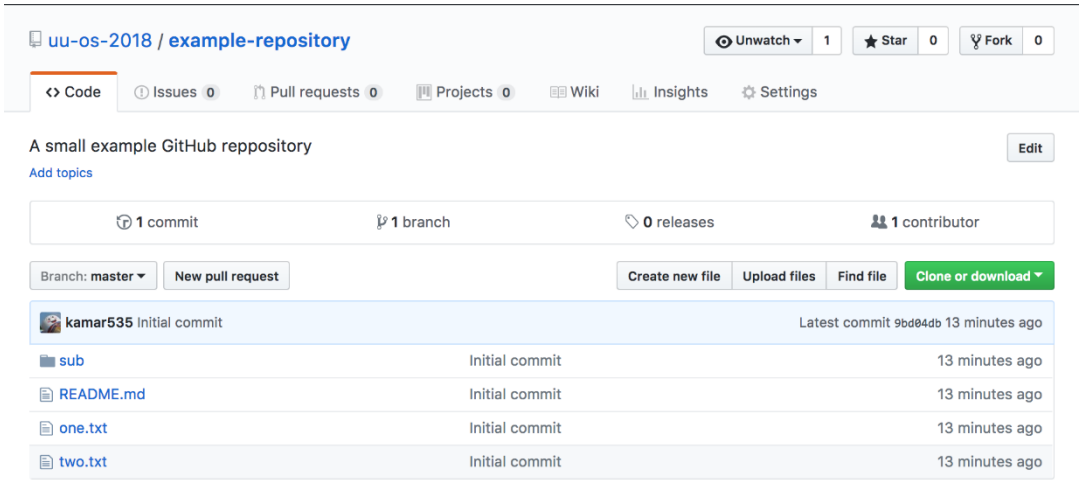

Figure 4: Example of a GitHub repository user-interface (Operating systems 2017)

Users interact with these repositories, however, by utilizing commits. A commit represents a specific snapshot or a point-in-time version of a repository, encapsulating changes made to files within the repository at that specific moment. Each specific commit records a set of modifications, additions, or deletions made to the repository's files (GitHub, n.d.). According to Git's official documentation, a commit consists of four main key elements:

1. Unique Identifier (SHA-1 Hash): Each commit is uniquely identified by a SHA-1 hash; a 40-character hexadecimal string generated based on the commit's contents. This hash serves as a unique fingerprint for that commit, allowing the possibility of tracking commits generated by specific users and analyzing user behavior.
2. Author and Committer Information: A commit includes details about the author and committer. The author is the person who originally created the code changes, while the committer is the person who applied the changes to the repository. These details typically include the name and email address of both individuals and a timestamp indicating when

the commit was made. Many times, however, the author and committer end up being the same user, only providing then one set of information.

3. Commit Message: A commit is accompanied by a commit message—a descriptive note that explains the changes introduced in that commit. The commit message provides context to collaborators, outlining the purpose, scope, and significance of the modifications. This message varies depending on the type of commit, and these differences will be explored below.

Changes Introduced: Each commit represents a set of changes made to the repository's files. It includes information about the modified files, additions, deletions, or updates made to specific lines within those files.

2.1.2 Commit and Repository Event Types

GitHub sorts commits and repository interaction in different categories, called event types. These encompass a myriad of actions, and our analysis will target specific types depending on what is to be analyzed, looking for the best suitability in terms of data. According to official GitHub documentation, there are 16 unique commit and repository event types, listed and explored in Table 1 below:

#	Event Type	Definition
1	*CommitCommentEvent*	Triggered upon comments made on specific commits. These comments often contain context, feedback, or discussions related to the changes introduced in that particular commit.
2	*CreateEvent*	Signals the creation of repositories, branches, or tags. This event furnishes details about the newly created entity and the user responsible for its creation.
3	*DeleteEvent*	Indicates the removal of repositories, branches, or tags. It specifies the entity that was deleted and the user accountable for the deletion.
4	*ForkEvent*	Captures instances of repository forking, providing information about the source repository and the user initiating the fork.
5	*GollumEvent*	Notifies when Wiki pages are created or modified, offering insights into changes made to

		the Wiki, such as page creation, updates, or deletions.
6	*IssueEvent*	Indicates when issues are opened, closed, or reopened in a repository. Contains data regarding the issue, including its title, number, state, and the user who acted upon it.
7	*IssueCommentEvent*	Triggers upon comments added to issues within a repository. It includes comment content and the user who posted it.
8	*LabelEvent*	Signifies label changes within a repository, encompassing label creation, deletion, or updates. Provides details about the label and the user responsible for the modification.
9	*MilestoneEvent*	Triggered upon milestone creation, closure, or updates. Contains milestone details, like title, state, and the user performing the action.
10	*PublicEvent*	Marks the transition of a repository from private to public status, signifying when a repository becomes publicly accessible.
11	*PullRequestEvent*	Occurs upon pull request actions such as opening, closing, merging, or synchronization. Provides information about the pull request, its number, title, state, and the user initiating the action.
12	*PullRequestReviewEvent*	Generated upon submission, editing, or dismissal of a review on a pull request. Details the review and the user conducting the review action.
13	*PullRequestReviewCommentEvent*	Triggers when comments are made on pull request reviews. Contains comment details and the user who posted it.
14	*PushEvent*	Indicates commits pushed to a repository, providing information about the commits, their IDs, messages, and the user responsible for the push.
15	*ReleaseEvent*	Notifies about the creation or publication of a release within a repository, offering specifics about the release, like tag name, target commit, and the user creating it.
16	*WatchEvent*	Triggered when a user stars a repository, signifying their *watching* or following of updates to that repository.

Table 1: GitHub commit and repository event types

One of these events deserves a bit more of an in-depth explanation, as it is not as directly interpretable as the others. The *ForkEvent* signifies the act of creating a fork—a personal copy—of a repository. It occurs when a user decides to duplicate an existing repository to their account, allowing them to work on the code independently without affecting the original repository (GitHub, n.d.). Developers often fork repositories to experiment, test changes, or explore new features without affecting the original codebase, serving as a sandbox for personal

development. This act of forking a repository generates then another event type, the *PullRequestEvent*, which basically proposes that that specific fork be merged into the main repository and, in a way, officialized. After a pull request is reviewed and approved, and only then, will the personal fork changes be transferred to the main repository as mentioned above. Figure 5 shows this workflow and process in a simplified way, illustrating the effects of both these event types.

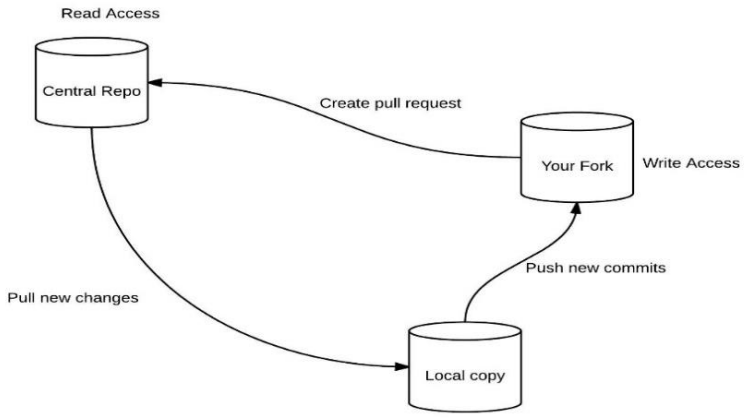

Figure 5: Example of a ForkEvent and PullRequestEvent workflow (Daityari 2016)

As previously stated, our research will be leveraged on different types of commit and repository event types, which include *ForkEvents* and *PullRequestEvents* for example. With this system and different outputs shown in the table above, we can extract enough data to be able to possibly make significant conclusions in various areas. This data is not extracted directly from the GitHub website or its desktop app however, and one of the main sources of this data will be presented in the next section.

2.1.3 GitHub Archive

The GitHub Archive serves as an essential resource for examining the public undertakings on GitHub, essential for software development research (Mombach and Valente 2018). Initiated by Ilya Grigorik, GHArchive.org has archived public events on GitHub since February 2011, capturing a broad spectrum of activities within the platform's public repositories. It leverages GitHub's event API to aggregate a diverse range of activities from repository creation to code

sharing, providing an open-access database for comprehensive analysis (Grigorik 2023). The Archive's commitment to storing event data chronologically has made it an indispensable tool for our research, providing all the publicly available commit and repository events on our specific time period of interest, with the recorded date offering precision even down to minutes. Details about the more technical data retrieval information about GitHub Archive will be discussed and demonstrated in section 3.2.

The next section demonstrates the main method which we will use to perform statistical analysis after retrieving and analyzing the GitHub data presented above.

2.2 Research Design – Difference-in-Differences

The goal of this work is to provide empirical answers to the previously introduced research questions around the influence of LLMs like ChatGPT in collaboration, code quality, and coding efficiency among developers. In a real-world setting however, it is challenging to reliably measure their impact, as an ideal evaluation would require a randomized controlled trial, with a group of individuals being randomly chosen to use ChatGPT, while others are not. Such experimental conditions are hard to find in the real world, as deliberately denying some businesses access to the latest beneficial technology is neither practical nor ethical.

The unexpected regulatory decision by Italy on April 1st, 2023, to ban ChatGPT over concerns of consent and privacy of personal data has provided just such an opportunity. The ban's implementation was abrupt, enforced by Italy's privacy watchdog without previous public indication or dialogue with OpenAI, the creators of ChatGPT. This ensures that there were no anticipatory behavior changes by users due to forewarning of the event and it can be considered an exogenous shock to the system. Furthermore, the fact that the ban was initiated due to regulatory concerns, independent of economic motivations related to ChatGPT, strengthens the argument that the observed effects can be linked to the technology's absence rather than other economic factors. The one-month period during which the ban was in effect, lasting until April

28th, 2023, creates a well-defined window to observe the impact. Further, measuring the effects of the ban on Italian software developers requires for a control group, a group of users unaffected by the ban to which the effects in Italy can be compared to. Germany was selected to be the control group due to several reasons. Primarily, Germany's economic and technological landscape is closely aligned with Italy's, providing a similar context in terms of development practices and industry standards. Both countries have mature software development industries and share comparable levels of technological infrastructure and access to digital tools, which is crucial for such a comparison (worlddata.info 2014). Additionally, the cultural and regulatory environments in Germany offer a parallel to Italy, which ensures that any observed differences in software development practices are more likely due to the absence of ChatGPT rather than differing national characteristics.

In the following analysis, the effect of the introduction of ChatGPT on GitHub is analyzed by means of a quasi-experimental approach, the Difference-in-Differences (DID) analysis. This research design is used to study causal relationships in settings, where conducting randomized controlled trials is not feasible. This method is well established in research fields where it is important to understand the effect of a certain action or event across groups and time periods, dating back to the mid-nineteenth century (Snow 1855).

In general, causal inference lies at the heart of the DID approach, which refers to drawing conclusions about the causal effect of one variable on another, based on observed data. Essentially, by comparing changes over time between a group that was affected by an intervention (treatment group) and a group that was unaffected (control group), it is possible to infer the direct impact of this intervention or treatment. In its simplest form, a DID has two groups, observed over two distinct time periods, which Goodman-Bacon (2021) calls the classic 2×2 DID. In this design, the first difference is calculated by comparing the outcome before with the one after treatment in both groups (control and treatment). Then, the second difference

is simply difference between those first differences, i.e. difference in first differences, providing an estimate of the causal effect of the treatment, assuming certain conditions are met. One critical assumption in this design is the common trend or parallel trend assumption. It stipulates, that in the absence of the intervention, both the treatment and the control group would have followed the same trend over time. If this precondition holds, then any divergence in trends between the two groups can be attributed to the treatment itself, rather than other factors that may not be accounted for (Cunningham 2021).

2.3 Baseline Two-way Fixed Effects Model

In the context of analyzing the effect of Italy's ChatGPT ban on GitHub activity, our panel data includes many entities (users) observed over multiple time periods (four weeks daily measures), which may have unique characteristics that influence the outcome variable. To control for such factors and to isolate the effect of the ban more accurately, a panel regression model with two-way fixed effects is estimated as follows:

$$Y_{it} = \beta_0 + \beta_1 (Italy\_X\_After\_ban)_{it} + \alpha_i + \lambda_t + \varepsilon_{it} \quad (1)$$

In this model, Y_{it} is the dependent variable, which denotes the outcome of interest for user *i* at time *t.* We will fit variants of this regression for each of our variables of interest, going into more detail on the estimation of each of the dependent variables in the respective chapters. For RQ1 for example, it would be the log-transformed collaboration activity-level, defined by its entity (user) and time period. β_0 is the intercept term, representing the expected value of Y_{it} when all independent variables are set to zero. The independent variable of primary interest is the interaction term *Italy_X_After_ban.* Here, *Italy* is an indicator variable, equal to 1 if the observation is from the treatment group (Italian users) and equal to 0 for the control group (German users). *After_ban* is also an indicator variable, equal to 1 if the observation is from the post-treatment period, i.e. during the ban of ChatGPT in Italy. This interaction term is crucial as its coefficient, β_1, quantifies the change in the dependent variable for the Italian users after

introduction of the ban, compared to before the ban and relative to a comparison group over the same time period. It is therefore also considered the Difference-in-Differences estimator. Further, entity fixed effects, denoted by α_i are added to control for all time-invariant characteristics of entities that could potentially influence the dependent variable, thus allowing for the mitigation of unobserved heterogeneity. Similarly, time fixed effects denoted by λ_t, control for factors that fluctuate over time yet remain constant across entities, capturing influences such as macroeconomic conditions or seasonal patterns. Lastly, the error term ε_{it} includes the random variation caused by other unobserved factors which are not explained by the model. For estimation, the *PanelOLS* estimator from the *linearmodels* package is employed (see Appendix 1 in the Appendix). This estimator is particularly suited for fixed effects models in panel data contexts. Further, standard errors are clustered at the entity level to address the potential for serial correlation within entities and heteroskedasticity across them. By clustering the standard errors at user-level, the inference drawn from the model is safeguarded against within-entity correlation, making the standard errors more reliable as a result. This acknowledges that GitHub users may not be independent and may have specific behavioral patterns that could cause their errors to be correlated over time (Nick Huntington-Klein 2019).

2.4 Dynamic DID Model

The introduction of the ChatGPT ban in Italy presented an abrupt policy intervention, with the platform's access being entirely restricted overnight. However, as time progresses, it is reasonable to assume that users may seek and employ alternative methods, such as virtual private networks (VPNs), to circumvent the ban and regain access to ChatGPT (Kreitmeir & Raschky 2023). It is therefore important to understand not only if the ban of ChatGPT affected GitHub activity in general, but also how this effect developed from one day to the next. This can be achieved with a dynamic DID regression model. Methodologically, it incorporates a set of day-specific interaction terms between the treatment indicator for Italy and dummy variables

representing each day within the study period. These interactions (IXD1, IXD2, ..., IXD28) allow for the treatment effect to differ across each day relative to the ban's imposition, while still accounting for unobserved individual heterogeneity and time-specific effects through entity and time fixed effects. To avoid perfect multicollinearity, one reference day is excluded from the model, providing a baseline against which to compare the other days' effects. The dynamic DID regression is formally specified as:

$$Y_{it} = \beta_0 + \sum_{\substack{d=1 \\ d \neq 15}} \beta_d \times (IXD_d)_{it} + \alpha_i + \lambda_t + \varepsilon_{it} \quad (2)$$

Just like in our baseline model, Y_{it} is the outcome variable, β_0 the intercept, α_i and λ_t capture entity and time fixed effects, respectively and ε_{it} is the error term. The dynamic aspect of this model is captured in the sum of the interaction terms $\sum_{d=1}^{28} \beta_d \times (IXD_d)_{it}$. Here, each β_d is a different coefficient representing the change in the dependent variable associated with the ban's effect on day *d*, relative to the omitted day (day 15). The interaction terms IXD_d stand for $Italy\_X\_day_d$ and are created by multiplying a dummy variable for the treatment (*Italy*) with a dummy variable for each day (*day*) (see Appendix 2 in the Appendix for the Code).

By observing the significance and size of the coefficients for each day, we can discern patterns such as delayed reactions, peak impacts, or adaptation over time. The *PanelOLS* estimation summary confirms the model's appropriateness by providing robust standard errors, clustered at the entity level, to address potential serial correlations and heteroskedasticity across entities. As shown in the "Enlightenment: A Flexible Functional Form" chapter, the traditional two-way fixed effects (TWFE) model may suffer from bias due to time-heterogeneous effects, particularly when the treatment effect evolves over time. Our dynamic DID model overcomes this by introducing day-specific treatment effects, thus offering a more nuanced and flexible functional form. This allows for the heterogeneous effects of the treatment to be captured accurately, reflecting the reality that the ban's influence on GitHub activity may change from day to day.

3. Methodology

3.1 Introduction

This chapter describes the research methodology used for this study which focuses on data processing strategies implemented. Given the data-intensive nature of the research, a combination of cloud computing, containerization, and interactive data analysis tools was implemented to ensure efficiency, reproducibility, and collaborative accessibility. The capabilities of the servers provided uninterrupted computing power for all users simultaneously and made it possible to conduct this research within the given timeframe of the thesis, as using the computing power of edge devices would have taken too long to process all the data.

3.2 Data Retrieval

For the purpose of this study the timeframe which will be examined is defined for before the ban took effect (18/03/2023 00:00 – 31/03/2023 23:59) and during the ban (01/04/2023 00:00 – 14/04/2023 23:59).

In the first step, the data is downloaded for each hour and then combined for each timeframe,

In the second step, all users with the ending '*[bot]*' are filtered out, as the aim of this study is to understand the behavior of human developers with the influence of ChatGPT, without the interference of bots – which might be a topic for a different study.

In the third step, the main goal was to filter the data for only Python Repositories. For this separate data collection step is required, as the GH Archive does not contain information about the language of the repository. To achieve this information, two possible solutions are available:

- BigQueries language table
- Using the GitHub API to get the repository language

3.2.1 BigQuery

The BigQuery library enables SQL-based querying to retrieve repository names with Python as their primary language (refer to Appendix 1 in the Appendix).

This approach yields a total of 5,372 Python repositories. Subsequently, these repositories can be used to filter the full dataframe, resulting in a total of 41,080 events (654,271 without checking for primary language). Since this represents less than 0.1% (1.5%) of the initial dataset (43,225,558 *total*), it is necessary to evaluate the second approach – using the GitHub API.

3.2.2 GitHub API

The GitHub API's repository languages endpoint presents a challenge due to its 5,000 requests per hour limit (GitHub, n.d.). Considering the 5,280,182 unique repositories, retrieving this data would take about 44 days. To address this, an efficient data retrieval method was essential. After reviewing several options, GraphQL emerged as the optimal solution for data querying, especially with its GitHub API integration (GitHub, n.d.).

As an advanced, open-source query language, GraphQL excels in data retrieval efficiency, as it allows for precise data extraction, such as determining repository locations (Lawi et al. 2021). Its utility is particularly evident in high-volume API request scenarios, like the anticipated 5,280,182 calls in this study, where rate limits are a significant constraint.

Comparing GraphQL with REST APIs highlights GraphQL's superiority in high-load situations (Brito & Valente 2020):

- Consolidated Requests: GraphQL can combine multiple queries into one, reducing the number of requests and streamlining data retrieval (Mikuła & Dzieńkowski 2020).
- Targeted Data Retrieval: It allows clients to request specific data, minimizing unnecessary data transfer and saving network bandwidth (Farré et al. 2019).
- Improved Performance: Less data transfer leads to faster response times, a crucial factor in handling large datasets and numerous API calls (Lawi et al. 2021).
- Better Error Reporting: GraphQL provides detailed error feedback, aiding in the swift resolution of issues in large-scale request executions.

Given these advantages, GraphQL is well-suited for projects involving extensive data interaction under tight API constraints, as it allows for precise and efficient data fetching, enabling clients to request exactly what they need. This efficiency is key in reducing latency, a common issue with REST in high data demand scenarios (Vazquez-Ingelmo et al. 2017). Figure 6 visualizes the differences between REST and GraphQL.

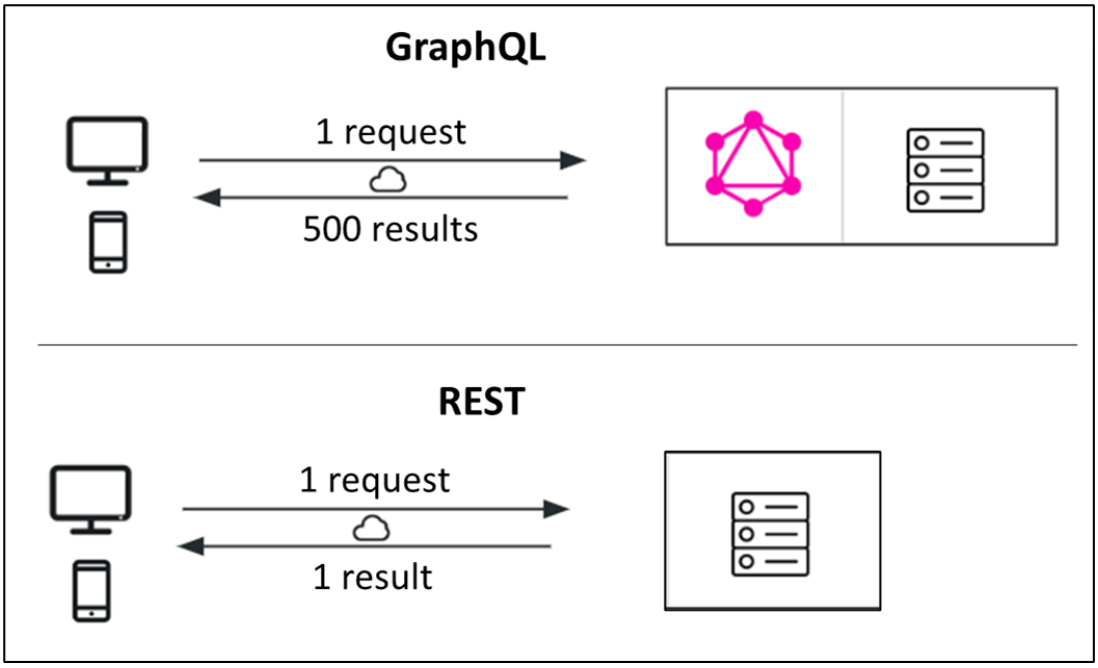

Figure 6: REST vs GraphQL

Overall, GraphQL's ability to handle large-scale data exchanges and optimize server resource use makes it the preferred choice over REST for this project. Its precise, client-focused nature ensures that data transactions stay within API provider limits, maximizing the effectiveness of our technological infrastructure.

Utilizing GraphQL, we are now able to simultaneously request languages for multiple repositories. By integrating asyncio, which facilitates concurrent programming with its async/await syntax for efficient I/O-bound and high-level network code execution, we further enhance request performance (Python Software Foundation 2023). The implementation details, including comprehensive error handling, are provided in the Appendix under Appendix 5.

This approach yields 497,539 unique Python repositories, culminating in a total of 4,521,431 total events within the final dataframe. This represents approximately 10% of the initial dataset, offering an 8.5% increase in data coverage compared to the BigQuery method.

3.3 Data Processing

For the exploration and analysis of the data of the expansive dataset provided by GitHub Archive, we deployed a Jupyter Server as our primary computational environment, leveraging robust, interactive computing capabilities. This platform facilitated a collaborative workspace for team members to execute and share their work in real-time, enhancing both productivity and reproducibility of results.

Considering the voluminous nature of the GitHub Archive dataset, which averages around 1.5 GiB per day of metadata, we focused on a total of 28 days – 14 days before and after the benchmark event, which is the day ChatGPT was banned in Italy – on the 1st of April 2023. This amounted to approximately 45 GiB of raw data. To manage this effectively, we incorporated PySpark into our data processing pipeline. PySpark, the API used in Python for Apache Spark is an engine designed specifically for large-scale data processing. Its capacity to handle vast datasets with great speed and variability made it a useful and efficient asset for our analytical tasks, including the initial stages of the data gathering and filtering.

3.4 Version Control and Collaboration

Given the collaborative nature of this research and the necessity for transparency and reproducibility, we employed GitHub, a leading platform for version control and collaborative software development.

3.4.1 Private Cloud Platform

Our research employed a private cloud that is hosted on ‘bestewolke.de’, powered by the Proxmox virtualization technology. Utilizing a private cloud environment ensured:

Control and Flexibility: With direct control over our virtual environment, we maintained

consistent conditions throughout the research, minimizing potential variables. Furthermore, computations could be executed on the server, eliminating the need for the client side to be continuously online.

Scalability: Proxmox allows for easy scalability, ensuring that as the computational demands of the research grow, resources can be added or reallocated efficiently.

Security and Safety: Our server operated behind a reverse proxy and employed HTTPS connections at all times, along with regular backups. This approach provided all users with a focus on data analysis while minimizing concerns about data loss or accidental deletion, ensuring high overall security.

3.4.2 Containerized Platform

The research operations were conducted within a containerized instance allocated with 32 cores, 128 GiB RAM, and 256 GiB of disk space. Containerization provided us with the following benefits:

Isolation: This ensured that our computations ran in a controlled environment, maintaining the integrity and consistency of results.

Resource Dedication: The dedicated resources, particularly the high RAM, supported efficient in-memory computation, significantly reducing processing times for large datasets, and enabling simultaneous usage by all users.

As seen in Figure 7 a client connects using a web browser to the server's front end. The server is an Ubuntu container running version 23.04 Standard with Jupyter Server 2.8.0, PySpark 3.5.0, and Python 3.11.x

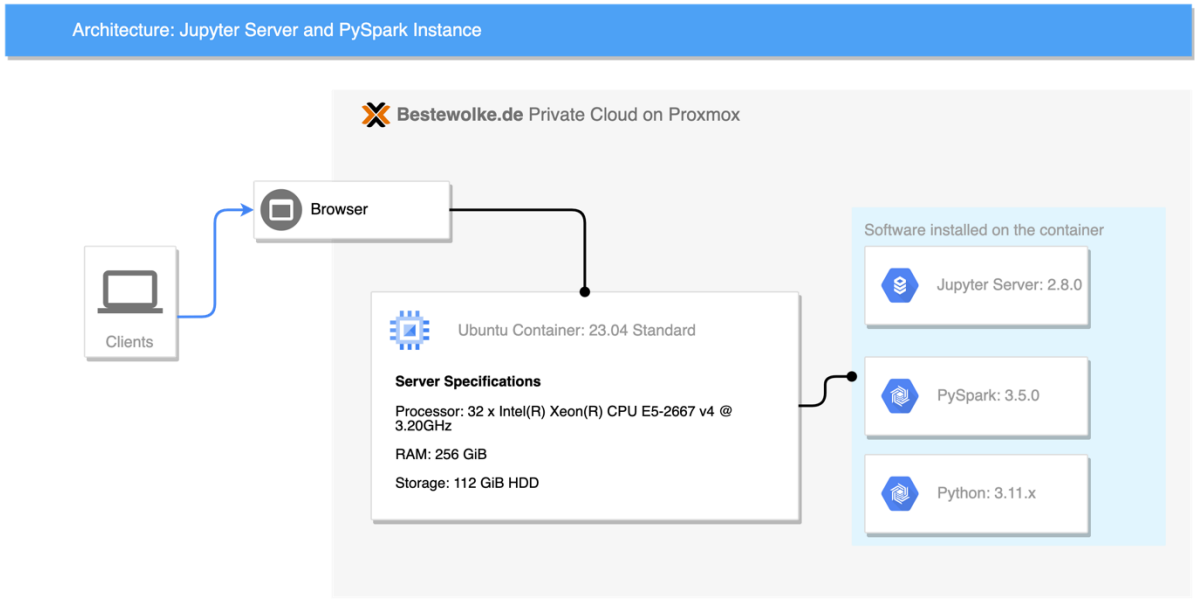

Figure 7: Server Architecture hosting Jupyter and PySpark

II. Research Phase

4. Identification of Control and Treatment Group

In the context of applying the DID approach, the establishment of Control and Treatment Groups emerges as a fundamental prerequisite (Schwerdt & Woessmann 2020). In experimental research, a control group acts as a baseline, allowing researchers to determine the causal effects by keeping the independent variable constant (Philip H. Henning 2013). This methodology involves manipulating the independent variable within the treatment group while maintaining its consistency within the control group, thereby allowing for a comparative analysis between the outcomes observed in each group.

In the context of applying the Difference-in-Differences approach, the establishment of Control and Treatment Groups emerges as a fundamental prerequisite (Schwerdt & Woessmann 2020). In experimental research, a control group acts as a baseline, allowing researchers to determine the causal effects by keeping the independent variable constant (Philip H. Henning 2013). This methodology involves manipulating the independent variable within the treatment group while maintaining its consistency within the control group, thereby allowing for a comparative analysis between the outcomes observed in each group.

Using a control group helps to ensure that any variations observed in the outcome are due to the independent variable, not other factors and thereby mitigating the potential distortion of results. This method also aids in reducing the risk of errors or research bias in the results that might be caused by external or unrelated variables.

In this study, we examine the impact of ChatGPT, a language model, on software development. Italy, where ChatGPT was banned as of April 1st. 2023, to privacy concerns, serves as the control group. The ban was issued by the Garante per la protezione dei dati personali, Italy's authority for data protection, who implemented a suspension on the utilization of the ChatGPT chatbot, referring to issues related to privacy (Garante per la protezione dei dati personali 2023).

The ban led to an investigation into whether OpenAI complies with the General Data Protection Regulation (GDPR). The authority specifically pointed out the absence of a lawful rationale for the extensive aggregation and retention of personal data that serves the training of algorithmic frameworks integral to the chatbot's functionality. Moreover, they emphasize the lack of age verification, even though the service is allegedly addressed to users aged above thirteen.

Germany, with ongoing access to ChatGPT, is the treatment group. The rationale for this choice hinged on the necessity for a comparative country that shares similar for the characteristics with the control group and possesses a primary language that is not extensively spoken beyond its borders, thus providing a defined context for potential language analysis (worlddata.info 2014). In addition, Germany and Italy are similar in size, population, both belong to the EU and have a similar government type (NationMaster.com 2023). Also, important to mention is the strong intertwined history of both countries during the second world war, which still has its effects nowadays (Federal Foreign Office 2023).

In various research domains Italy and Germany have frequently been utilized as benchmarks for contrasting distinct cohorts, bolstering their suitability as candidates for control and treatment groups, respectively (Balta-Ozkan et al. 2014; Bazzani 2017; Notermans & Piattoni 2021; Vrontis et al. 2018).

Subsequent to the decision to focus on Italy and Germany in the study, a significant challenge emerged: identifying users' country of origin in the absence of direct indicators in the data set. Extensive research and a thorough review of the GitHub API documentation revealed several potential identifiers (GitHub, n.d.). Furthermore, the exploration of language analysis as a method for user classification presented itself as a promising approach (Euaa 2022; Patrick 2012). This concept is exemplified by the European Union Agency for Asylum (EUAA), which utilizes Language Assessment for the Determination of Origin (LADO). This tool aids in pinpointing the origin of applicants through linguistic analysis. Such a methodology offers a

potential parallel for deducing user locations in this study, based on their language usage patterns (Euaa 2022).

4.1 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA), as outlined by Shin (2020) and Komorowski et al. (2016), serves as a fundamental process in data analysis. It involves scrutinizing datasets to unearth patterns, identify anomalies, validate hypotheses, and verify assumptions through summary statistics and graphical interpretations. The objective is to optimize the comprehension derived from a dataset while minimizing potential errors that might arise in subsequent stages.

Exploratory Data Analysis is a process in data analysis used for investigating datasets to discover patterns, spot anomalies, test hypotheses, and check assumptions through summary statistics and graphical representations (Shin 2020; Komorowski et al. 2016).

The exploratory efforts are directed towards maximizing the insights of a dataset and minimizing potential errors that may occur later in the process. The Analysis will be done with the research question in mind, to guide the exploratory analysis on features, which might contribute to its resolution.

In pursuit of a comprehensive analysis, datasets from both pre- and post-ChatGPT ban periods will be combined. However, the 'payload' column will be excluded from the EDA, given its complex format and the computational limits we face.

The preliminary stage involves scrutinizing the dataset for missing values. The analysis identifies null entries exclusively in the 'org' column, which aligns with the logical premise that an 'actor' may either be part of an organization (indicated by a non-empty value) or not (reflected by an empty value), as shown in Table 2.

Actor	Id	Org	Public	Repo	Type	Language
0	0	5,987,269	0	0	0	0

Table 2: Count of null values in the dataset

Subsequently, the data types in each column are verified to prevent conflicts in later analysis.

Notably, many columns exhibit a nested structure indicative of complex, multi-level data arrangements where each cell can contain a DataFrame.

While inspecting the data types, no abnormalities are observed (Appendix 4: Data types of each variable). The boolean values in the *public* and the *type* columns warrant further examination.

For the *public* column, all entries are *True*, aligning with the dataset's focus on public repositories. Consequently, this column may be disregarded. The *type* column, cataloging various GitHub event types, presents a rich avenue for deeper analysis, given its representation of diverse activities. An analysis of the occurrence frequency of each event type is visually depicted in Figure 8, illustrating a predominance of *PushEvents*, with less frequent occurrences of types like *ReleaseEvent* or *MemberEvents*, indicating varying impacts on the overall analysis.

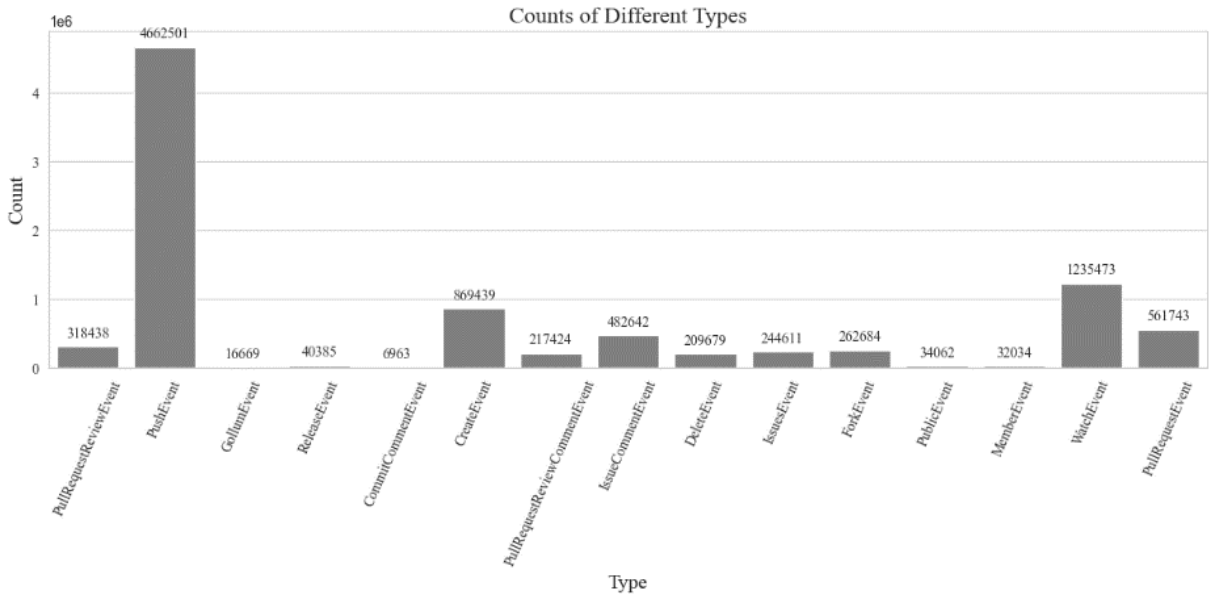

Figure 8: Occurrence of each event type before filtering for control and treatment group

Taking one step back and examining the total number of events per day, which is illustrated in Figure 9, consistent growth trend is visible with weekends distinctly visible. A notable spike on April 3rd, representing 387,023 events, demands further investigation. Nevertheless, this is only a time constrained view of about two weeks, the whole world and only Python repositories, so there could be a multitude of reasons for that notable difference. The red line in Figure 9

symbols the April 1st breakpoint, guiding the study's division of the dataset.

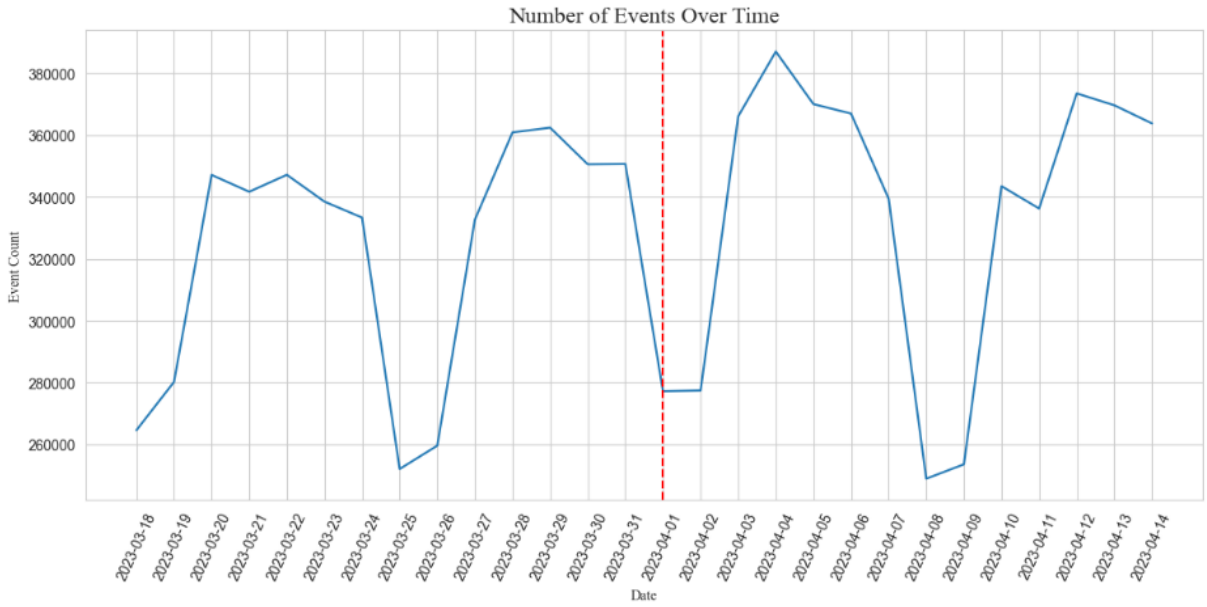

Figure 9: Number of Events over time

In Figure 10 the total number of events is shown for each unique event type. The difference between each type is clearly visible and the data completeness for each type, as there are no line breaks. Moreover, the spike on the 3rd of April can be traced to the *PushEvent* type, as it is the only line with a distinct increase. Lastly, the trend is similar before and after the “red line”, which further proves the comparability.

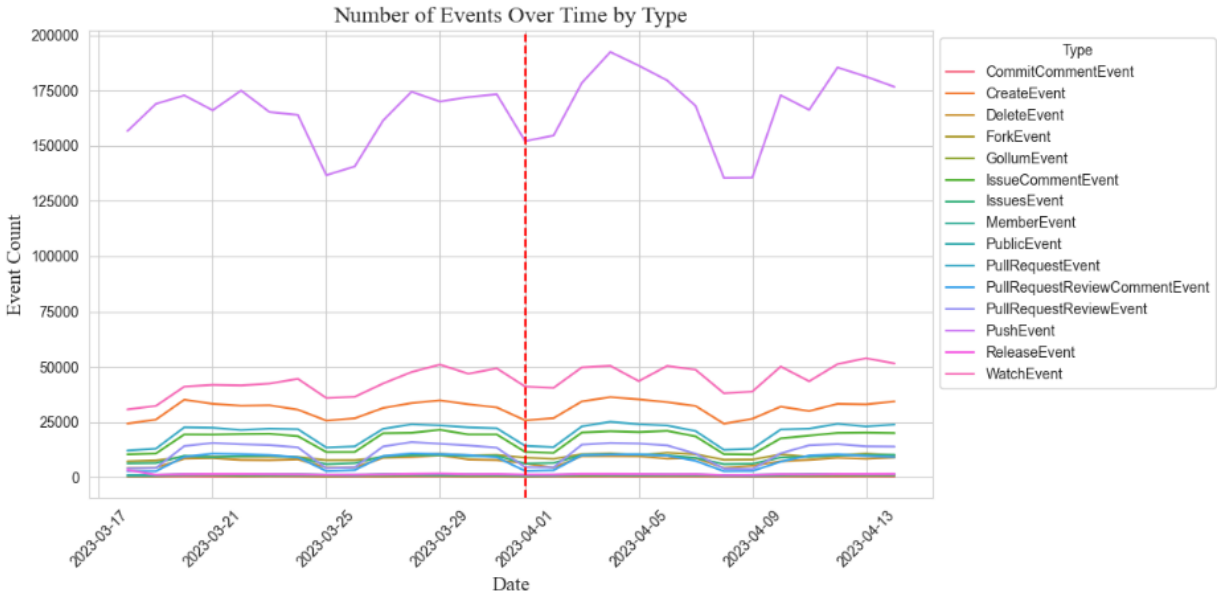

Figure 10: Number of Events over time grouped by the event type

4.2 Scoring and Selection Approach

In this study, the waterfall approach is adopted for its sequential and systematic progression, aligning with our need for precise and orderly data processing (Foremski et al. 2017). This method is particularly beneficial in data processing, as it mimics a cascade where each filtering stage builds upon the output of the preceding one, thus ensuring the integrity and relevance of the dataset. To supplement the GitHub Archive dataset, which lacks direct user location data, we integrate external data points from the GitHub API, enhancing our ability to determine user country of origin.

The first stage of our waterfall process involves pinpointing users based on the location information set in their GitHub profiles, utilizing the GitHub API's '/users/{*username*}' endpoint. In the subsequent stage, we reverse-search users via their email addresses, diverging from traditional username-based methods. The process then advances to a more nuanced stage, where a linguistic analysis of repository and organization descriptions is conducted. This is done to accurately ascertain the contributors and members affiliated with these entities. The analysis leverages two specific GitHub API endpoints (GitHub, n.d.):

> *https://api.github.com/repos/{owner}/{repo}*
> *https://api.github.com/orgs/{org}*

This comprehensive approach ensures a multifaceted analysis, encompassing various dimensions of user activity and association within the GitHub community. Figure 11 illustrates this Waterfall Cascade Process for user identification, including the number of users found in each cascade.

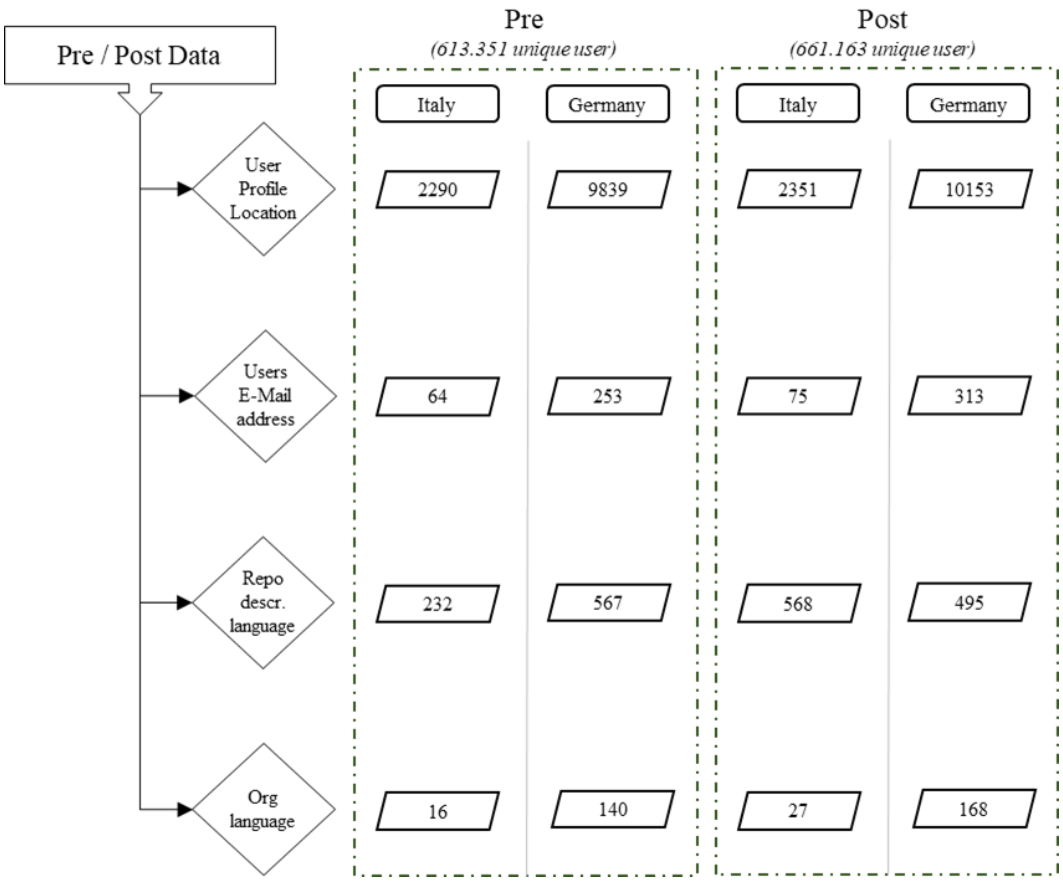

Figure 11: Waterfall Cascade Process for identifying users

Concluding our discussion on the methodology used in this study, it's insightful to compare the waterfall approach with the weighted sum approach, as it would be a suitable approach to filter the data (Mitra & Ozbek 2021). While the weighted sum approach offers flexibility and is effective in scenarios requiring simultaneous evaluation of multiple criteria, it did not align as well with the specific objectives of our study. Our focus was on methodically processing each data layer to identify user origins accurately, which necessitated a more structured approach. The waterfall method's ability to sequentially process and build upon each data layer provided a clearer path to achieving our research goals – accurate identification of a user's origin.

4.3 Location of User

The first step in the waterfall process consists of extracting the geolocation data specified by each user, isolating those users situated within Italy and Germany. A primary challenge encountered was the expansive volume of the dataset, necessitating individual location requests

for users prior to country-specific filtering. The dataset encompassing the period before the ban from the 17th to 31st of March 2023 comprised 613,351 unique authors, while the dataset subsequent to the ban from the 1st to 14th of April 2023, contained 661,163 unique authors.

Simple API requests would quickly exhaust the 5,000-request hourly limit and would have resulted in an incredible long execution time of 5.5 days per period. Therefore, a more efficient approach was required, allowing us to get information faster and more efficiently. As previously mentioned in I.3.2.2, GraphQL is the solution for faster information gathering.

In managing data retrieval through GraphQL, particularly for user location data as demonstrated in Appendix 6: GraphQL Request for User Location, it's essential to first undertake the sanitization of usernames before incorporating them into GraphQL queries. This step is crucial both for preserving the query syntax's integrity and for upholding security standards.

The sanitization process involves replacing any special characters in usernames with standard alphanumeric characters. This precaution is vital to avoid any errors in query formation.

Furthermore, the implementation of error checking mechanisms is paramount in API transactions (Dhanani 2023). These mechanisms detect anomalies like unusual HTTP responses or missing data, allowing for quick resolution and ensuring continuous data flow, which is imperative especially during extensive data retrieval activities.

Alongside these processes, the selection of an appropriate Batch size in data fetch operations must be approached with strategic consideration. An optimal batch size is essential to navigate the equilibrium between request frequency and compliance with API rate limits, ensuring uninterrupted service while avoiding overburdening the API with excessive simultaneous requests (GitHub, n.d.). This careful determination of batch size is thus fundamental to optimizing data retrieval efficiency and aligning with the data source's throughput capacity. Determining the ideal size required a period of calibration, as initial observations indicated an increased probability of errors with larger batch sizes. Especially in the code development stage

a smaller batch size is recommended, as with an imperfect sanitation of usernames and only little error handling, one single user could turn a whole batch into an error. Moreover, manually checking a large batch size for potential errors is tedious and time consuming.

Later the batch size can be increased, to speed up the process, when the code is more robust, and errors have been fixed. Even though there is no explicit limit of the batch size, setting the size too high will result in too many outputs at the same time and therefore could reach computational limits. After thorough testing, a batch size of 2,000 emerged as the most effective balance between performance and stability.

Finally, managing null returns in API operations is essential for maintaining reliable and effective data retrieval. Null returns often indicate that data is unavailable for specific requests, which can be due to various reasons such as privacy settings, access restrictions, or empty user information. Especially in our study, where we look at data from the past, users could have changed their username or simply have deleted their account. Properly handling these scenarios is critical because it allows us to preserve the integrity of the data set, as it ensures that only meaningful and valid data is included for subsequent analysis. This approach not only strengthens the robustness of the study but also supports the generation of reliable insights and results.

In our processing methodology, the initial step is to verify its response status. Should the status code be 200, indicating a successful transfer, we proceed to ensure a payload is in the response. In cases where a payload is detected, it is returned. Conversely, in the absence of data, an error is thrown with the corresponding error code and message (see Appendix 7).

By processing each result individually, we efficiently incorporate all accurately identified users into the final dataset, while excluding those that are not successfully processed. This approach enables us to maintain the accuracy of the dataset, capitalizing on the advantages of batch processing, yet ensuring that only valid and correct data is included for analysis.

Upon the application of the data retrieval methodologies via the GitHub GraphQL API, the outcomes were substantial. Out of 613,351 unique authors, the API provided location information for 211,397, demonstrating the efficacy of our query and fetch techniques in extracting relevant data and highlighting the prevalence of location data among GitHub profiles. The process of pinpointing Italian and German locales, as self-reported by users, necessitated a methodological application of text comparison. For this purpose, the *fuzzywuzzy* Python library was employed, utilizing the Levenshtein Distance metric provided by the Python Software Foundation (2023) to ascertain the extent of similarity between textual strings (Appendix 8). The effectiveness of the library lies in its ability to measure the minimal changes needed to transform one text string into another. It provides a comprehensive range of functions for comparing a string with a collection of others and measuring the degree of similarity between pairs of strings.

For user-generated location data, comparisons were made using a list of Italian city keywords (as delineated in Appendix 8). To enhance the library's performance and ensure data security, a preliminary step of data sanitization was applied. This step was crucial for removing unusual characters and normalizing text formats, thus improving the dataset's consistency, and preventing computational errors in later analysis stages.

The process was repeated for Germany with a specifically tailored list of keywords (Appendix 12: German Keywords). The completed tables containing Italian and German users, along with their locations, are subsequently exported and saved as *.csv* files, facilitating convenient access for future processing.

login	***location***
ewerybody	germany
Letgamer	germany
ctom2	munich Germany
Linus789	gemany
PACTInGame	germany

Table 3: Excerpt of location results for Germany after fuzzywuzzy matching

A crucial aspect of this method was the careful selection of keywords for each country to avoid false positives and ensure no cities were missed. Initially, a comprehensive database from Geonames & Opendatasoft (2023) , listing cities with populations over 1,000, was considered. For Italy the list included 9,948 and for Germany 7,366 unique cities, which theoretically served as a good data foundation. However, due to the high potential for mismatches, particularly with city names common to multiple countries, this approach was deemed less effective.

The study ultimately relied on a custom-created list of keywords combined with the *fuzzywuzzy* string matching method, which yielded better performance and accuracy in identifying user locations. This detailed, methodical approach was instrumental in successfully distinguishing and documenting the geographic distribution of GitHub users from Italy and Germany.

Total user with location data			
211,397			
Italy		Germany	
Pre	Post	Pre	Post
2,290	2,351	9,839	10,153

Table 4: Result Location Analysis

4.4 Email Domains

The initial phase of the research involved extracting email addresses from commits to identify German and Italian users, based on their email domain endings (.de and .it). However, this preliminary method, which relied on matching the actor's name in the row with an email in the payload column, encountered significant inaccuracies. These inaccuracies stemmed from two primary issues:

1. Multiple commits from various users within a single event
2. The actor initiating the event not always being the same as the individual responsible for the commit

As this was not always the case, finding the root cause of this error seemed tedious and difficult.

An additional complication arose due to the absence of user logins in the payload; only the full names of users were provided. This lack of specific identifiers rendered direct matching impracticable without supplementary data gathering. To circumvent this, the GitHub API was employed to retrieve usernames using email addresses, addressing the limitations of direct matching. However, GraphQL did not provide a direct endpoint for this purpose, necessitating the use of an alternative API endpoint.

Moreover, as email addresses are considered personal data, GitHub imposes limitations on the frequency of requests to this relevant API endpoint. While there is no explicit documentation confirming this restriction, the patterns observed in API request responses were indicative. To circumvent these limitations, a strategic implementation of *time.sleep* - a function in Python's time module that pauses the execution of a program for a specified number of seconds - was critical in ensuring the successful execution of requests, as demonstrated in Appendix 9 in the Appendix (Python Software Foundation 2023). By strategically pausing between requests, it helps in adhering to API rate limits and reducing the risk of being blocked or throttled by the API server, ensuring more stable and reliable interactions with the API (see Appendix 9: Getting Username from E-Mail). In instances where the sleep interval was insufficiently set, such as only 1 second, error messages were consistently generated for every request, indicating a breach of some form of limitation. In instances where the sleep interval was insufficiently set, such as only one second, error messages were consistently generated for every request, indicating a breach of some form of limitation. Through repeated trials, it was determined that a sleep interval of four seconds ensured stable request execution while minimizing waiting time. However, this adjustment significantly increased the overall execution duration, thereby impeding rapid data collection (see Appendix 10: Error Messages).

Furthermore, it is imperative to acknowledge another privacy-related constraint. Users have the option to keep their personal email addresses private, choosing instead to utilize a noreply email

address provided by GitHub as their commit email (GitHub, n.d.). This practice, while commendable for safeguarding individual privacy, posed additional challenges for data collection in this research. In addition, users may opt to set their email addresses to private in their GitHub profiles. Consequently, even if a user's commit email is public, it cannot be employed to determine their username, which is a critical element for the join operations integral to this study. This could also account for the relatively low number of users identified in this part of the study.

Italy		Germany	
Pre	*Post*	*Pre*	*Post*
64	75	253	313

Table 5: Results Email Analysis

4.5 Linguistic Analysis of Repo and Organization Description

In part of the research, an analysis of the description of repositories and organizations was used, to identify the country of origin based on the written language in their descriptions. If the descriptions were in German or Italian, associated users were identified and labeled accordingly. This approach needed to be checked carefully, as the repositories and organizations can include a vast amount of users and therefore are prone to false positives.

As the descriptions were not available in the GitHub Archive Data, Grigorik (2023), and are not accessible via BigQuery, the first step was to leverage the GitHub API to obtain the description for each distinct repo and organization.

The second step consists of identifying the language based on the description, which only consists of a short text. For these three different language analysis packages were compared: *lingua-language-detector 2.0.0*, *langdetect* and *fasttext-langdetect* (Cavdar 2023; Danilak 2021; Stahl 2023).

Recent studies provide insights into the performance of these language detection models, especially in the context of short text descriptions. For instance, Gottron & Lipka (2010) found that language detection accuracy on short, query-style texts can reach over 80% for single

words, with slightly longer texts achieving close to 100% accuracy. This is relevant as the descriptions in question are similarly concise. Additionally, Balazevic et al. (2016) emphasized that incorporating personalized user-specific information into language detection algorithms significantly improves detection results for short text messages on social media platforms, a finding that may be applicable to our context of analyzing repository descriptions.

Considering these findings, the comparison of *lingua-language-detector*, *langdetect*, and *fasttext-langdetect* for our specific use case - short descriptions of repositories - should take into account their performance in accurately identifying languages in brief texts. The choice among these models will be influenced by their efficiency and accuracy in handling the nuances and brevity of such descriptions.

Using the Accuracy comparison of Lingua, then Lingua 2.0.0 should perform the best.

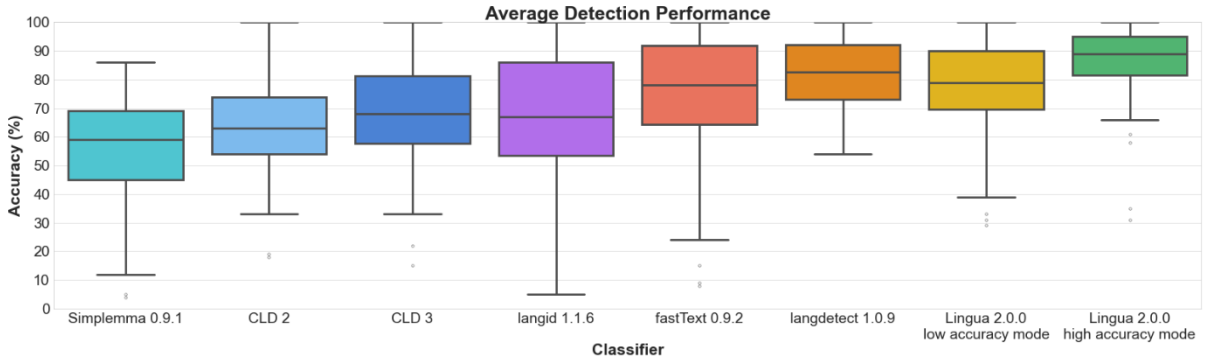

Figure 12: Average detection Performance (Stahl 2023b)

To assess the accuracy of the models, a manual evaluation was conducted by comparing the results of all three models against a subset of the dataset, manually checking their performance to gauge their effectiveness. In the evaluation the optimal method was found by integrating *fasttext* and *lingua*, with a match only accepted when both models concurred. In the subsample of 25 repos and 25 organizations, the combined models achieved an accuracy of 0.98, a precision of 0.977 and a recall of 0.85. Based on this high performance, the method was applied to the entire dataset, yielding the results outlined in the subsequent table.

	REPOSITORY	ORGANIZATIONS
TOTAL DESCRIPTIONS	431,102	28,940
LANGDETECT	5,631	322
FASTTEXT	189	21
LINGUA	165	16
REAL ITALIAN REPOS	97	13

Table 6: Language Analysis Results for Italy in dataset before the ban

An abstract of the results can be found in the Appendix under Appendix 11. It is important to note that in our tests, *lingua* was most effective when set with a minimum relative distance of 0.6. This setting was crucial to ensure accurate identification of Italian repositories, given that *lingua* typically predicts the most likely language for any given text input (Stahl 2023).

4.6 Limitations

In the execution of this study, we encountered several limitations regarding the accurate identification of the nationalities of users and repositories. This section outlines these limitations and provides context for the methodologies employed.

A primary limitation arises from the reliance on user-declared locations. Users manually input their locations on profiles, which can lead to inaccuracies. For example, a user based outside Italy might list an Italian city as their location, thus skewing the data. Furthermore, the methodology of string matching to identify specific keywords also proved to be imperfect. An instance for this imperfection was the identification of "*Syracuse, NY*" as Italian, owing to the existence of a city named Syracuse in Italy. This wrong identification was later rectified through manual removal.

Secondary, the utilization of email addresses to infer nationality also has its shortcomings. It is common for users to operate email addresses registered in Italy or Germany while being based in a different country, leading to potential misclassifications.

Thirdly, the global nature of collaborations in repositories introduces further complexity. For example, repositories tagged as Italian might include collaborators from around the world. This

diversity makes it challenging to ascertain their actual geographical locations based solely on their collaborative activities.

Organizations, like individual users, often have a global footprint, which complicates the process of accurately determining their national origins. The analysis also depends on the public listing of organization members, which is not a universal practice. However, it is generally unlikely for a global organization to have a non-English description, which somewhat mitigates this limitation.

In light of these challenges and the unavailability of more definitive methods, such as IP address tracking, the approach, while not flawless, represents the most feasible method under the given constraints. Post-application of each method, a subset of data is manually reviewed to ensure accuracy. In the assessments of the various methods, no clear false positives were identified in a random subset of 20 entries.

Lastly it is important to note that one particular method was removed from the study due to its unreliability. This approach involved extracting the website or blog details of each user and filtering them based on the domain (“.it” and “.de”). However, it yielded a high rate of false positives, with 9 out of 15 users in a random sample incorrectly identified as residing in Italy according to their profile information. This led to the method's exclusion to preserve the database's accuracy.

To summarize, we have developed a refined dataset through a systematic waterfall filtering method. This approach involved progressively incorporating unique actors at each stage, resulting in a comprehensive dataset that covers both pre-ban and post-ban periods for Italy and Germany. This dataset encompasses users identified using four distinct methods: location information, email domain analysis, linguistic analysis of repository and organizational descriptions. The final output comprises the following elements:

	Italy		Germany	
	Pre	*Post*	*Pre*	*Post*
# events	18,837	22,780	10,3576	99,206
# user	2,427	2,650	10,248	10,520
# repos	3,501	4,092	13,707	14,188
# orgs	873	947	3,100	3,137
# users in orgs	905	981	4,600	4,488
# users not in orgs	1,522	1,669	5,648	6,032

Table 7: Final Result of the Italian and German Dataset

5. Overall Results Discussion

The following chapter summarises and analyses the findings per variable across the entire work, placing them within a broader context and discussing their implications beyond the individual hypotheses tested in the previous work. Particular emphasis will be placed on the results that are statistically significant.

5.1 Collaboration

In examining the impact of ChatGPT's ban on collaborative efforts among software developers, the analysis reveals significant coefficients indicating a change in collaboration levels. However, the violation of parallel trends across all collaboration-related regressions suggests caution in attributing these changes directly to the ban. Even after the subsample analysis, we cannot confirm a causal relationship between the ban of ChatGPT and collaboration activities among software developers. More generally, we cannot find statistical evidence, that the introduction of ChatGPT influenced collaboration among software developers. Collaboration might be part of a more complex array of influences not captured by the study. Therefore, while AI innovations like ChatGPT represent a shift in coding practices, their direct impact on collaborative behaviors on platforms like GitHub is not confirmed the presented analysis. The absence of a clear causal link between ChatGPT's availability and changes in collaboration among developers indicates that the integration of AI tools into development practices is complex and subject to various factors. While AI innovations like ChatGPT may change how developers work together, these changes are not solely driven by AI tools but also by the intricate dynamics of the software development process. This complexity might require a deeper exploration of how AI tools are adopted and adapted within the quickly evolving landscape of software development.

5.2 Code Quality

In the broader context of our analysis, the impact of the ChatGPT ban on code quality among software developers presents a nuanced picture. Initially, the results for the full sample did not allow to infer causality due to the violation of parallel trends, thereby suggesting that the ban's effect was not universal across all developers. Similar results were obtained for the subsample focusing on users with organisational membership. However, a more detailed look at the subsample of developers active on business days revealed a different story. On business days, coding errors increased by 8.59%, with this increase being statistically significant at the 5% level ($p < 0.05$). We therefore find statistical evidence to believe that the ban of ChatGPT decreased code quality among Italian software developers which were active during the standard work week. The results got even more pronounced in the following subsample, which only included organisation-related users on business days. The number of error-related commits increased by 20.06% ($p < 0.001$) after restricting access to ChatGPT, implying a significant decrease in code quality.

Following the assumption, that organization-related users and those active on business days are more likely to be professional software developers, the restriction of ChatGPT appears to have had a more pronounced negative impact on their code quality. This suggests that professionals may rely more heavily on AI tools for tasks like debugging and code review. The absence of ChatGPT likely resulted in the need for more issue reporting and commit reversals, indicating challenges in maintaining code standards without AI assistance. In contrast, individual or non-professional users, who may not follow strict development protocols, showed less impact from the ban, potentially due to diverse coding practices and less dependence on AI for code quality. This trend extends beyond GitHub, reflecting a broader shift in the software development industry towards integrating AI for critical tasks like debugging and code review. The differential impact observed between professional and individual users indicates a varied

reliance on AI tools across the software development spectrum, highlighting the need for adaptable AI integration strategies catering to different levels of expertise and project requirements.

5.3 Code Efficiency

Analyzing the impact of ChatGPT's ban on coding efficiency among software developers, the overall results do not support a significant change in coding efficiency, as evidenced by the lack of significant coefficients. This outcome suggests that the ban of ChatGPT did not lead to observable differences in coding efficiency among the broader group of Italian software developers on GitHub. However, examining the subsamples, particularly those focusing on organization-related users active on business days, provides further insights. For organizations, the ban of ChatGPT in Italy is associated with a statistically significant increase in GitHub actions of approximately 12.12% ($p < 0.001$). An increase in the number of actions indicates a reduction in coding efficiency. Furthermore, for the subsample containing only organisation-affiliated users on business days, we find a statistically significant increase in the number of GitHub actions of roughly 13.39% following the ban ($p < 0.001$), also indicating a reduction in coding efficiency. This implies that in more structured professional environments, where workflows are likely more reliant on AI tools like ChatGPT for efficient coding practices, the absence of such tools can have a noticeable impact.

The contrast between the general and subsample findings indicates that the effect of AI tools on coding efficiency may vary depending on the context and nature of software development activities. While the broader developer community may not have shown a dependence on ChatGPT for efficiency, professional environments, particularly in organizational settings, exhibit a reliance on such tools. This suggests that AI tools are becoming integral to certain sectors of software development, particularly where efficiency and time management are crucial. The findings highlight the evolving role of AI in software development, pointing

towards a future where AI integration may become essential in certain professional domains to maintain high levels of productivity and efficiency.

6. Conclusion

As we approach the conclusion of this research, it is evident that the advancement of artificial intelligence, particularly in the form of tools like ChatGPT, has significantly altered everyday life, with a marked impact on software development practices. This research provides new insights in the field of AI-driven software development by exploring a potential causal relationship between the introduction of ChatGPT and software development practices. The study employed a DID analysis, more precisely a two-way fixed effects model, to analyze the impact of Italy's ChatGPT ban on GitHub activity over a four-week period from 17/03/2023 to 14/04/2023. A dataset spanning 244,401 commits from 10,520 individual users contrasts Italian developers (Treatment group) with German ones (Control group) to quantify the ban's effect, controlling for entity and time fixed effects. For the purpose of this study, software development practices are separated into three main categories: Collaboration between developers, code quality and coding efficiency. Leveraging GitHub's API and GraphQL, additional data from Python repositories was gathered to identify Italian and German users, offering a unique perspective on AI's role in enhancing software development, especially in the context of collaboration and code quality. Technical resources like the creation of a cloud server and architecture allowed us to process large amounts of data effectively. Once the available data was obtained, it was sorted into the Control and Treatment group. Several methods were applied in a Waterfall Cascade Process, and later language detector machine learning models, to ensure proper division of groups and filter out any non-python commits, finally resulting in the main dataset used to conduct this research.

6.1 Main Results

This study reveals that ChatGPT's influence on GitHub user activity and software development more generally, is context-dependent: it does not seem to significantly affect overall collaboration practices on GitHub but appears important for code quality and efficiency in

professional settings, suggesting that such tools have become embedded in the workflow and practices of professional developers. Comparing the lack of empirical evidence for subgroups related to individual GitHub users (i.e., users without organisational affiliation and with activity on weekends and public holidays) as opposed to significant effects found for user groups which are likely related to organisations (i.e., users affiliated with organisations and activity on business days), highlights AI's growing but varied role, even within software development. As such, the rise of ChatGPT and similar AI-driven tools presents both an opportunity and a challenge for the evolution of software development practices. In detail, this thesis presents the following results:

RQ1 - RQ3: We do not find empirical evidence, that the introduction of ChatGPT significantly affects collaboration, code quality or coding efficiency among software developers in the full sample. We cannot infer causality between a change in software development practices and the restriction of ChatGPT in Italy and there is reason to believe that other unobserved factors might have influenced the results.

RQ4 - RQ5: With regards to a subset of users that belong to organisations, however, results indicate a causal relationship between the ban of ChatGPT and coding efficiency. On average, users affiliated with organisations show a 12.12% ($p < 0.001$) increase in GitHub events, implying a decrease in coding efficiency after the ban of ChatGPT, while for individual users no significant effect is found. Similarly, regarding development activity on business days, we find an 8.91% ($p < 0.01$) increase in coding errors after the introduction of the ban, indicating a decrease in code quality, as opposed to no significant effect for weekends and public holidays.

***RQ6*:** Finally, for a subgroup of only organization-affiliated software developers which are active on business days, we find a statistically significant increase in coding errors ($20.6\%, p < 0.001$) and GitHub actions ($13.39\%, p < 0.001$) for Italian software developers after the

introduction of the ChatGPT ban, which implies a significant decrease in code quality and coding efficiency.

6.2 Limitations and Future Research

The results of this study are subject to several limitations. Firstly, the data analysed for this work only comprises a small subset of public GitHub repositories in the python programming language. This is due to the sheer volume of data available and technological limitations regarding data storage and handling. To allow for better generalisation across the entire domain of software development, future research could focus on a broader range of programming languages or compare a potential differential impact of ChatGPT on different programming languages. Further, since only public repositories can be analysed, this leaves a big part of the GitHub activity in the dark. Thus, in the case of companies, more sensitive information may be held in private repositories, in the interests of data protection.

Another caveat lies in the possibility for users to circumvent the ban by using VPNs or servers with foreign IP addresses, which could result in a diminished effect of the ban (Kreitmeir and Raschky 2023). Considering that Google's Bard was launched a short time before the ban, it is also possible to expect some developers already had the option to switch to alternative services for AI coding support. However, this impact was potentially limited, as it was initially only offered to a limited group of users and only introduced to the wider public in the European Union on July 13th, which is after the analysed period of the ban (Deutsch 2023).

Additionally, the study faces limitations in accurately identifying user nationalities and repositories, primarily due to the reliance on user-declared locations, which can be inaccurate or misleading. String matching techniques for location identification and using email addresses to infer nationality can lead to misclassifications. Future research could therefore incorporate more advanced methods for nationality identification, such as IP address tracking or having

designated control and treatment groups, to enhance the accuracy of geographical categorization.

Further, the short period of four weeks, inherent to the nature of the event, may not allow to fully capture the long-term effects and trends of ChatGPT's impact on software development practices. With software development cycles usually spanning four weeks, future research could cover a longer time frame, covering multiple sprint cycles, to provide a richer understanding of the topic (Stephanie Ockermann 2023).

Lastly, incorporating qualitative methods such as surveys and interviews could also provide deeper insights into developers' perspectives on AI integration in their work, which may not be captured by quantitative data alone.

6.3 Concluding Remarks

In conclusion, this thesis contributes a foundational understanding of the impact of AI tools like ChatGPT on software development, offering insights into their influence on various aspects such as collaboration, code quality, and coding efficiency. Especially adding to existing work by Kreitmeir and Raschky (2023), who find that the output of Italian developers on GitHub decreased by around 50% in the first two business days after the ban but recovered after that, we expanded the analysis to further areas of interest. Our findings reveal a context-dependent impact of ChatGPT, with a notable emphasis on its crucial role in enhancing code quality in professional settings, while its influence on collaboration and coding efficiency appears more nuanced. This highlights the evolving importance of AI in structured, professional environments within the broader software development sphere. Additionally, the study acknowledges its limitations, including the challenges in interpreting GitHub data as noted by Kalliamvakou et al. (2014), which underscores the complexity of data analysis in this domain. These limitations serve as a catalyst for future research, suggesting a need for more in-depth, comprehensive methodologies to fully understand how AI technologies like ChatGPT reshape

software development practices. This research therefore not only contributes to the existing body of knowledge but also paves the way for future studies to delve deeper into the transformative role of AI in software development.

III. References

-, Mohammad Ikbal Hossain. 2023. “Software Development Life Cycle (SDLC) Methodologies for Information Systems Project Management.” *International Journal For Multidisciplinary Research* 5 (5). https://doi.org/10.36948/ijfmr.2023.v05i05.6223.

Amin, Mostafa M., Erik Cambria, and Björn W. Schuller. 2023. “Will Affective Computing Emerge from Foundation Models and General AI? A First Evaluation on ChatGPT,” March. http://arxiv.org/abs/2303.03186.

Balazevic, Ivana, M. Braun, and K. Müller. 2016. “Language Detection For Short Text Messages In Social Media.” *ArXiv* abs/1608.08515 (August). https://doi.org/.

Balta-Ozkan, Nazmiye, Oscar Amerighi, and Benjamin Boteler. 2014. “A Comparison of Consumer Perceptions towards Smart Homes in the UK, Germany and Italy: Reflections for Policy and Future Research.” *Technology Analysis & Strategic Management* 26 (10): 1176–95. https://doi.org/10.1080/09537325.2014.975788.

Barua, Anton, Stephen W. Thomas, and Ahmed E. Hassan. 2014. “What Are Developers Talking about? An Analysis of Topics and Trends in Stack Overflow.” *Empirical Software Engineering* 19 (3): 619–54. https://doi.org/10.1007/s10664-012-9231-y.

Bazzani, Tania. 2017. “Italy, Denmark and Germany: A Comparative Analysis in Active and Passive Labour Market Policies.” *European Labour Law Journal* 8 (November): 133–53. https://doi.org/10.1177/2031952517712124.

Bhattacharjee, Avijit, Sristy Sumana Nath, Shurui Zhou, Debasish Chakroborti, Banani Roy, Chanchal K. Roy, and Kevin Schneider. 2020. “An Exploratory Study to Find Motives Behind Cross-Platform Forks from Software Heritage Dataset.” In *Proceedings of the 17th International Conference on Mining Software Repositories*, 11–15. New York, NY, USA: ACM. https://doi.org/10.1145/3379597.3387512.

Borges, Hudson, Andre Hora, and Marco Tulio Valente. 2016. “Understanding the Factors That Impact the Popularity of GitHub Repositories.” In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 334–44. IEEE. https://doi.org/10.1109/ICSME.2016.31.

Braga, Pedro Henrique Pereira, Katherine Hébert, Emma J. Hudgins, Eric R. Scott, Brandon P. M. Edwards, Luna L. Sánchez Reyes, Matthew J. Grainger, et al. 2023. “Not Just for Programmers: How GitHub Can Accelerate Collaborative and Reproducible Research in Ecology and Evolution.” *Methods in Ecology and Evolution* 14 (6): 1364–80. https://doi.org/10.1111/2041-210X.14108.

Brito, Gleison, and Marco Tulio Valente. 2020. “REST vs GraphQL: A Controlled Experiment.” In *Proceedings - IEEE 17th International Conference on Software Architecture, ICSA 2020*, 81–91. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ICSA47634.2020.00016.

Cavdar, Zafer. 2023. “Fasttext-Langdetect 1.0.5.” January 9, 2023. https://pypi.org/project/fasttext-langdetect/.

Chen, Nuo, Yan Wang, Haiyun Jiang, Deng Cai, Yuhan Li, Ziyang Chen, Longyue Wang, and Jia Li. 2022. “Large Language Models Meet Harry Potter: A Bilingual Dataset for Aligning Dialogue Agents with Characters,” November. http://arxiv.org/abs/2211.06869.

Cheng, Xin. 2023. “Generative AI in Software Development.” 2023.

Cunningham, Scott. 2021. *Causal Inference: The Mixtape*. Yale University Press. http://www.jstor.org/stable/j.ctv1c29t27.

Daityari, Shaumik. 2016. “Quick Tip: Sync a GitHub Fork via the Command Line.” 2016.

Danilak, Michal. 2021. “Langdetect 1.0.9.” May 7, 2021. https://pypi.org/project/langdetect/.

Deutsch, Jillian. 2023. “Google’s AI Chatbot Bard Gets Belated European Release.” Bloomberg. July 13, 2023. https://www.bloomberg.com/news/articles/2023-07-13/google-s-ai-chatbot-bard-gets-belated-european-release.

Dhanani, Error. 2023. “Error Handling in GraphQL | Postman Open Technologies Docs.” March 15, 2023. https://learning.postman.com/open-technologies/blog/graphql-error-handling/.

Euaa. 2022. “Study on Language Assessment for Determination of Origin of Applicants for International Protection - Executive Summary.”

Fahmideh, Mahdi, Aakash Ahmad, Ali Behnaz, John Grundy, and Willy Susilo. n.d. “Software Engineering for Internet of Things: The Practitioners’ Perspective.”

Farré, Carles, Jovan Varga, and Robert Almar. 2019. “GraphQL Schema Generation for Data-Intensive Web APIs.” *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11815 LNCS: 184–94. https://doi.org/10.1007/978-3-030-32065-2_13.

Federal Foreign Office. 2023. “Germany and Italy: Bilateral Relations.” October 5, 2023. https://www.auswaertiges-amt.de/en/aussenpolitik/laenderinformationen/italien-node/italy/227948.

Flint, Samuel W., Jigyasa Chauhan, and Robert Dyer. 2022. “Pitfalls and Guidelines for Using Time-Based Git Data,” September. http://arxiv.org/abs/2209.04511.

Foremski, Paweł, Christian Callegari, and Michele Pagano. 2017. “Waterfall Traffic Classification: A Quick Approach to Optimizing Cascade Classifiers.” *Wireless Personal Communications* 96 (4): 5467–82. https://doi.org/10.1007/S11277-016-3751-5.

Fu, Quchen, Zhongwei Teng, Marco Georgaklis, Jules White, and Douglas C. Schmidt. 2023. “NL2CMD: An Updated Workflow for Natural Language to Bash Commands Translation,” February. https://doi.org/10.13052/jmltapissn.2023.002.

Garante per la protezione dei dati personali. 2023. “Artificial Intelligence: Stop to ChatGPT by the Italian SA Personal Data Is Collected Unlawfully, No Age Verification System Is in Place for Children.” 2023. https://www.garanteprivacy.it/web/guest/home/docweb/-/docweb-display/docweb/9870847#english.

Geonames, and Opendatasoft. 2023. “Geonames - All Cities with a Population > 1000 — Opendatasoft.” November 10, 2023. https://public.opendatasoft.com/explore/dataset/geonames-all-cities-with-a-population-1000/table/?disjunctive.cou_name_en&sort=name.

GitHub. n.d.-a. “GitHub GraphQL.” Accessed November 7, 2023. https://docs.github.com/de/graphql.

———. n.d.-b. “GitHub REST API Documentation.” Accessed November 7, 2023. https://docs.github.com/en/rest?apiVersion=2022-11-28.

———. n.d.-c. “Setting Your Commit Email Address - GitHub Docs.” Accessed November 17, 2023. https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-email-preferences/setting-your-commit-email-address.

Goodman-Bacon, Andrew. 2021. “Difference-in-Differences with Variation in Treatment Timing.” *Journal of Econometrics* 225 (2): 254–77. https://doi.org/10.1016/j.jeconom.2021.03.014.

Gottron, Thomas, and Nedim Lipka. 2010. “A Comparison of Language Identification Approaches on Short, Query-Style Texts.” *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5993 LNCS: 611–14. https://doi.org/10.1007/978-3-642-12275-0_59.

Grigorik, Ilya. 2023. “GH Archive.” Https://Www.Gharchive.Org/. November 5, 2023.

Gurung, Gagan, Rahul Shah, and Dhiraj Prasad Jaiswal. 2020. “Software Development Life Cycle Models-A Comparative Study.” *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, July, 30–37. https://doi.org/10.32628/cseit206410.

Hou, Xinyi, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. “Large Language Models for Software Engineering: A Systematic Literature Review,” August. http://arxiv.org/abs/2308.10620.

Kalla, Dinesh, and Sivaraju Kuraku. 2023. “Study and Analysis of Chat GPT and Its Impact on Different Fields of Study.” *International Journal of Innovative Science and Research Technology*. Vol. 8. www.ijisrt.com.

Kalliamvakou, Eirini, Leif Singer, Georgios Gousios, Daniel M. German, Kelly Blincoe, and Daniela Damian. 2014. “The Promises and Perils of Mining GitHub.” In *11th Working Conference on Mining Software Repositories, MSR 2014 - Proceedings*, 92–101. Association for Computing Machinery, Inc. https://doi.org/10.1145/2597073.2597074.

Komorowski, Matthieu, Dominic C. Marshall, Justin D. Salciccioli, and Yves Crutain. 2016. “Exploratory Data Analysis.” *Secondary Analysis of Electronic Health Records*, January, 185–203. https://doi.org/10.1007/978-3-319-43742-2_15.

Kreitmeir, David, and Paul A Raschky. 2023. “The Unintended Consequences of Censoring Digital – Evidence from Italy’s ChatGPT Ban.” https://doi.org/http://dx.doi.org/10.2139/ssrn.4422548.

Kukhnavets, Pavel. 2019. “7 Stages of Software Development Life Cycle.” October 8, 2019. https://welldoneby.com/blog/7-stages-of-software-development-life-cycle/.

Kuzman, Taja, Igor Mozetič, and Nikola Ljubešić. 2023. “ChatGPT: Beginning of an End of Manual Linguistic Data Annotation? Use Case of Automatic Genre Identification,” March. http://arxiv.org/abs/2303.03953.

Latinovic, Milan, and Viktoria Pammer-Schindler. 2021. “Automation and Artificial Intelligence in Software Engineering: Experiences, Challenges, and Opportunities.” In *Proceedings of the Annual Hawaii International Conference on System Sciences*, 2020-January:146–55. IEEE Computer Society. https://doi.org/10.24251/hicss.2021.017.

Lawi, Armin, Benny L.E. Panggabean, and Takaichi Yoshida. 2021a. “Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System.” *Computers 2021, Vol. 10, Page 138* 10 (11): 138. https://doi.org/10.3390/COMPUTERS10110138.

———. 2021b. “Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System.” *Computers 2021, Vol. 10, Page 138* 10 (11): 138. https://doi.org/10.3390/COMPUTERS10110138.

Liu, Yiheng, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, et al. 2023. “Summary of ChatGPT-Related Research and Perspective Towards the Future of Large Language Models,” April. https://doi.org/10.1016/j.metrad.2023.100017.

Maruping, Likoebe M., and Sabine Matook. 2020. “The Evolution of Software Development Orchestration: Current State and an Agenda for Future Research.” *European Journal of Information Systems*. Taylor and Francis Ltd. https://doi.org/10.1080/0960085X.2020.1831834.

Megahed, Fadel M., Ying-Ju Chen, Joshua A. Ferris, Sven Knoth, and L. Allison Jones-Farmer. 2023. “How Generative AI Models Such as ChatGPT Can Be (Mis)Used in SPC Practice, Education, and Research? An Exploratory Study,” February. https://doi.org/10.1080/08982112.2023.2206479.

Mikuła, Mateusz, and Mariusz Dzieńkowski. 2020. “Comparison of REST and GraphQL Web Technology Performance.” *Journal of Computer Sciences Institute* 16 (September): 309–16. https://doi.org/10.35784/JCSI.2077.

Mitra, Tapan, and Kemal Ozbek. 2021. “Ranking by Weighted Sum.” *Economic Theory* 72 (2): 511–32. https://doi.org/10.1007/S00199-020-01305-W/METRICS.

NationMaster.com. 2023. “Compare Key Data on Germany & Italy.” 2023. https://www.nationmaster.com/country-info/compare/Germany/Italy.

Nick Huntington-Klein. 2019. “Library of Statistical Techniques (LOST).” Https://Lost-Stats.Github.Io/Model_Estimation/Research_Design/Event_study.Html. 2019.

Notermans, Ton, and Simona Piattoni. 2021. “Italy and Germany: Incompatible Varieties of Europe or Dissimilar Twins?” *German Politics* 30 (3): 319–39. https://doi.org/10.1080/09644008.2021.1890717.

Operating systems. 2017. “Example GitHub Repository.” 2017. https://github.com/uu-os-2018/example-repository.

Patrick, Peter. 2012. “Language Analysis For Determination Of Origin: Objective Evidence For Refugee Status Determination.” In . https://doi.org/10.1093/oxfordhb/9780199572120.013.0039.

Philip H. Henning. 2013. “Experimental Research Methods.” Edited by David Jonassen and Marcy Driscoll. *The Handbook of Research for Educational Communications and Technology*, January, 1007–29. https://doi.org/10.4324/9781410609519-51.

Privacy Laws & Business. 2023. “Italy’s Garante Imposes and Then Withdraws a Ban on ChatGPT.” May 3, 2023. https://www.privacylaws.com/news/italy-s-garante-imposes-and-then-withdraws-a-ban-on-chatgpt/.

Python Software Foundation. 2023a. “Asyncio — Asynchronous I/O.” 2023. https://docs.python.org/3/library/asyncio.html.

———. 2023b. “Fuzzywuzzy.” 2023. https://pypi.org/project/fuzzywuzzy/.

———. 2023c. “Time — Time Access and Conversions — Python 3.12.0 Documentation.” 2023. https://docs.python.org/3/library/time.html.

Rahmaniar, Wahyu. 2023. “ChatGPT for Software Development: Opportunities and Challenges.” Kanagawa. https://doi.org/10.36227/techrxiv.23993583.v1.

Ray, Partha Pratim. 2023. “ChatGPT: A Comprehensive Review on Background, Applications, Key Challenges, Bias, Ethics, Limitations and Future Scope.” *Internet of Things and Cyber-Physical Systems*. KeAi Communications Co. https://doi.org/10.1016/j.iotcps.2023.04.003.

Roumeliotis, Konstantinos I., and Nikolaos D. Tselikas. 2023. “ChatGPT and Open-AI Models: A Preliminary Review.” *Future Internet*. MDPI. https://doi.org/10.3390/fi15060192.

Schwerdt, Guido, and Ludger Woessmann. 2020. “Empirical Methods in the Economics of Education.” *The Economics of Education: A Comprehensive Overview*, January, 3–20. https://doi.org/10.1016/B978-0-12-815391-8.00001-X.

Shaw, Minitutorial Mary. n.d. “Writing Good Software Engineering Research Papers.”

Sheikh, Haroon, Corien Prins, and Erik Schrijvers. n.d. “Mission AI Research for Policy.”

Shin, Terence. 2020. “An Extensive Step by Step Guide to Exploratory Data Analysis.” Towards Data Science. January 2020. https://towardsdatascience.com/an-extensive-guide-to-exploratory-data-analysis-ddd99a03199e.

Snow, John. 1855. “Mode of Communication of Cholera. By John Snow, MD: Second Edition - London, 1855, Pp 162.” *International Journal of Epidemiology*. https://doi.org/10.1093/ije/dyt193.

Son, Jungha, and Boyoung Kim. 2023. “Trend Analysis of Large Language Models through a Developer Community: A Focus on Stack Overflow.” *Information* 14 (11): 602. https://doi.org/10.3390/info14110602.

Stahl, Peter. 2023a. “Lingua.” October 25, 2023. https://pemistahl.github.io/lingua-py/lingua.html.

———. 2023b. “Lingua-Language-Detector 2.0.0.” November 15, 2023. https://pypi.org/project/lingua-language-detector/.

Stephanie Ockermann. 2023. “How to Choose the Right Sprint Length in Scrum.” Scrum.Org . May 2, 2023. https://www.scrum.org/resources/blog/how-choose-right-sprint-length-scrum.

Teubner, Timm, Christoph M. Flath, Christof Weinhardt, Wil van der Aalst, and Oliver Hinz. 2023. “Welcome to the Era of ChatGPT et al.: The Prospects of Large Language Models.” *Business and Information Systems Engineering*. Springer Gabler. https://doi.org/10.1007/s12599-023-00795-x.

Treude, Christoph. 2023. “Navigating Complexity in Software Engineering: A Prototype for Comparing GPT-n Solutions,” January. http://arxiv.org/abs/2301.12169.

Vazquez-Ingelmo, Andrea, Juan Cruz-Benito, and Francisco J. García-Penalvo. 2017. “Improving the OEEU’s Data-Driven Technological Ecosystem’s Interoperability with GraphQL.” *Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality* Part F132203 (October). https://doi.org/10.1145/3144826.3145437.

Vrontis, Demetris, Alberto Mazzoleni, Alberto Ferraris, and Elisa Giacosa. 2018. “A Model for Testing the Relationship between Companys Size and Performance: A Cross Country Analysis.” *Global Business and Economics Review* 20 (November): 1. https://doi.org/10.1504/GBER.2018.10006977.

worlddata.info. 2014. “German Speaking Countries.” December 2014. https://www.worlddata.info/languages/german.php.

Xia, Chunqiu Steven, and Lingming Zhang. 2023. “Conversational Automated Program Repair,” January. http://arxiv.org/abs/2301.13246.

Xu, Yongjun, Xin Liu, Xin Cao, Changping Huang, Enke Liu, Sen Qian, Xingchen Liu, et al. 2021. “Artificial Intelligence: A Powerful Paradigm for Scientific Research.” *Innovation*. Cell Press. https://doi.org/10.1016/j.xinn.2021.100179.

Zagalsky, Alexey, Joseph Feliciano, Margaret-Anne Storey, Yiyun Zhao, and Weiliang Wang. 2015. “The Emergence of GitHub as a Collaborative Platform for Education.” In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, 1906–17. New York, NY, USA: ACM. https://doi.org/10.1145/2675133.2675284.

Zampetti, Fiorella, Simone Scalabrino, Rocco Oliveto, Gerardo Canfora, and Massimiliano Di Penta. 2017. “How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines.” In *IEEE International Working Conference on Mining Software Repositories*, 334–44. IEEE Computer Society. https://doi.org/10.1109/MSR.2017.2.

Zhang, Bowen, Daijun Ding, and Liwen Jing. 2022. “How Would Stance Detection Techniques Evolve after the Launch of ChatGPT?,” December. http://arxiv.org/abs/2212.14548.

IV. Appendix

```
formula_standard = f'l_collab ~ Italy_X_After_ban + EntityEffects + TimeEffects'

model_standard = lm.PanelOLS.from_formula(formula_standard, panel)

standard_clfe = model_standard.fit(cov_type = 'clustered',

cluster_entity = True)

standard_clfe.summary
```

Appendix 1: Fitting the PanelOLS Model at the example of l_collab

```
all_days = list(range(1, 29))
all_days.remove(14)
days_formula = ' + '.join(f'IXD{day}' for day in all_days)

formula_dynamic = f'collab_log ~ {days_formula} + EntityEffects + TimeEffects'
model_dynamic = lm.PanelOLS.from_formula(formula_dynamic, panel_dynamic)
dynamic_clfe = model_dynamic.fit(cov_type = 'clustered',
cluster_entity = True)

dynamic_clfe.summary
```

Appendix 2: Regression Formula in Python

```
# Required Libraries

from google.cloud import bigquery

from google.oauth2 import service_account


# Setting up the credentials - credentials is defined beforehand

client = bigquery.Client(credentials=credentials)


# Crafting the SQL query to extract repositories with Python as the primary language

sql = """

SELECT *

FROM `bigquery-public-data.github_repos.languages`

WHERE ARRAY_LENGTH(language) > 0 AND language[SAFE_OFFSET(0)].name = 'Python'

"""

# Querying the data

bigquery_df = client.query(sql).to_dataframe()
```

Appendix 3: BigQuery to get Python Libraries

Actor	*avatar_url*	string
	display_login	string
	gravatar_id	string
	id	bigInt
	login	string
	url	string
Id		string
Org	*avatar_url*	string
	gravatar_id	string
	id	bigInt
	login	string
	url	string
public		boolean
repo	*Id*	bigInt
	name	string
	url	string
type		string
language		string

Appendix 4: Data types of each variable

```
import aiohttp
import asyncio
import pandas as pd
import re
import nest_asyncio
import time

# Enable asyncio in Jupyter Notebook
nest_asyncio.apply()

def sanitize_for_alias(repo_name, owner):
    """Sanitize repository name and owner to create a valid GraphQL alias, with fixed logic."""
    combined = repo_name + "_" + owner
    # Replace hyphens and underscores with distinct sequences to avoid conflicts
    combined = re.sub(r'[-]', '__', combined)
    combined = re.sub(r'[_]', '___', combined)
    # Encode non-ASCII characters as Unicode code points in a valid format for GraphQL
    sanitized = ''.join(f'U{ord(ch):X}' if not ch.isalnum() else ch for ch in combined)
    # Ensure the alias starts with an underscore or a character
    if sanitized and sanitized[0].isdigit():
        sanitized = '_' + sanitized
    return sanitized

def construct_repo_query(repo_names_batch):
    query_parts = []
    for item in repo_names_batch:
        if isinstance(item, (list, tuple)) and len(item) == 2:
            repo_name, owner = item
            # Construct query part
            query_part = f'''
            {sanitize_for_alias(repo_name, owner)}: repository(owner: "{owner}", name:
"{repo_name}") {{
                primaryLanguage {{
                    name
                }}
            }}
            '''
            query_parts.append(query_part)
        else:
            print(f"Invalid item format: {item}")  # Add a debug message for invalid items
```

```
    return '{' + ''.join(query_parts) + '}'

# Function to fetch data using a constructed query
async def fetch_data(query, tokens, session, token_index, max_retries=3):
    headers = {
        'Content-Type': 'application/json',
        'Authorization': f'bearer {tokens[token_index]}'  # Use token based on token_index
    }
    backoff_time = 2  # Backoff time in seconds
    for attempt in range(max_retries):
        try:
            async with session.pre('https://api.github.com/graphql', json={'query': query},
headers=headers) as response:
                # Check response status
                if response.status == 200:
                    response_json = await response.json()
                    #print(response_json)
                    return response_json.get('data', {})
                else:
                    response_json = await response.text()  # Get the response text for non-200
responses
                    print(f"Non-200 response: {response.status} message: {response_json}")
                    await asyncio.sleep(backoff_time)  # Wait before the next retry
                    backoff_time *= 2  # Increase backoff time exponentially
        except aiohttp.ClientError as e:
            print(f"Request failed with error: {e}")
            await asyncio.sleep(backoff_time)  # Wait before the next retry
            backoff_time *= 2  # Increase backoff time exponentially
    print("Max retries reached, returning empty data.")
    return {}  # Return an empty dict after max retries

# Function to fetch repository data for a batch of repository names
async def fetch_repo_data(repo_names_batch, session, tokens, token_index):
    query = construct_repo_query(repo_names_batch)
    data = await fetch_data(query, tokens, session, token_index)
    return data

# Asynchronous main function to run the event loop
async def main(pre_parts, n, batch_size, tokens):
    pre_repo_languages_all = {}  # Dictionary to store DataFrames
    async with aiohttp.ClientSession() as session:  # Create a session
        for i in range(n):
            start_time = time.time()
            print(f"Processing Main Batch: {i + 1}/{n}")
            token_index = 0  # Initialize token index for token rotation
            batch_data = []  # Initialize batch_data as an empty list
            language_list = []

            for j in range(0, len(pre_parts[i]), batch_size):
                sub_batch_number = (j // batch_size + 1)
                if sub_batch_number % 25 == 0:
                    print(f"Processing Sub-batch: {j // batch_size + 1}")
                sub_batch = pre_parts[i][j:j + batch_size]
                # Fetch data for the sub-batch
                batch_result = await fetch_repo_data(sub_batch, session, tokens, token_index)
                for repo_name, owner in sub_batch:
                    alias = sanitize_for_alias(repo_name, owner)
                    repo_data = batch_result.get(alias, {})
                    if repo_data and 'primaryLanguage' in repo_data and
repo_data['primaryLanguage']:
                        language = repo_data['primaryLanguage']['name']
                    else:
                        language = None
                    language_list.append((owner, repo_name, language))

            token_index = (token_index + 1) % len(tokens)  # Rotate the token index
            test = spark.createDataFrame(language_list, schema = ['owner', 'repo', 'language'])
            test.write.csv(f"data/Nik_Data_lake/pre_batch_{i + 1}.csv", mode="overwrite")
```

```
            print(f"Saved Batch {i + 1}")

            # Add DataFrame to pre_repo_languages_all dictionary
            pre_repo_languages_all[f"batch_{i + 1}"] = test
            print(f"Batch Nr. {i} took ", time.time() - start_time)

    return pre_repo_languages_all

# Run the event loop and fetch data
tokens = [github_token3, github_token2, github_token]#, github_token3, github_token4,
github_token]
batch_size = 400
df_repos = asyncio.run(main(pre_parts, n, batch_size, tokens))
```

Appendix 5: GraphQL and async repo language request

```
nest_asyncio.apply()

def sanitize_for_alias(username):
    return 'a' + ''.join(ch if ch.isalnum() else '_' for ch in username)

def construct_query(logins):
    query_parts = [
        f'''
        {sanitize_for_alias(login)}: user(login: "{login}") {{
            location
        }}
        ''' for login in logins
    ]
    return '{' + ''.join(query_parts) + '}'

async def fetch_data(query, tokens, session, token_index, max_retries=3):
    headers = {
        'Authorization': f'bearer {tokens[token_index]}',
        'Content-Type': 'application/json'
    }
    backoff_time = 1  # Initial backoff time in seconds

    for attempt in range(max_retries):
        try:
            async with session.post('https://api.github.com/graphql', json={'query': query},
headers=headers) as response:
                if response.status == 200:
                    response_json = await response.json()
                    if 'data' in response_json:
                        return response_json['data']
                    else:
                        raise Exception(f"Data key missing from response: {response_json}")
                else:
                    response_text = await response.text()
                    print(f"Attempt {attempt + 1}: Request failed with status code
{response.status}: {response_text}")
        except Exception as e:
            print(f"Attempt {attempt + 1}: Request failed with error: {e}")

        await asyncio.sleep(backoff_time)  # Wait before the next retry
        backoff_time *= 2  # Increase backoff time exponentially

    raise Exception(f"All {max_retries} retries failed")


async def batched_fetch_data(logins, batch_size, tokens):
    all_data = {}

    async with aiohttp.ClientSession() as session:
        token_index = 0  # Initialize token index for token rotation
        total_batches = len(logins) // batch_size + (1 if len(logins) % batch_size != 0 else 0)

        for i in range(0, len(logins), batch_size):
```

```
            batch = logins[i:i + batch_size]
            query = construct_query(batch)

            try:
                data = await fetch_data(query, tokens, session, token_index)

                for login in batch:
                    alias = sanitize_for_alias(login)
                    user_data = data.get(alias, None)
                    if user_data:
                        all_data[login] = user_data
            except Exception as e:
                print(f"Failed to process batch {i // batch_size + 1} of {total_batches}: {e}")

            token_index = (token_index + 1) % len(tokens)  # Rotate the token index

            # Print update every 25 batches
            current_batch = i // batch_size + 1
            if current_batch % 25 == 0:
                print(f"Processed {current_batch} of {total_batches} batches")

    return all_data
```

Appendix 6: GraphQL Request for User Location

```
async def fetch_data(query, tokens, session, token_index, max_retries=3):
    headers = {
        'Authorization': f'bearer {tokens[token_index]}',
        'Content-Type': 'application/json'
    }
    backoff_time = 1  # Initial backoff time in seconds

    for attempt in range(max_retries):
        try:
            async with session.post('https://api.github.com/graphql', json={'query': query},
            headers=headers) as response:
                if response.status == 200:
                    response_json = await response.json()
                    if 'data' in response_json:
                        return response_json['data']
                    else:
                        raise Exception(f"Data key missing from response: {response_json}")
                else:
                    response_text = await response.text()
                    print(f"Attempt {attempt + 1}: Request failed with status code
                                            {response.status}: {response_text}")
        except Exception as e:
            print(f"Attempt {attempt + 1}: Request failed with error: {e}")

        await asyncio.sleep(backoff_time)  # Wait before the next retry
        backoff_time *= 2  # Increase backoff time exponentially

    raise Exception(f"All {max_retries} retries failed")

# Other code in between ….

            try:
                data = await fetch_data(query, tokens, session, token_index)

                for database_id in batch:
                    node_key = f"node_{database_id}"
                    user_data = data.get(node_key, None)
                    if user_data:
                        all_data[database_id] = user_data
            except Exception as e:
                print(f"Failed to process batch {i // batch_size + 1} of {total_batches}: {e}")
```

Appendix 7: Error handling GraphQL Request

```
from fuzzywuzzy import process

# Function to sanitize the location field
def sanitize_location(location):
    if pd.isna(location):
        return ""
    # Keep only alphanumeric, space, and underscore
    sanitized_location = re.sub(r'[^a-zA-Z0-9 _]', '', location)
    return sanitized_location.strip().lower()  # Convert to lowercase and remove leading/trailing
spaces

# Apply the sanitization function to the location column
all_users_df['location'] = all_users_df['location'].apply(sanitize_location)

italian_keywords = [
    "rome", "roma", "milan", "milano", "naples", "napoli", "turin", "torino", "palermo",
    "genoa", "genova", "bologna", "florence", "firenze", "venice", "venezia", "verona",
    "cagliari", "modena", "parma", "perugia", "rimini", "trieste", "trento", "udine",
    "la spezia", "salerno", "catania", "bari", "taranto", "l'aquila", "aosta", "trentino",
    "bolzano", "bozen", "pescara", "foggia", "reggio calabria", "cosenza", "potenza",
    "catanzaro", "siracusa", "ragusa",  "trapani", "sassari", "olbia", "oristano", "nuoro",
    "matera", "campobasso", "frosinone", "latina", "rieti", "viterbo", "ancona",
    "ascoli piceno", "fermo", "macerata", "pesaro", "urbino", "arezzo", "grosseto",
    "livorno", "lucca", "massa", "carrara", "pisa", "pistoia", "prato", "siena", "italia",
    "trentino-alto adige", "valle d'aosta", "aosta valley", "veneto", "apulia", "puglia",
    "calabria", "campania", "emilia-romagna", "friuli-venezia giulia", "lazio", "liguria",
    "lombardy", "lombardia", "marche", "molise", "abruzzo", "basilicata", "sicily", "sicilia",
    "sardinia", "sardegna", "piedmont", "piemonte", "tuscany", "toscana", "umbria", "italy"
]

def is_italian_location(location):
    if not location:
        return False
    if any(keyword in location for keyword in italian_keywords):
        return True
    # Using fuzzy matching to account for typos
    closest_match, score = process.extractOne(location, italian_keywords)
    return score > 95

# Filter in rows with Italian locations using the pandas DataFrame method .loc
italian_df = all_users_df.loc[all_users_df['location'].apply(is_italian_location)]
```

Appendix 8 : FuzzyWuzzy identify German and Italian Locations

```
# Assuming 'email_list' contains a list of emails to look up
post_IT_data = []
# Loop through each email and get the GitHub username
for email in post_IT_email_list:
    username = get_github_username_by_email(email)
    post_IT_data.append({'email': email, 'username': username})
    time.sleep(4)
```

Appendix 9: Getting Username from E-Mail

```
Error for andrea.dallefratte@elmec.it: 403 Client Error: Forbidden for url:
https://api.github.com/search/users?q=andrea.dallefratte@elmec.it%20in:email

Error for luca.cominoli@elmec.it: 403 Client Error: Forbidden for url:
https://api.github.com/search/users?q=luca.cominoli@elmec.it%20in:email2 Problems
```

Appendix 10: Error Messages

	owner	name	description	langdetect_result	fasttext_result	lingua_result
0	siddheshmhatre	Big-Interleaved-Dataset	biginterleaveddataset	None	Uncertain	True
1	e-caliano	Labyrinth2023	progetto del corso di programmazione del corso...	it	it	True
2	italia	writing-toolkit	guida al linguaggio della pubblica amministraz...	it	it	True
3	Mattezucca1	Auto-writer	un auto writer per tutte le applicazioni	it	it	True
4	xingfuggz	baykeShop	baykeshop拜客商城系统是一款全开源python栈商城系统后端依托django强大的框...	None	zh	True
5	hishixuan	fluent_python	haihaihai	None	Uncertain	True
6	lordtitanium87	python-docs-flask-minimal	a simple flask app to accompany python dev cen...	None	Uncertain	True
7	CScorza	Generatore-di-Password	generatore di password	it	it	True
8	Otich	E5	e	None	it	False
9	Asadek14	C41	génie logiciel	None	fr	True
10	ivanosaccheo	my_functions	contiene la versione pù aggiornata del file li...	it	it	True
11	imillagrimemc	e	e	None	it	False
12	ELtechtalks	esempio-di-calcolo-condice-fiscale	ecco un esempio di codice python completo che ...	it	it	True
13	hobuinc	doppkit	doppkit	None	it	True
14	HelKim	cet_check_grade	cet英语四六级穷搜查分忘了准考证的同学可以试试	None	Uncertain	True
15	AntonioBerna	gauss-distribution	software di simulazione distribuzione normale ...	it	it	True
16	alemiaschi	Linguistica_Computazionale_22-23	repository per il laboratorio python del corso...	it	it	True
17	drego85	RSSFeedGenerator	feed rss di alcune fonti che non offrono diret...	it	it	True
18	XdRonny	Vcpersonal	vcraid	None	it	False
19	All-In-One-Discord-Program-Tools	All-In-One	aio dpt all in one discord program tools tok...	None	Uncertain	True

Appendix 11: Language Analysis of repo description

```
german_keywords = [
   "berlin", "hamburg", "munich", "muenchen", "cologne", "koeln", "frankfurt", "stuttgart",
    "dusseldorf", "dortmund", "essen", "leipzig", "bremen", "dresden", "hanover", "nuremberg",
    "nuernberg", "duisburg", "bochum", "wuppertal", "bielefeld", "bonn", "muenster", "karlsruhe",
    "mannheim", "augsburg", "wiesbaden", "gelsenkirchen", "mönchengladbach", "braunschweig",
    "chemnitz", "kiel", "aachen", "halle", "magdeburg", "freiburg", "krefeld", "lübeck",
    "oberhausen", "erfurt", "mainz", "rostock", "kassel", "hagen", "saarbrücken", "hamm",
    "mülheim", "potsdam", "ludwigshafen", "oldenburg", "leverkusen", "osnabrück",
    "baden-württemberg", "bavaria", "bayern", "berlin", "brandenburg", "bremen", "hamburg",
    "hesse", "hessen", "lower saxony", "niedersachsen", "mecklenburg-western pomerania",
    "mecklenburg-vorpommern", "north rhine-westphalia", "nordrhein-westfalen",
    "rhineland palatinate", "rheinland-pfalz", "saarland", "saxony", "sachsen", "saxony-anhalt",
    "sachsen-anhalt",   "schleswig-holstein", "thuringia", "thüringen", "germany", "deutschland"
]
```

Appendix 12: German Keywords

```
def get_github_username_by_email(email):
    # Define the search URL
    search_url = f'https://api.github.com/search/users?q={email} in:email'
    try:
        # Make the GET request to the GitHub API
        response = requests.get(search_url, headers=headers)
        response.raise_for_status()  # Raises an exception for HTTP errors
        # Parse the response JSON and extract the username
        search_results = response.json()
        items = search_results.get('items', [])
        if items:
            username = items[0].get('login')
            print(username)
            return username  # Return the username
    except requests.exceptions.RequestException as err:
        print(f"Error for {email}: {err}")
        time.sleep(5)
    except Exception as e:
        print(f"An unexpected error occurred for {email}: {e}")
```

```
        time.sleep(5)

    # Return None if the request fails or no items are found
    return None
```

Appendix 13: def get_github_username_by_email