PREDICTING SOFTWARE SIZE FROM REQUIREMENTS WRITTEN IN NATURAL LANGUAGE: A GENERATIVE AI APPROACH

A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements of the Degree of

MASTER OF SCIENCE

In Computer Engineering

by
Dhia Eddine KENNOUCHE

July 2024
İZMİR

We approve the thesis of **Dhia Eddine KENNOUCHE**

Examining Committee Members:

Prof. Dr. Onur DEMİRÖRS
Department of Computer Engineering, İzmir Institute of Technology

Prof. Dr. Oğuz DİKENELLİ
Department of Computer Engineering, Ege University

Asst. Prof. Dr. Emrah İNAN
Department of Computer Engineering, İzmir Institute of Technology

1 July 2024

Prof. Dr. Onur DEMİRÖRS
Supervisor Department of Computer
Engineering, İzmir Institute of Technology

Prof. Dr. Onur DEMİRÖRS
Head of Department of Computer
Engineering, İzmir Institute of Technology

Prof. Dr. Mehtap EANES
Dean of the Graduate School

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor, Prof. Dr. Onur Demirörs, for his insightful guidance, and invaluable expertise throughout my research journey. His mentorship has been instrumental in shaping my academic development.

I am also deeply thankful to Hüseyin and Samet, whose assistance and advice have been a constant source of encouragement. Their contributions have been invaluable to the completion of this thesis.

To my wife, Aysel, your love, patience, and understanding have been my pillar of strength. Your unwavering support has made this journey possible, and for that, I am eternally grateful.

I extend my heartfelt thanks to my family members: my mother, my father, Takai, Chihab, Somia, and Esma. Your constant encouragement, belief in me, and emotional support have been vital to my success. Each of you has played a unique and significant role in my life, and I am deeply appreciative of your love and care.

Finally, I want to thank all the friends and colleagues I have met throughout my career. Your support, trust, and companionship have been a source of motivation and strength. Each of you has contributed to my growth in ways that words cannot fully capture.

Thank you all for being part of this journey with me.

This thesis is supported by The Scientific and Technological Research Council of Turkey (TUBITAK) ARDEB 1001 [Project number: 121E389] program.

ABSTRACT

PREDICTING SOFTWARE SIZE FROM REQUIREMENTS WRITTEN IN NATURAL LANGUAGE: A GENERATIVE AI APPROACH

In project management, software size measurement represents a critical process aimed at visualizing a project. This quantification is pursued independently of the specific technologies or technical decisions adopted during the project's development phase. Among the various methodologies employed for this purpose, the COSMIC Functional Size Measurement (FSM) and Event Points are used to facilitate such assessments. These methodologies are instrumental in offering a standardized approach for measuring software size, yet they inherently demand a considerable amount of manual effort. Furthermore, these methods require the manual extraction of Objects of Interest and Event Names, adding to the labor-intensive nature of the process.

In response to these challenges, this thesis implements a suite of Artificial Intelligence (AI)-based methods that have dramatically transformed the measurement process. These innovative approaches encompass the creation of a Regression Model that predicts software sizes with remarkable accuracy, a Summarization Model that automates the extraction of Event Names, and a finely tuned Large Language Model (LLM) that generates Objects of Interest with a significant precision. The adoption of these AI-driven techniques has proven to be highly successful, substantially minimizing the manual effort traditionally required in software size measurement and thereby greatly enhancing both efficiency and reliability of estimation practices.

Together, these AI-based methodologies represent a significant advancement in software size measurements, offering a more streamlined and efficient approach. By reducing the reliance on manual processes, these methods not only enhance the accuracy and reliability of measurements but also contribute to a more agile project management environment.

ÖZET

DOĞAL DİLDE YAZILMIŞ GEREKSİNİMLERDEN YAZILIM BOYUTUNU TAHMİN ETME: ÜRETKEN YAPAY ZEKÂ TABANLI BİR YAKLAŞIM

Proje yönetiminde, yazılım boyutunun ölçülmesi, bir projenin çeşitli yönlerini görselleştirmeyi amaçlayan kritik bir süreci temsil eder. Bu nicelendirme, projenin geliştirme aşamasında benimsenebilecek spesifik teknolojilerden veya teknik kararlardan bağımsız olarak gerçekleştirilir. Bu amaçla kullanılan çeşitli metodolojiler arasında, COSMIC Fonksiyonel Boyut Ölçümü (FSM) yöntemi ve Olay Noktaları, bu tür değerlendirmeleri kolaylaştırmak için kullanılır. Bu metodolojiler, yazılım boyutunu ölçmek için standart bir yaklaşım sunmakla birlikte, önemli miktarda manuel çaba gerektirir. Özellikle, her bir kullanım senaryosunun bireysel özelliklerine bağlı olarak detaylı hesaplamalar yapılmasını gerektirirler. Ayrıca, bu yöntemler, manuel olarak İlgi Nesneleri ve Olay İsimlerinin çıkarılmasını gerektirir, bu da sürecin emek yoğun doğasını artırır.

Bu zorluklara yanıt olarak, bu tez, ölçüm sürecini dramatik bir şekilde dönüştüren bir dizi Yapay Zekâ (AI) tabanlı metodolojiyi uygulamaktadır. Bu yenilikçi yaklaşımlar, yazılım boyutlarını remarkable doğrulukla tahmin eden bir Dizi Regresyon Modeli, Olay İsimlerinin çıkarılmasını otomatikleştiren bir Özetleme Modeli ve İlgi Nesnelerini büyük bir doğrulukla üreten ince ayarlı bir Büyük Dil Modeli (LLM) yaratılmasını kapsar. Bu AI odaklı tekniklerin benimsenmesi, geleneksel olarak yazılım boyutunu ölçmede gerekli olan manuel çabayı önemli ölçüde azaltmış ve böylece tahmin uygulamalarının hem verimliliğini hem de güvenilirliğini büyük ölçüde artırmıştır.

Bu AI tabanlı metodolojiler, proje yönetiminde önemli bir ilerlemeyi temsil eder, yazılım boyutunu ölçmek için daha düzenli ve verimli bir yaklaşım sunar. Manuel süreçlere olan bağımlılığı azaltarak, bu yöntemler ölçümlerin doğruluğunu ve güvenilirliğini artırmakla kalmaz, aynı zamanda daha çevik ve duyarlı bir proje yönetim ortamına da katkıda bulunur.

To Aysel, and Maya.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

CHAPTER 1

INTRODUCTION

In the domain of software development, achieving the dual objectives of delivering a functional product within specified timeframes and budgetary allocations is paramount.[1] Central to realizing these objectives is the precise measurement of software size at the outset and during the progress of a project.[2]

Software size measurement is a crucial step in the planning phase of software development, helping teams to forecast the effort, resources, and time required for a project.[3] Size measurement methods fall into two broad categories: formal and expert judgments, each with its own approach and set of techniques.[4]

Formal software size measurement methods are grounded in quantifiable data and structured techniques. These methods aim to provide a repeatable and unbiased measure by relying on measurable attributes of software development. For instance, Lines of Code (LOC) measures the number of lines of code, while Function Points (FP) measure the functionality delivered to the user, factoring in the complexity of various inputs, outputs, and interactions.[5] Use Case Points (UCP) consider the use cases of a project, assessing the complexity and the technical and environmental elements involved. Event Points focuses on event-driven architectures, and the different types of events needed for building such a system[6], these events can be split into three categories: Communication, Processing, and Interaction. These objective methods strive for accuracy and precision, utilizing historical data, mathematical models, or predefined metrics to measure software size.

On the other hand, subjective software size estimation methods depend on the judgment, experience, and intuition of individuals or groups.[7] Unlike objective methods that focus on measurable aspects, subjective techniques are based on perceptions and expectations. Expert judgment is one such method, where one or more experts use their knowledge of similar projects to estimate the size of the current project. The Delphi Technique involves a panel of experts who, through structured communication and multiple rounds of questionnaires, refine their estimates to converge on an accurate

prediction.[8] Planning Poker, often used in Agile development, is a consensus-based technique that employs a gamified approach to estimating effort or relative size, aiming to avoid bias by having team members make estimates anonymously.[9]

The key difference between these approaches lies in their basis of measurement—objective methods are data-driven and repeatable, less influenced by individual biases, while subjective methods are flexible, relying on personal insights but susceptible to variability based on the estimators' experience and biases. Objective methods require historical data or a well-defined model, which might not be available for novel projects, whereas subjective methods can be applied even with limited information, capturing the nuances of complex or innovative projects through expert insight.[4]

In this dissertation, the focus will be placed upon two distinct objective methodologies: COSMIC Functional Size Measurement (COSMIC FSM) and Event Points. The COSMIC FSM methodology enjoys international recognition and is employed by a diverse array of organizations and projects globally for a multitude of purposes, including but not limited to cost measurement, project planning, and productivity analysis.[10] It represents one of several available functional size measurement methods, alongside others such as the International Function Point Users Group (IFPUG) Function Point Analysis (FPA) and the Mk II FPA. Each method is governed by its own specific set of rules and applications, yet collectively, they share the common objective of providing a standardized approach to measuring the functional size of software.[11]

On the other hand, Event Points emerges as a novel FSM methodology specifically designed to address the unique requirements of event driven architectures.[6] This method seeks to offer more precise measurements by focusing on the events that transpire within such projects. Through the lens of Event Points, the aim is to capture and quantify the dynamic interactions, communication, processing, and decision functionalities inherent in event driven project, thereby facilitating more accurate and reflective software measurement and analysis.[12]

A principal issue associated with both COSMIC Functional Size Measurement (COSMIC FSM) and Event Points methodologies is their reliance on manual measurement processes.[13] Such processes are heavily dependent on the expertise and discernment of the estimator, thereby introducing a considerable level of variability and the potential for inaccuracies.[14] This variability and susceptibility to error have contributed to a diminished interest in the adoption of these methods, leading

organizations to prefer Subjective Software Size Measurement methods.[15] The labor-intensive characteristic of manual measurements highlights the imperative for the development and implementation of more objective and automated strategies for software size estimation.

Despite various efforts to automate software size measurement we will discuss in detail in the following chapter, a notable research gap persists: the application of Machine Learning (ML) models for the automation of COSMIC FSM process has yet to be explored comprehensively, and Event Points measurement has not been explored yet at all. Moreover, the automation of critical tasks, including the identification of Objects of Interest and the generation of Event Names, remains largely unaddressed. These tasks, crucial for the accurate and efficient measurement of software size, continue to be performed subjectively and manually.

This thesis endeavors to address these shortcomings by investigating the potential of ML models to automate the measurement of Event Points and COSMIC FSM, facilitate the identification and naming of Objects of Interest, and generation of Event names in software size estimation. By proposing a shift from subjective, manual estimation methods to an objective, ML-driven approach, this research aims to significantly improve the accuracy, efficiency, and reliability of software size estimation. Such advancements are expected to improve resource allocation and project planning, thereby contributing to the successful execution of software development projects.

Employing a Sequence Regression model to predict COSMIC Functional Size Measurement (FSM) utilizing our proprietary dataset, we attained an average accuracy level denoted as 94%. A similar Sequence Regression model applied to Event Points yielded an average accuracy of 92%. To aggregate predicted COSMIC points, the summation approach resulted in an outcome of 83%, whereas the application of the same Sequence Regression model facilitated an achievement of 88%. A parallel procedure was conducted for Event Points, where the summation yielded 92, and the Sequence Regression model application resulted in 81%.

For extracting Event Names, a Summarization Model was utilized, and the outcomes were evaluated using Rouge Metrics, yielding scores of Rouge1: 73%, Rouge2: 49%, and Rougel: 70%, respectively. In the task of extracting the Object of Interest, the mistral-7b model was employed, achieving an exact match accuracy of 79%.

This dissertation is organized into several chapters, each designed to explore different facets of "A Method for Predicting Software Size from Requirements" in depth. Following this introduction, the structure of the dissertation is as follows:

Chapter 2: Literature Review - This chapter provides a comprehensive review of the existing literature relevant to AI usage in SSM. It critically examines previous studies, and methodologies that have shaped the current understanding of the subject. The aim is to identify gaps in the existing knowledge base and to position the current research within the broader academic discourse.

Chapter 3: Methodology - In this chapter, the research methodology employed in this study is detailed. It outlines data collection methods, and analytical techniques used to address the research questions. The rationale behind the choice of methodology and its alignment with the research objectives are also discussed, ensuring transparency and reproducibility of the research process.

Chapter 4: Results and discussion - This chapter synthesizes the findings of this thesis, discussing the different metrics used in the different models and details the result of each model.

Chapter 5: Conclusion and Recommendations - The concluding chapter summarizes the key findings of the research, reflecting on the research objectives and questions. It also outlines the limitations of the study and provides recommendations for future research in the area.

CHAPTER 2

LITERATURE REVIEW

Chapter 2 is about diving into the research that's been done on SSM, COSMIC FSM, and usage of AI within SSM context. We'll start by looking at the basics of software measurement and why COSMIC FSM is important. Then, we'll talk about Event Points, another important method of software measurement. After that, we'll explore how summarization models and Language Models (LM) work and why they matter in software measurement. Finally, we'll see how Artificial Intelligence (AI) is being used in software measurement and COSMIC, looking at different studies and trends. This chapter lays the groundwork for understanding the rest of the research in this thesis.

2.1. Background on Software Measurement, COSMIC FSM

In the nascent stages of software size assessment, the primary metric utilized was Lines of Code (LOC), serving as a barometer for productivity and quality within the software development milieu. Nonetheless, the efficacy of LOC encountered notable impediments due to its reliance on specific programming languages and its incapacity to comprehensively encapsulate the intricacies of software functionalities. Consequently, the emergence of Function Points as an alternative metric ensued, aimed at quantifying the business functionality intrinsic to software delivery. While Function Points marked a progression by mitigating certain limitations of the LOC metric, they encountered challenges of their own, including subjective interpretation and a lack of automated mechanisms. Simultaneously, explorations into alternative size measurement methodologies, such as Object-Oriented (OO) metrics and Use Case Points, were pursued but faced hurdles in achieving widespread acceptance within the software development community.[16]

The introduction of Agile methodologies ushered in novel metrics for gauging software size, notably Story Points, which accentuate the business value engendered

through iterative development cycles. This paradigm shift reflects the ongoing endeavor within the software realm to pinpoint the most efficacious means of measuring software size. This endeavor is informed by multiple factors, including the aspiration to augment productivity, ensure quality, and optimize business value. Presently, there is a discernible inclination towards embracing service-oriented architectures, promising refined practices for sizing software.[16] This inclination underscores the dynamic nature of software measurement practices and the perpetual endeavor to reconcile the imperatives of technological advancement with the exigencies of precise and meaningful measurement.

Functional size measurement (FSM) entails the quantification of the functionality furnished by a software system through the evaluation of its cohesive sequences of execution. Alan Albrecht pioneered this concept in 1979 through the methodology of function point analysis (FPA), galvanizing the development and evolution of diverse functional size metrics and methodologies over subsequent years.[17]

Functional Size Measurement methods are delineated into two distinct generations, each embodying a disparate approach and underlying philosophy towards quantifying software functionality.[18]

The first generation of FSM methods is characterized by its empirical underpinning, predominantly drawing from observed effort or tangible software components. This category encompasses methodologies such as IFPUG Function Point Analysis (FPA)[19], and Nesma FPA.[20] An inherent limitation of these first-generation methods lies in their dependence on subjective assessment criteria rather than a universally accepted measurement unit, employing a rule-based approach to evaluate segments of the Functional User Requirements.[18] While this approach assigns measurement units based on specific criteria, it lacks a standardized unit of measurement, potentially engendering variability, and inconsistency in results.

Conversely, the second generation of FSM methods emerged in response to the deficiencies of their precursors, with a concerted effort to ground the measurement process in fundamental software engineering principles. These principle-based methods strive for a well-defined measurement unit, departing from the subjective assessment rules of the first generation. The focus shifts towards identifying instances of a predefined measurement unit, aspiring towards a more objective and scientifically rigorous framework for measuring software size. This transition towards second-generation FSM

methods signifies a notable advancement in the field, underscoring a collective shift towards more rigorous, principle-based approaches in assessing software functionality.[18]

This distinction between the two generations of FSM methods elucidates the evolution and refinement in methodologies aimed at measuring software size, spotlighting an industry-wide inclination towards adopting more principled and scientifically grounded approaches.

Numerous fundamental concepts underlie the COSMIC methodology[18], particularly within the context of Agile software development projects. These concepts play a crucial role in refining the accuracy and dependability of planning and measurement processes within such projects. The following delineates these key concepts[21]:

Functional Size Measurement: At the heart of COSMIC's application lies its functionally sizing of software systems. This holds relevance at the micro-level within Agile projects, where it is employed to gauge elements such as User Stories. This approach facilitates a detailed and precise assessment of software functionality, crucial for Agile methodologies that prioritize flexibility and incremental development.

COSMIC Function Point (CFP): The methodology utilizes COSMIC Function Points as a metric for measuring the size of software functions. Serving as a tangible quantifier, this metric aids in the measurement and planning phases of Agile software projects. By furnishing a standardized measure of software functionality, CFPs enable more accurate forecasting and resource allocation.

Parametric Approach: COSMIC advocates for a parametric approach to project planning and measurement, supplementing subjective insights derived from developer experience with a structured model. Drawing inspiration from Boehm's Constructive Cost Model (COCOMO), this approach suggests that incorporating such models can significantly enhance the accuracy of project measurements in Agile environments.

Documentation Quality: The methodology underscores the significance of documentation quality in determining the functional size of each User Story. High-quality, comprehensive, and precise documentation is pivotal as it directly influences the efficacy of Agile planning and measurement processes. Emphasizing documentation quality ensures that all stakeholders possess a clear and shared understanding of project requirements and scope.

Continuous Refinement of Estimates: COSMIC encourages the ongoing refinement of project estimates, leveraging a Project Historical Database that encompasses qualitative and quantitative data. This iterative measurement approach permits developers to adjust forecasts based on empirical data and accumulated project experience, thereby enhancing the reliability and accuracy of project timelines and resource requirements.

In summary, the integration of the COSMIC methodology into Agile software development projects aims to enhance the effectiveness of planning, measurement, and decision-making processes. By emphasizing documentation quality and the strategic use of historical data alongside a parametric approach, COSMIC offers a comprehensive framework for addressing the unique challenges of Agile project environments.

The progression and refinement of the COSMIC method for functional size measurement can be comprehensively understood through its evolving principles and features. Initially, COSMIC focused on quantifying software systems' functional size by identifying and measuring Base Functional Components (BFCs) outlined within functional user requirements.[21]

Central to COSMIC's methodology were several key principles, including process initiation through inputs, execution of data processing tasks, and delineation of data groups as subsets of Objects of Interest (OOI). A significant innovation introduced by COSMIC was the concept of data movements—categorized into Entry, Exit, Read, and Write types—aimed at quantifying software functionality in a nuanced manner.[11] Additionally, COSMIC implemented a layering strategy to systematically partition software into distinct layers, facilitating consistent functional processes across the software architecture.

Over time, COSMIC has undergone methodological enhancements to effectively measure functional size, adapt to diverse software architectures, and address challenges posed by emerging technology platforms. By adhering to foundational principles while embracing adaptability, COSMIC has evolved into a comprehensive and robust methodology for estimating and measuring functional size in software systems.

In addition to its primary functions, the COSMIC FSM method serves a critical role in benchmarking activities, allowing organizations to compare project unit costs against analogous endeavors within the industry, facilitating standardization and internal benchmarking. Its versatility in measuring functional size at any software life cycle stage

empowers stakeholders to support various decision-making processes, including project measurement, budgeting, iteration planning, project re-measurement, and process improvement monitoring, significantly enhancing software development and project management practices.[22]

In essence, the COSMIC Functional Size Measurement method not only provides a robust mechanism for accurately estimating and managing software projects but also enhances organizational capabilities in benchmarking and strategic decision-making, reinforcing its status as a critical component of contemporary software project management and development strategies.

2.2. Background on Event Points

Event-driven architecture (EDA) represents a design paradigm wherein diverse software components and services engage in communication through the exchange of events. Within an event-driven architecture, events serve as triggers and controllers for the flow of processes and interactions across various segments of a system. This methodology facilitates decoupled and asynchronous communication among components, thereby affording greater flexibility and scalability in the design and implementation of applications and systems.[23]

The Event-Driven Architecture (EDA) was introduced as a method to facilitate the transmission of events between decoupled software components and services, aimed at achieving several objectives.[23] These include complementing Service-Oriented Architecture (SOA) by introducing capabilities for long-running asynchronous processes, enabling asynchronous events transmission distinct from the request/response exchanges typical of SOA. Moreover, EDA fosters decoupled interactions by allowing event publishers to operate without awareness of event subscribers, thereby establishing a decoupled interaction model. It also facilitates many-to-many communications through publish/subscribe messaging, enabling one event to influence multiple subscribers and thereby facilitating flexible communication patterns. Furthermore, EDA enables event-based triggers, where the recipient determines the flow of control based on the posted event, allowing for dynamic activation of components. Finally, EDA supports

asynchronous operations through event messaging, thereby offering flexibility and scalability in managing events and processing tasks asynchronously.

Understanding the usability of Event-Driven Architecture (EDA) necessitates a consideration of its components and implementation aspects. EDA implementation components encompass various elements such as Event Metadata, Event Processing Engine, Event Transport, Enterprise Information Caching, Service Invocation, Enterprise Integration Backbone, Event Data, Event Development Tools, Event Management Tools, Enterprise Applications, Management Portals, and Integrated Resources.[24] The ease of use of EDA is contingent upon factors including the organization's existing infrastructure, expertise, and the availability of tools and components to support event-driven processes. Organizations equipped with well-established event metadata architecture, clear event specifications, and efficient event processing rules may find it more straightforward to implement EDA. Additionally, the availability of adequate event development and management tools can streamline the adoption and utilization of EDA within an organization, with usability varying based on readiness, resources, and commitment to aligning architecture with event-driven principles.[24]

The historical origins of event points can be traced to advancements in software engineering and size measurement methodologies.[12] Studies on business process modeling, state machines, and Event-Driven Architecture have been conducted since the 1970s, underscoring the longstanding interest in event-driven software models. Furthermore, the integration of events into functional size measurement methods, such as COSMIC FSM, highlights the incorporation of events for decomposing functional processes. This historical context lays the groundwork for the exploration and emergence of event points as a concept for software size measurement.[24]

The purpose of Event Points is to furnish a method for software size measurement focusing on events rather than data.[6] It facilitates the measurement of software size based on events occurring within the system, offering an alternative approach to size measurement across various software architectures, particularly in agile environments. Event Points can be deployed for size measurement at different stages of the software development lifecycle and prove particularly valuable in scenarios where traditional size measurements based on lines of code (LOC) are impractical, such as during early measurement phases.

2.3. Summarization Model, and LLM

Large Language Models (LLMs) have been a very hot subject in the last years. LLMs are described as large pre-trained AI systems that showcase expressive and interactive capabilities, allowing them to be repurposed across different domains and tasks with minimal effort.[25]

Large language models (LLMs) are built using various pretraining methods such as Left-to-Right Language Models[26], Masked Language Models[27], and Encoder-Decoder Models.[28,29]

Language models are built using different methodologies and architectures, with advancements in the field continually shaping their design. Several key aspects of language model development are discussed by Tianyu W et al.[30]

1. Pre-Trained Language Models (PLM): These models utilize self-supervised learning over large-scale texts and have gained significant attention since 2018. The PLM learns general language models based on self-learning tasks such as masked word prediction, sequence recognition of sentences, text filling in the blank, and text generation. Notable PLMs include ELMo, BERT, and GPT-3, which have each made significant contributions to natural language processing.
2. Model Compression: The size of modern large language models poses practical challenges due to the vast number of parameters they contain. To address this issue, researchers have focused on model compression and optimization techniques to reduce the size of language models while maintaining their performance on various tasks. Approaches include distillation-based methods, pruning-based techniques, and quantization-based methods.
3. In-Context Learning (ICL): ICL is a meta-learning method introduced by models like GPT-3, enabling the model to learn and complete tasks by imitation based on the context provided. This method allows the model to model more context information to solve specific tasks effectively, improving the performance of various tasks and enhancing zero-shot and few-shot learning capabilities.

4. Instruction Fine-Tuning (IFT): IFT involves describing all NLP tasks using natural language instructions and fine-tuning large language models accordingly. This approach enables models like FLAN to achieve better performance on specific NLP tasks and improve the generalization ability for new tasks by adjusting model parameters based on natural language instructions.
5. Reinforcement Learning From Human Feedback: Reinforcement learning focuses on learning the optimal policy to maximize rewards or reach specific targets through interactions between the agent and the environment. This approach has shown strong capabilities in various fields with large action spaces, such as gaming, robotics control, and molecular optimization.

These aspects provide a glimpse into the diverse methodologies and techniques used to build and enhance large language models like ChatGPT, showcasing the evolution and complexity of modern language model development.

In the other hand, summarization is a crucial natural language processing task that involves condensing large volumes of information from a source document into a shorter and concise form while preserving the key information and meaning.[31] It helps in extracting the most important points, ideas, and concepts from the original content, allowing for a quick understanding of the main content without the need to read the entire document. There are two primary types of summarization algorithms: extractive and abstractive summarization.[31] Extractive summarization involves selecting and combining existing phrases and sentences from the source document, while abstractive summarization involves generating new phrases and sentences that may not exist in the original text but represent the same meaning. Summarization is widely used in various applications such as news summarization, search engines, research paper abstracts, and more, to provide users with concise and informative content summaries.

Summarization involves condensing a longer piece of text into a shorter version while retaining the essential information and main points. The process typically includes the following steps[32] :

- Understanding the Text: The summarization system comprehends the input text to identify important concepts and themes.
- Identifying Key Information: It selects significant sentences, phrases, or paragraphs that encapsulate the main ideas.

- Paraphrasing and Condensing: The system rephrases the selected content and condenses it to form a concise summary.
- Ensuring Coherence: The summarization tool ensures that the summary flows logically and retains the coherence of the original text.
- Evaluation: The quality of the summary is evaluated based on criteria such as factual accuracy, relevance, and fluency.

2.4. Application of AI in SSM and COSMIC

In this section we will be exploring the different research work that exists related to using AI tools for SSM and COSMIC Points. Each paragraph will have an overview about a research paper, and at the end you can see FIGURE that summarizes all papers, used methods, and their outcome.

This research paper titled "Exploring Machine Learning Techniques for Software Size Estimation"[33] explores the application of machine learning techniques, specifically Genetic Programming (GP) and Neural Networks (NN), for estimating software size metrics such as lines of code (LOC) in the early stages of software development. The study compares the effectiveness of these techniques in estimating LOC using two different data sets based on function points and number of components. Experimental results and analysis regarding the accuracy and comparison of the machine learning techniques are presented. The paper discusses the principles of Genetic Programming and Neural Networks, describes the experiment setup, data sets used, evaluation measurements, and the configuration of the GP tool and neural network tool. Furthermore, it analyzes the obtained results, accuracy evaluations, and provides conclusions on the potential of GP and NN for software size estimation. The outcome of the paper showcased positive and negative aspects of applying Genetic Programming (GP) and Neural Networks (NN) for software size estimation. The models obtained using GP and NN showed varying levels of accuracy based on different metrics. The models using the function points (FP) metric had better accuracy using GP compared to NN, but neither model was deemed acceptable according to the set criteria. On the other hand, the models using the number of components (NOC) metric were acceptable for both techniques and presented similar accuracy.

In the paper titled “Enhancing Software Effort Estimation with Ant Colony Optimization Algorithm and Fuzzy-Neural Networks”[34], the authors endeavored to enhance the accuracy of software effort estimation by integrating an Adaptive Neuro-Fuzzy Inference System with the Ant Colony Optimization algorithm, juxtaposed with other evolutionary algorithms like Differential Evolution, Genetic Algorithm, Artificial Neural Network, and Particle Swarm Optimization. Through the application of this innovative model to various widely-used software effort estimation datasets such as Albrecht, Desharnais, and Kemerer, they demonstrated the superiority of their model over the aforementioned algorithms. The enhanced estimation provided by their model could potentially aid software project managers in more accurate project cost estimation, thereby exhibiting significant potential for augmenting precision and reliability in software effort estimation methodologies.

The authors of “Software Cost Estimation using Adaptive Neuro Fuzzy Inference System”[35] endeavored to refine the accuracy of cost and effort estimation in software development through the utilization of the Adaptive Neuro-Fuzzy Inference System (ANFIS) model. Employing the NASA63 data collection and ANFIS models, they achieved an MMRE error of 0.0984 and an estimate accuracy of 0.889. Their objective was to mitigate errors and enhance precision in estimating software project costs and efforts. The proposed method exhibited an accuracy of 96% for training data and 89% for testing data.

The paper titled "Deep Learning Model for Function Point Based Software Cost Estimation - An Industry Case Study"[36] delves into the criticality of precise software cost estimation for both developers and customers. It provides an overview of existing software cost estimation methodologies and introduces a Deep Neural Network (DNN) model designed to augment function point estimation. The investigation illustrates that function point-based estimation can facilitate early software budget evaluation, while deep neural networks can enhance labor productivity and estimation accuracy. The manuscript proposes a method that amalgamates Conditional Random Field (CRF) and Recurrent Neural Network (RNN) techniques to tackle challenges in function point analysis and attain robust outcomes. It expounds on the problem statement, relevant literature, proposed methodology, experimental findings, and a case study conducted at the State Grid Energy Research Institute in China. The approach endeavors to refine function point analysis in software development through the BiLSTM-CRF framework,

as delineated within the document's contents. The outcome of this paper entails the proposition of a method employing a fusion of Conditional Random Field (CRF) and Recurrent Neural Network (RNN) techniques to confront challenges in function point analysis. This methodological approach aims to ameliorate function point analysis by devising a framework leveraging the BiLSTM-CRF model to recognize and categorize function points in software development. Furthermore, the manuscript highlights the successful deployment of this framework across 52 projects, showcasing its efficacy in enhancing function point recognition and analysis in the industry. The findings indicate that despite encountering certain challenges, the proposed method achieved enhancements in accuracy and efficiency, particularly in classifying various function point types. Detailed accounts of the experimentation outcomes, including accuracy percentages for different function point categories and the utilization of BiLSTM and CRF layers, are elucidated within the industry case study section.

The article titled "Software Effort Estimation Accuracy Prediction of Machine Learning Techniques: A Systematic Performance Evaluation"[37] investigates the accuracy of software effort estimation utilizing machine learning techniques. The primary objective of the study is to aid researchers in identifying which machine learning technique yields promising accuracy in effort estimation within software development. The researchers assess the performance of ensemble and individual machine learning techniques utilizing two commonly employed accuracy evaluation metrics. Ultimately, the study concludes that ensemble effort estimation techniques generally yield more promising accuracy in estimation compared to individual techniques.

The research paper titled "Software Effort Estimation Using Machine Learning Methods"[38] focuses on scrutinizing software effort estimation to mitigate issues concerning budget and schedule extensions in software projects by proposing a model integrating machine learning methods. The investigation evaluates various machine learning models utilizing both public datasets and data sourced from software organizations in Turkey. The research discerns that the optimal method for a dataset may vary, indicating that a singular model may not consistently yield optimal outcomes. The outcome of the research underscores that the developed hybrid effort estimation model, integrating model-related data with machine learning methods, performs commendably compared to other machine learning techniques. Experimental findings manifest that machine learning methods such as back-propagation neural networks, regression trees,

radial basis functions, and support vector regression methods outperform traditional parametric models like COCOMO.[39] Significantly, the research accentuates that an evolving learning system leveraging current project data can enhance software effort estimation accuracy vis-à-vis static parametric models. Additionally, the paper underscores the significance of integrating machine learning methods in software effort estimation to adapt to the evolving landscape of software development practices and technologies. The study advocates for a continual learning system capable of updating estimations with new project data to bolster accuracy and adaptability in software effort estimation.

The journal article titled "Deep learning model for end-to-end approximation of COSMIC functional size based on use-case names"[40] discusses the utilization of deep learning techniques to automatically approximate the COSMIC functional size of software applications based solely on their use-case names. This approach aims to offer a more efficient and automated method for functional size estimation, particularly beneficial for agile projects with evolving requirements. The study explores various neural network architectures and pretrained word embeddings to enhance the accuracy of the prediction model. By leveraging deep learning, the research presents a novel approach to functional size approximation without the need for extensive manual intervention in requirements engineering.

The research paper “Development of a Deep Learning Model for Story Points Estimation”[41] introduces a deep learning model, termed Deep-SE, tailored for estimating story points in software development projects. Deep-SE comprises four sequential components: Word Embedding, Long Short-Term Memory (LSTM)-based Document Representation, Recurrent Highway Net (RHWN)-based Deep Representation, and Differentiable Regression. By processing title and description information from issue reports, the model converts them into vector representations, obviating the need for manual feature engineering. The authors utilized pre-trained word embedding matrices derived from issue report corpora. Results demonstrated significant improvements in predictive performance compared to traditional methods, attributed to the model's ability to automatically learn features from raw textual data and discern semantic relations between words. The study underscored the model's efficacy in enhancing predictability and consistency in project planning and management within agile software development contexts.

This research "A model for software effort estimation using pre-trained embedding models"[42] presents a software effort estimation model leveraging pre-trained embedding models, focusing on textual requirements such as user stories. The model employs deep learning techniques to generate vector representations of text, emphasizing the significance of contextual understanding and domain-specific data for model training. Fine-tuning pre-trained models like BERT facilitated enhanced estimation accuracy, with contextualized models exhibiting superior performance. The study underscores the adaptability of the proposed model to diverse project contexts and its capacity to estimate effort for both new and existing requirements. Continuous training and evaluation are highlighted as crucial for model refinement and adaptation to evolving project dynamics, particularly in agile environments.

Table 2.1. Summary of Literature Review and the different existing approaches for using AI in Software Size Estimation

Paper Title	Model/Technique Used	Purpose	Results
Exploring Machine Learning Techniques for Software Size Estimation	Genetic Programming (GP), Neural Networks (NN)	Investigate application of GP and NN for software size estimation, compare effectiveness with traditional methods	Models using function points (FP) had better accuracy with GP compared to NN; models using number of components (NOC) metric showed similar accuracy for both techniques
Enhancing Software Effort Estimation with Ant Colony Optimization Algorithm and Fuzzy-Neural Networks	Adaptive Neuro-Fuzzy Inference System (ANFIS) with Ant Colony Optimization, compared with other evolutionary algorithms	Improve accuracy of software effort estimation, compare with other algorithms	Demonstrated superiority of ANFIS with Ant Colony Optimization over other algorithms in various datasets
Software Cost Estimation using Adaptive Neuro Fuzzy Inference System	Adaptive Neuro-Fuzzy Inference System (ANFIS)	Refine accuracy of cost and effort estimation in software development using ANFIS	Achieved MMRE error of 0.0984 and estimate accuracy of 0.889, with 96% accuracy for training data and

(cont. on the next page)

Table 2.1 (cont.)

			89% for testing data
Deep Learning Model for Function Point Based Software Cost Estimation - An Industry Case Study	Deep Neural Network (DNN) incorporating Conditional Random Field (CRF) and Recurrent Neural Network (RNN)	Improve function point estimation accuracy, propose method combining CRF and RNN	Method using CRF and RNN showed enhancements in accuracy and efficiency, particularly in classifying various function point types
Software Effort Estimation Accuracy Prediction of Machine Learning Techniques: A Systematic Performance Evaluation	Ensemble and individual machine learning techniques	Investigate accuracy of machine learning techniques for effort estimation, compare ensemble vs. individual techniques	Ensemble effort estimation techniques generally yielded more promising accuracy compared to individual techniques
Software Effort Estimation Using Machine Learning Methods	Various machine learning models (e.g., back-propagation neural networks, regression trees, radial basis functions, support vector regression)	Scrutinize software effort estimation, propose hybrid model integrating machine learning methods	Hybrid effort estimation model performed well compared to other machine learning techniques, emphasized importance of continual learning system
Deep learning model for end-to-end approximation of COSMIC functional size based on use-case names	Deep learning techniques, various neural network architectures, pretrained word embeddings	Automate approximation of COSMIC functional size based on use-case names, improve accuracy of prediction model	Achieved superior prediction accuracy using deep learning techniques, particularly with convolutional neural network model
Development of a Deep Learning Model for Story Points Estimation	Deep-SE model (Word Embedding, LSTM, RHWN, Differentiable Regression)	Develop deep learning model for story points estimation, compare with	Significantly improved predictive performance compared to

(cont. on next page)

Table 2.1 (cont.)

		traditional methods	traditional methods, emphasized model's efficacy in enhancing predictability and consistency
A model for software effort estimation using pre-trained embedding models	Pre-trained embedding models (e.g., BERT)	Develop software effort estimation model leveraging pre-trained embedding models, emphasize contextual understanding	Achieved enhanced estimation accuracy through fine-tuning pre-trained models like BERT, highlighted adaptability of proposed model to diverse project contexts

2.5. Challenges and Limitations

Applying artificial intelligence (AI) to COSMIC FSM encounters several challenges that need careful consideration and resolution. One of the foremost challenges is ensuring data quality. As highlighted in the research papers exploring machine learning techniques for software size estimation[25], the effectiveness of AI models heavily relies on the quality and representativeness of the data used for training. However, obtaining high-quality data for COSMIC FSM, which involves quantifying software functionality, can be challenging due to subjective interpretation and documentation quality issues inherent in software development. Ensuring comprehensive and accurate data that captures the nuances of software functionality is crucial for training reliable AI models.

Model complexity is another significant challenge. Many AI models employed for COSMIC FSM estimation, such as deep learning architectures, can be highly complex and difficult to interpret, as seen in papers like "Deep Learning Model for Function Point Based Software Cost Estimation - An Industry Case Study".[28] Complex models may risk overfitting to training data, leading to poor generalization performance on unseen data, and require significant computational resources for training and inference. Simplifying

and optimizing AI models while maintaining their predictive power is essential for practical application in COSMIC FSM estimation.

Interpretability of AI models is another pressing challenge. Despite their high accuracy, many AI models used for COSMIC FSM estimation lack interpretability, making it challenging for stakeholders to understand and trust the model's predictions. This lack of interpretability can hinder the adoption of AI-based COSMIC FSM estimation approaches, particularly in contexts where decision-making relies on transparent and understandable models. Developing AI models with built-in interpretability features or post-hoc interpretability techniques are crucial for gaining trust and acceptance from stakeholders.

Reviewing the limitations of current research in this domain, several key issues emerge. Firstly, limited datasets pose a significant challenge for training AI models, as highlighted in various research papers.[28,25,26,27,29,30] Many AI models for COSMIC FSM rely on small or proprietary datasets, leading to biased models that may not generalize well to new or unseen data. Addressing this limitation requires efforts to collect, annotate, and share comprehensive and diverse datasets representative of real-world software projects.

Bias in AI models is another critical limitation. Biases present in the training data or inherent in the modeling approach can lead to biased predictions, as seen in various research papers.[28,25,26,27,30] Addressing bias in AI models requires careful consideration of data collection, preprocessing, and algorithm design to mitigate potential biases and ensure fairness and equity in the estimation process.

Scalability is also a limitation in current research. While AI models may demonstrate promising results on small-scale datasets, scaling these models to larger software projects or organizations with diverse requirements and constraints can be challenging. Scalability issues may arise due to computational limitations, data availability, or algorithmic constraints, as evidenced by the lack of scalability in certain research papers. Overcoming scalability issues requires developing efficient algorithms and architectures capable of handling large volumes of data and computational resources.

Despite the advancements in AI-based software size estimation methods discussed in the literature, a notable gap exists in addressing the extraction of Objects of Interest (OOIs) from software requirements. OOIs play a crucial role in COSMIC FSM, as they represent the functional components of a software system and are essential for

accurately quantifying software functionality. However, none of the research papers reviewed in this analysis explicitly tackle the task of OOIs extraction. Extracting OOIs from textual requirements poses several challenges, including ambiguity, variability, and context-dependency. Addressing these challenges requires innovative natural language processing (NLP) techniques capable of automatically identifying and categorizing OOIs based on their semantic and syntactic properties, or a much more innovative method which we will explore in this thesis. By incorporating OOIs extraction into AI-based COSMIC FSM estimation approaches, researchers can enhance the granularity and accuracy of software size measurements, ultimately improving decision-making processes in software development projects. Closing this gap represents an important avenue for future research in the field of AI-based software size estimation.

In conclusion, addressing the challenges and limitations of applying AI to COSMIC FSM requires interdisciplinary collaboration and innovative solutions. Improving data quality, simplifying model complexity, enhancing interpretability, increasing datasets quality, reducing bias, and addressing scalability issues are critical steps towards developing robust and reliable AI-based COSMIC FSM estimation approaches. By addressing these challenges, AI has the potential to revolutionize software size estimation and enhance decision-making processes in software development projects.

Table 2.2. Comparison of usage of AI in COSMIC, Event Points, or Size Estimation in general

	COSMIC POINTS	**Object of interest**	**Event Points**	**Event Names**	**Size Estimation**
Existing Work	-	-	-	-	X
Proposed Technique in this Thesis (Figure 2)	X	X	X	X	X

Chapter 3

Methodology

This chapter explains how we conducted the study step by step. It's divided into three main parts: Data Collection and Preprocessing, COSMIC & Event Points Measurement, and Model Selection. We'll start by talking about how we gathered and prepared the data we needed. Then, we'll explain how we did the COSMIC Functional Size and Event Points measurements and ensured both: data quality, and correctness of measurements. Finally, we'll discuss how we chose the models we used to analyze the data. By breaking down each part of our methodology, we aim to provide a clear understanding of how we carried out the study and why.

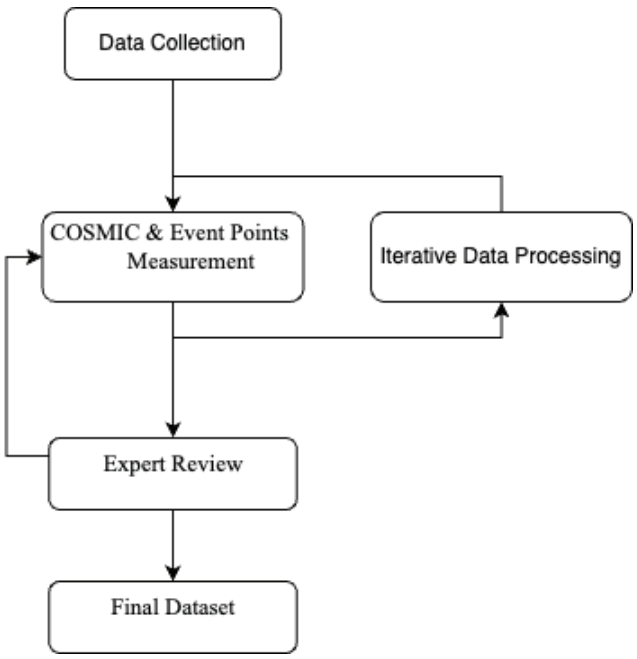

Figure 3.1. Methodology used in COSMIC and Event Points measurements to create the used Dataset. Data Preprocessing and Adaptation.

3.1. Data Collection and Preprocessing

In this section, we detail the process undertaken to assemble a comprehensive dataset of use cases spanning diverse projects, essential for the subsequent phases of analysis and modeling. The dataset, constituting the foundational part of our research, was meticulously curated to ensure representativeness and relevance to our study objectives.

Project Variety and Selection: To encapsulate the multifaceted nature of software systems, a deliberate effort was made to source use cases from a heterogeneous array of projects. These projects spanned a spectrum of domains, including healthcare, education, and e-commerce. This deliberate diversification ensured a broad representation of real-world scenarios, enriching the dataset with varying complexities and intricacies inherent to different application domains.

Data Collection Process: Domain experts from the projects mentioned meticulously crafted use cases, distilling essential functionalities and scenarios. Their intimate understanding ensured the authenticity and relevance of each use case to our study objectives.

Statistical Overview: The collected dataset comprises a total of 2041 unique use cases, sourced from 26 distinct projects. The shortest sentence within the dataset contains 3 words, while the longest extends to 156 words. On average, each use case consists of 13 words, with a standard deviation of 10.83, reflecting the variability in use case complexity across different projects.

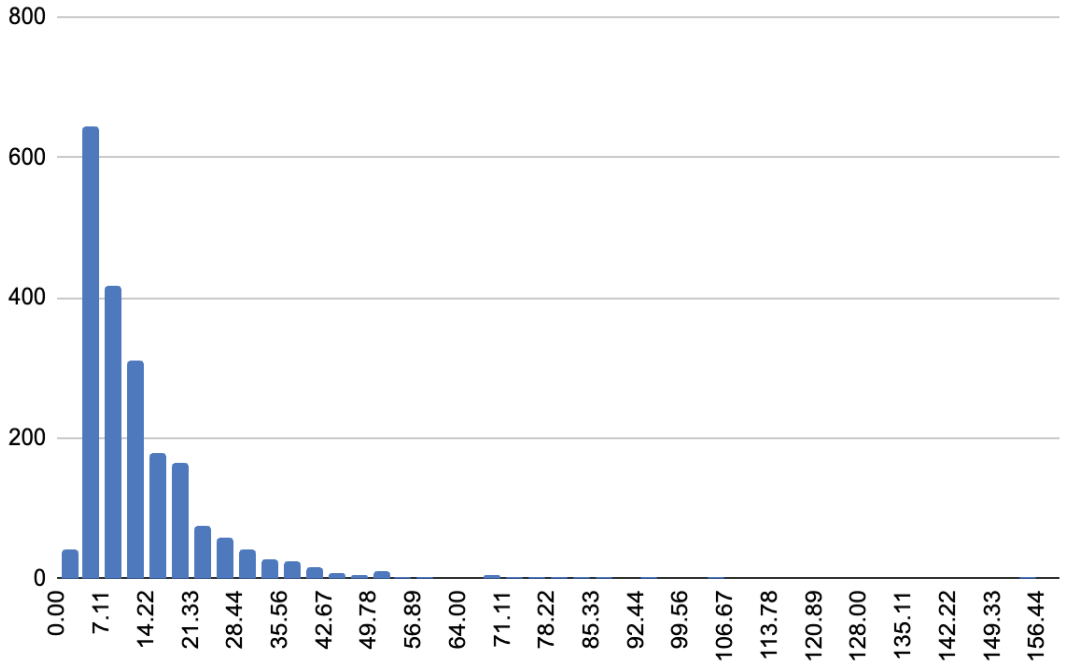

Figure 3.2. Histogram of wordcount in the dataset.

Statistic	Word size	Event driven size
Min	13.0	0.0
Max	1000.0	2.0
Average	83.76	0.83
Median	65.0	1.0
Variance	4714.67	0.14
Standard Deviation	68.66	0.38
Total	170711.0	1694.0

Figure 3.3. Statistics about dataset of Event Points

Statistic	Word size	COSMIC func size
Min	13.0	0.0
Max	1000.0	4.0
Average	83.8	1.23
Median	65.0	1.0
Variance	4714.2	0.6
Standard Deviation	68.66	0.77
Total	170709.0	2511.0

Figure 3.4. Statistics about dataset of COSMIC size

Data Preprocessing: Prior to measurement, the collected use cases underwent rigorous preprocessing to standardize formatting, correct errors, and eliminate redundancies. This preprocessing phase aimed to enhance the quality and consistency of the dataset, ensuring robustness and reliability in subsequent measurements and analyses. Additionally, the use cases were further refined by splitting them into granular use case lines. This approach allowed for more detailed estimations and analysis, enriching the dataset with finer granularity and depth.

3.2. COSMIC & Event Points Measurement

This chapter delves into the intricacies of measuring COSMIC Points and Event Points as integral components of our methodology. The meticulous process of measurement involved manual extraction, analysis, and validation, culminating in a comprehensive understanding of the functional complexities inherent in the software systems under study. This phase of our study involved extensive work and review spanning approximately three months, with an average commitment of around 10 hours per week. Through this meticulous methodology, we endeavor to provide a transparent account of our research process and its underlying rationale.

- Manual COSMIC Points Estimation:

The measurement of COSMIC Points commenced with a meticulous manual extraction process, wherein the object of interest within each use case was identified. This process involved dissecting the use cases into constituent components, discerning the

functional boundaries, and assigning COSMIC weights based on the established criteria. Each use case underwent meticulous scrutiny to ensure accuracy and consistency in the estimation process.

- Event Points Measurement and Event Names Extraction:

In tandem with COSMIC Points measurement, Event Points were rigorously evaluated to capture the dynamic aspects of software behavior. Event Points measurement involved discerning event triggers, actions, and outcomes within the use cases, facilitating a granular understanding of system interactions. Concurrently, Event Names were extracted to catalog and categorize the diverse range of events observed across the dataset.

- Iterative Dataset Processing:

The dataset underwent iterative processing to refine and validate the COSMIC and Event Points estimations. Each iteration involved meticulous review and adjustment, ensuring alignment with the underlying principles and objectives of the study. The iterative nature of the processing facilitated continuous improvement and optimization of the estimation methodologies, enhancing the robustness and reliability of the dataset.

- Expert Review:

The culmination of the estimation process was marked by a comprehensive review conducted by domain expert Hüseyin ÜNLÜ. His invaluable insights and expertise provided a critical perspective on the accuracy and validity of the estimations, further bolstering the credibility of the findings. Hüseyin's rigorous review underscored the collaborative nature of our methodology, highlighting the importance of domain expertise in ensuring the fidelity and relevance of the analysis.

3.4. Model Selection

This chapter elucidates the rationale behind the selection of Sequence Regression models for size prediction and fine-tuned summarization and language models for Event Name prediction and Object of Interest (OOI) extraction, respectively. Each model was meticulously chosen to address specific challenges inherent in the prediction tasks, leveraging their unique capabilities to enhance the accuracy and efficacy of our analyses.

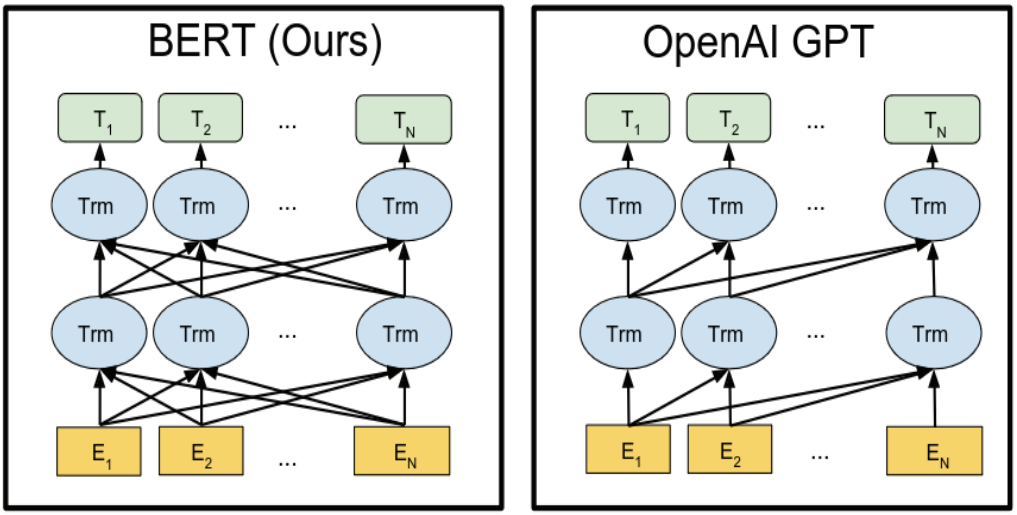

Figure 3.5. Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. BERT representations are jointly conditioned on both left and right context in all layers.[31]

3.5. BERT for Word Embedding

Bidirectional Encoder Representations (see Figure 3.5) from Transformers (BERT) has emerged as a revolutionary model in the field of natural language processing (NLP)[43], renowned for its ability to capture contextual nuances and semantic relationships within textual data. Developed by Google AI's research team, BERT utilizes the Transformer architecture (see Figure 3.6) a self-attention mechanism that enables efficient processing of sequential data while preserving contextual information.[44]

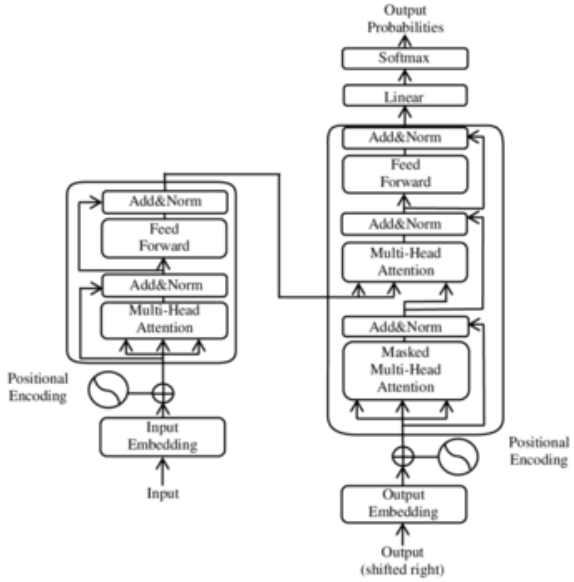

Figure 3.6. An illustration of main components of the transformer model.[44]

- Understanding BERT:

At its core, BERT employs a bidirectional approach to language modeling, allowing it to leverage both left and right context during training. Unlike traditional language models that process text in a unidirectional manner, BERT comprehensively captures contextual dependencies, resulting in more robust and nuanced representations of language.

- Pre-training and Fine-tuning:

BERT is pre-trained on large corpora of text using unsupervised learning objectives, such as masked language modeling and next sentence prediction. This pre-training phase enables BERT to learn rich, context-aware representations of words and phrases, encoding semantic information within its hidden layers.

Moreover, BERT's architecture lends itself well to fine-tuning on downstream tasks, such as word embedding. By fine-tuning BERT on task-specific datasets, it can adapt its learned representations to the specific nuances and characteristics of the target domain, enhancing performance on a wide range of NLP tasks.

- Advantages of BERT for Word Embedding:

Contextual Embeddings: BERT generates contextualized word embeddings that capture the nuanced meanings of words based on their surrounding context. This contextual understanding leads to more accurate representations of word semantics compared to traditional static word embeddings.

Domain Adaptability: BERT's pre-trained representations can be fine-tuned on task-specific data, making it highly adaptable to different domains and applications. This flexibility allows BERT to effectively capture domain-specific semantics and nuances, enhancing its performance on word embedding tasks within specialized domains such as software engineering.

- What is BERT_SE?

BERT_SE is a contextualized pre-trained language representation model specifically designed for software engineering (SE) domain tasks. BERT_SE aims to improve the classification of software requirements by recognizing specific and relevant terms in the context of SE, resulting in enhanced performance for various NLP tasks in the SE area.[45]

- State-of-the-Art Performance

BERT has consistently demonstrated state-of-the-art performance across a wide range of NLP tasks, including word embedding, sentiment analysis, and named entity recognition. Its robust architecture and effective utilization of contextual information contribute to its superior performance and generalization capabilities.

3.6. Regression Model for Size Prediction

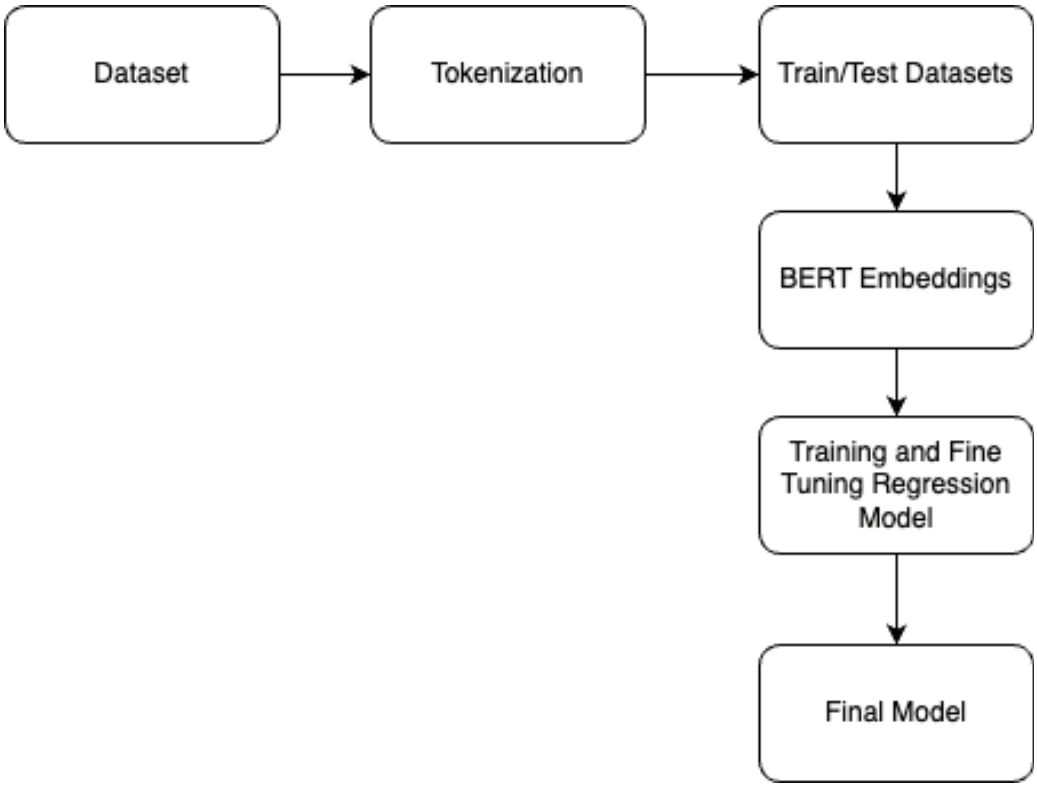

Figure 3.7. Steps taken for creation of a Regression Model for COSMIC & Event Points Estimation

The prediction of software sizes necessitated a robust and interpretable model capable of capturing the intricate relationships between input features and output sizes. To address this challenge, sequence regression models emerged as one of preferred choices due to their versatility and well-established framework for continuous value prediction.[46,47]

The utilization of BERT for Sequence Regression[48] models offered several advantages for this task. By training these models on a diverse dataset, they were able to discern and extrapolate intricate patterns present within the data. This facilitated accurate size estimations for the use cases under consideration.

For the implementation of the sequence regression models, we adopted a rigorous approach. Specifically, each fold within the dataset was subjected to model training and evaluation, employing techniques such as K-fold cross-validation to ensure robustness and reliability of the results. Within each fold, the model was initialized and trained using an AdamW[49] optimizer with a carefully chosen learning rate. Additionally, the use of a

Mean Absolute Error (MAE) and Mean Squared Error (MSE) loss function allowed for effective measurement of the model's performance

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |actual\_ef\ f - predicted\_ef\ f_i| \quad (1)$$

$$MSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(|actual\_ef\ f - predicted\_ef\ f_i|)^2} \quad (2)$$

Throughout the experimentation process, meticulous attention was paid to training scope, number of splits, and shuffle parameters, ensuring consistency and reproducibility of the results. The integration of sequence regression models underscored the effectiveness of our approach in addressing the complexities inherent in software size prediction tasks

The utilization of sequence regression models exemplifies a principled approach to model selection, leveraging the strengths of this model type to achieve accurate and reliable predictions in the domain of software engineering.

3.7. Fine-tuned Summarization Model for Event Name Prediction

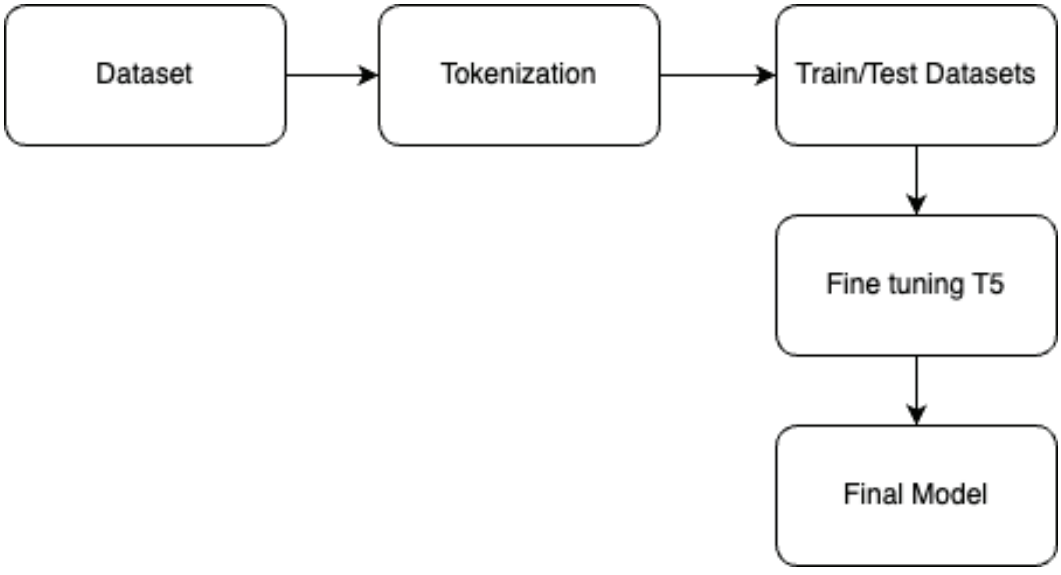

Figure 3.8. Steps taken for fining tuning T5 model for Event Names extraction.

Predicting event names presented a unique challenge due to the nuanced and context-dependent nature of event descriptors within the use cases. To address this challenge effectively, a fine-tuned summarization model was employed, specifically leveraging the T5 (Text-To-Text Transfer Transformer) model[50] fine-tuned on our dataset. This selection was driven by the model's remarkable ability to distill salient event names from textual descriptions with exceptional precision and context awareness.

The T5 model, developed by Google AI, represents a cutting-edge pre-trained language model renowned for its text-to-text framework.[50] This model has garnered significant attention in the field of natural language processing (NLP) owing to its remarkable versatility and efficacy across diverse domains.

The fine-tuning process of the T5 model[51] on our dataset entailed training it on an 90% of our use case descriptions. Through this meticulous process, the model honed its comprehension of textual data, adapting seamlessly to the nuances and specific demands inherent in event name prediction tasks.

Moreover, the fine-tuned T5 model demonstrated proficiency in capturing semantic nuances and contextual cues embedded within the text. This enabled the model to generate event name predictions with precision and contextual relevance. Leveraging its robust architecture and comprehensive pre-training, the T5 model excelled in distilling key information from input texts, a capability that proved particularly advantageous for event name prediction tasks.

One notable aspect of the T5 model is its text-to-text framework, which facilitates the generation of textual output from input. This feature played a pivotal role in producing concise and meaningful event name predictions, leveraging the rich contextual information encapsulated within the use case descriptions.

The choice of the fine-tuned T5 model for event name prediction offered several advantages. Its robust architecture, extensive pre-training, and capacity to generate high-quality summaries positioned it as a suitable solution for addressing the inherent complexities of event name prediction tasks. Furthermore, the training process incorporated metrics such as ROUGE[52] to evaluate the quality of generated summaries, ensuring rigorous assessment and refinement of the model's performance.

$$\text{ROUGE-N} = \frac{\sum_{S \in \{ReferemceSummaries\}} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in \{ReferenceSummaries\}} \sum_{gram_n \in S} Count(gram_n)}$$

Figure 3.9. Original formula of ROUGE-N as an n-gram recall between a candidate summary and set of reference summaries.[52]

3.8. Fine-tuned Large Language Model for Object of Interest (OOI) Extraction

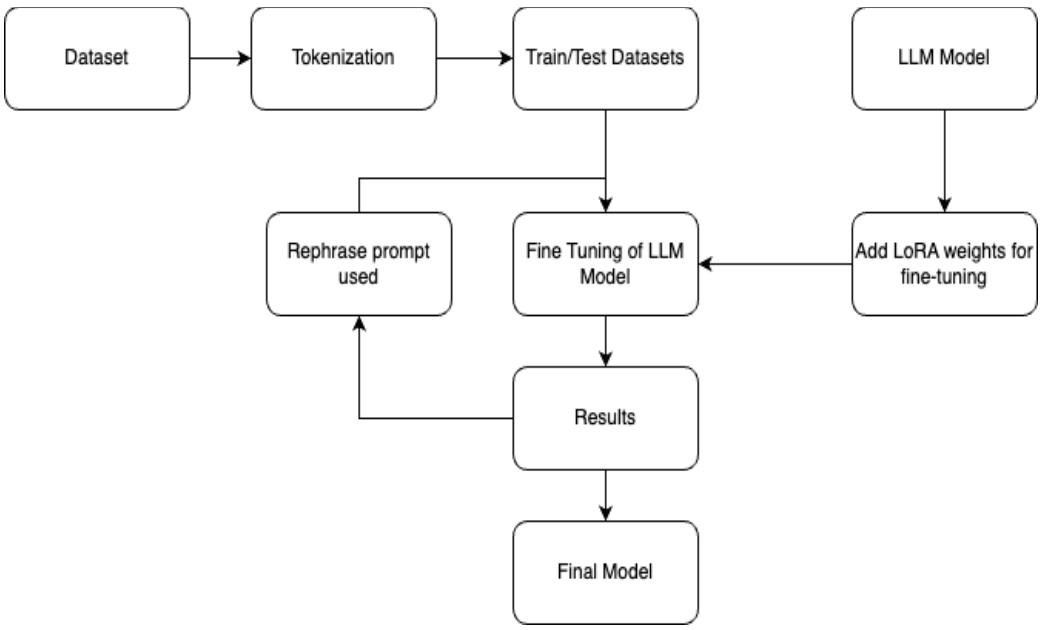

Figure 3.10. Steps taken for creation of fine-tuned LLM model used for COSMIC Object of interest extraction

Extracting the Object of Interest (OOI) from use cases necessitated a model capable of understanding and synthesizing complex textual information while preserving syntactic and semantic coherence. To address this requirement, we employed the state-of-the-art Mistral-7B language model[53] as the base model for fine-tuning. Mistral-7B, renowned for its exceptional performance on a wide range of natural language understanding tasks, provided a strong foundation for our OOI extraction task.

- Low-Rank Adapter (LoRA) Fine-Tuning:

To tailor Mistral-7B to the specific task of OOI extraction, we employed the innovative Low-Rank Adapter (LoRA) fine-tuning technique. LoRA, a recent advancement in transfer learning, enables efficient fine-tuning of large language models while preserving their pre-trained knowledge (see Figure 4) By leveraging low-rank projections, LoRA facilitates task-specific adaptation without compromising the robustness or generalization capabilities of the base model.[54]

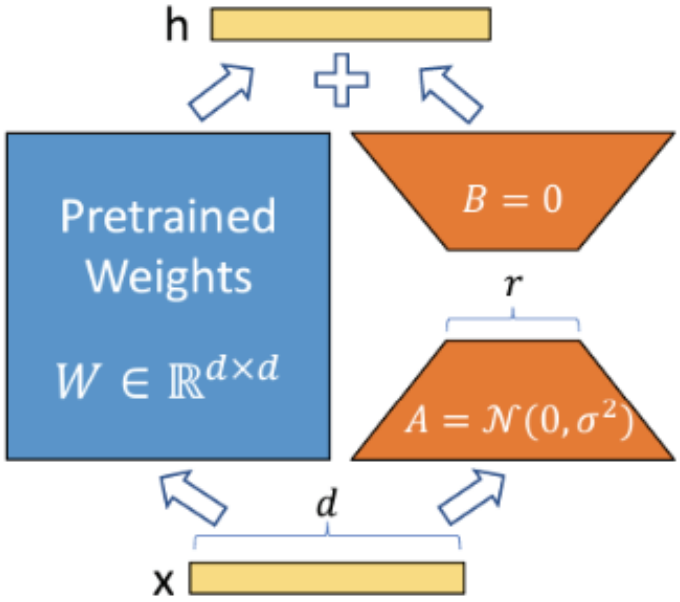

Figure 3.11. Reparameterization. LoRA only train A and B.[54]

- Advantages of Mistral-7B and LoRA:

The choice of Mistral-7B as the base model offered several advantages, including its extensive pre-training on diverse textual data, enabling it to capture intricate linguistic patterns and semantics inherent in software use cases. Additionally, the integration of LoRA for fine-tuning provided a principled approach to task-specific adaptation, optimizing the model for OOI extraction while minimizing the risk of overfitting or catastrophic forgetting.

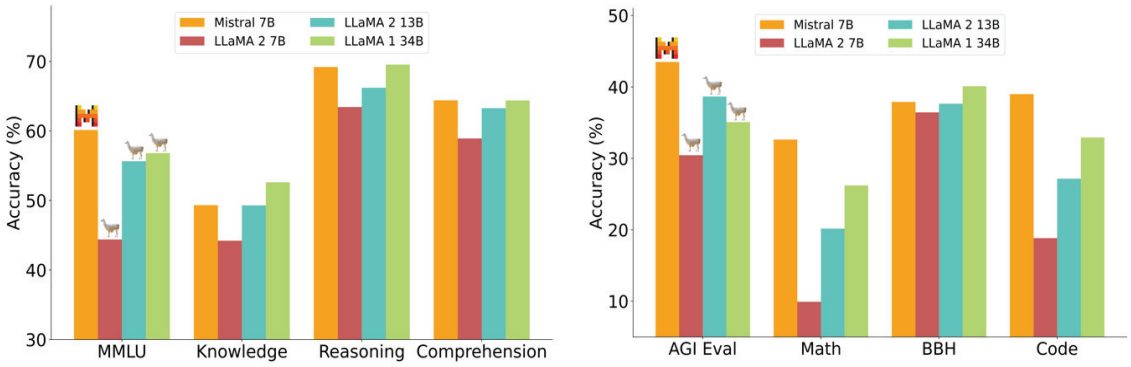

Figure 3.12. Benchmarks of performance of Mistral 7b and different Llama models.[55]

3.9. Rationale for Model Selection

The selection of sequence regression models for size prediction and fine-tuned summarization and language models for Event Name prediction and OOI extraction was driven by consideration of the inherent complexities and requirements of each prediction task. By leveraging the unique strengths of each model type, we aimed to maximize predictive accuracy and fidelity while mitigating the impact of inherent uncertainties and variability within the dataset.

CHAPTER 4

Results and Discussion

This research endeavors to introduce a novel Software Estimation Methodology, designed to automate the process of generating COSMIC Points and extracting Objects of Interest from User Case Line requirements through the implementation of a Machine Learning (ML) architecture. Furthermore, a parallel ambition is set forth for the Event Points and the extraction of associated Event Names. To appraise the efficacy of our methodology, the following inquiries were formulated:

2.1 Can a Machine Learning model achieve a high predictive accuracy in discerning the granular categories of COSMIC Points (i.e., write, read, exit, entry) and Event Point types (i.e., Interaction, Communication, Processing) and the cumulative points COSMIC Points Total and Event Points Total?

2.2 Can a Summarization Model adeptly generate syntactically correct event names?

2.3 *Is it viable for a Language Model to accurately generate Objects of Interest based on provided Use Case?*

In this chapter, the findings of various models are delineated. Initially, an exposition is made concerning the outcomes derived from three distinct categories of Event Points: Event Processing, Event Communication, and Event Interactions. Subsequently, the results of total Event Points are elucidated, both through the application of a sequence regression model and via aggregation of predictions across the categories. Analogously, a similar treatment is extended to COSMIC points, wherein an account of the Write, Read, Entry, and Exit outcomes is provided prior to presenting the aggregate results.

Statistic	cosmic_read	cosmic_write	cosmic_exit	cosmic_entry	cosmic_total
Min	-0.09	-0.12	-0.13	-0.11	0.35
Max	1.21	1.2	1.23	1.11	2.35
Average	0.2	0.19	0.36	0.5	1.23
Median	0.02	0.01	0.03	0.35	1.01
Variance	0.13	0.14	0.22	0.21	0.18
Standard Deviation	0.36	0.37	0.47	0.46	0.43

Figure 4.1. Results of COSMIC predictions.

Statistic	event_communication	event_process	event_interaction	event_total
Min	-0.12	-0.14	-0.1	0.76
Max	1.2	1.14	1.2	2.83
Average	0.09	0.85	0.4	1.27
Median	0.02	0.96	0.09	1.02
Variance	0.05	0.1	0.19	0.22
Standard Deviation	0.22	0.31	0.44	0.47

Figure 4.2. Results of Event Points predictions.

4.1. Results of Regression Models

1. Event Points

In terms of Event Processing, the model demonstrated consistent performance across folds, with an average Mean Squared Error (MSE) of 0.0502 and an average Mean Absolute Error (MAE) of 0.0847. Specifically, the training process unfolded over six epochs per fold, revealing a progressive reduction in both training and testing losses. Moving to Event Communication, similar evaluation metrics were observed, with an average MSE of 0.0914 and an average MAE of 0.1359. The model exhibited convergence over six epochs per fold, with a diminishing trend in both training and testing losses. Lastly, Event Interaction evaluations displayed an average MSE of 0.0700 and an

average MAE of 0.1173, illustrating a consistent training trajectory across folds, marked by a decline in losses over the course of six epochs per fold. These findings collectively underscore the model's efficacy in capturing and predicting nuanced aspects of event-related processes, paving the way for informed decision-making in event management contexts.

In addition to evaluating the performance of the models through traditional metrics like Mean Squared Error (MSE) and Mean Absolute Error (MAE), it's essential to consider the accuracy of the predicted results. The reported accuracies for Event Communication, Event Processing, and Event Interaction, standing at 0.9, 0.94, and 0.92 respectively, indicate a high degree of alignment between the predicted values and the actual observations. (see Figure 4.3)

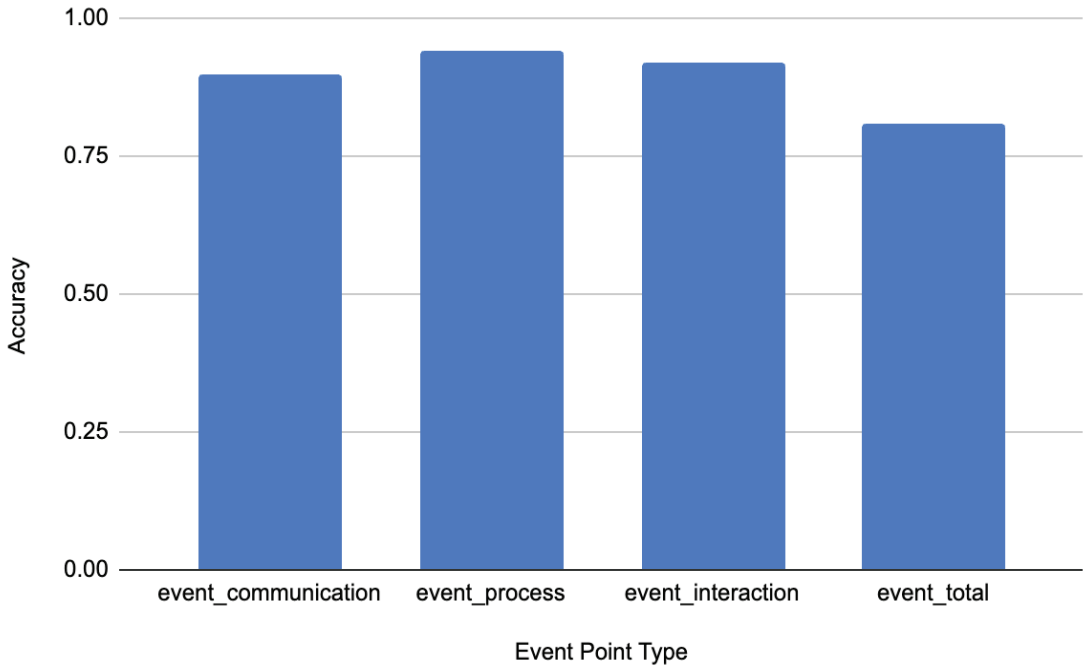

Figure 4.3. Event Point Prediction Accuracies

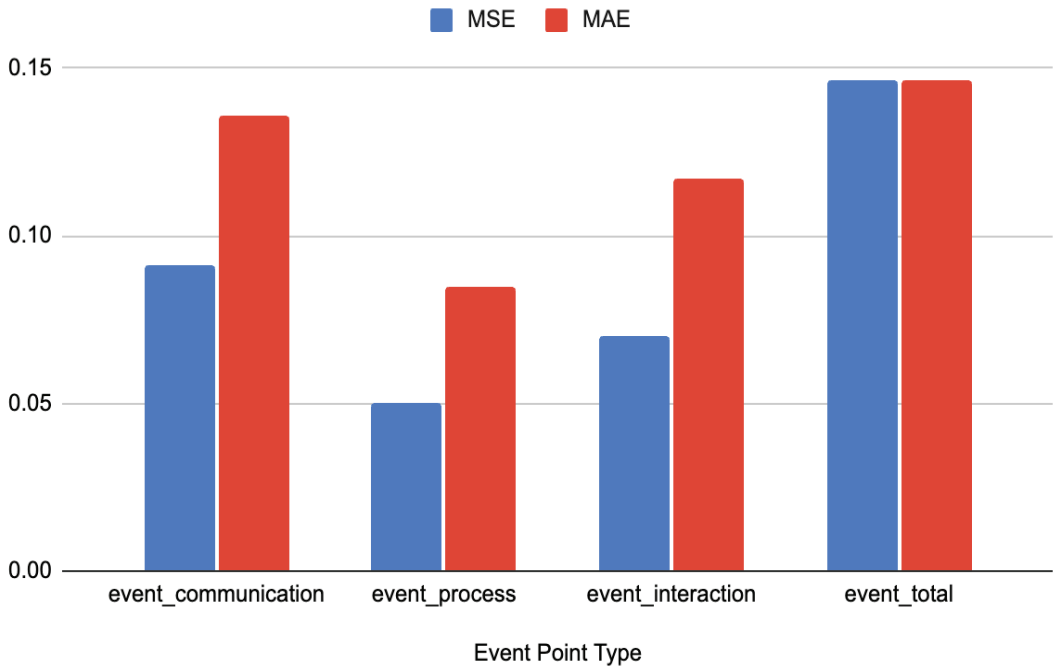

Figure 4.4. Event Point Training Model MSE and MAE Average results.

2. COSMIC Points

The COSMIC model demonstrates impressive accuracy across various movement types, with COSMIC Entry leading at 0.95, closely followed by COSMIC Write and COSMIC Exit at 0.94 each, and COSMIC Read at 0.92. When examining mean squared error (MSE) and mean absolute error (MAE) for individual movement types, COSMIC Entry stands out with the lowest MSE of 0.048 and a correspondingly low MAE of 0.0858. For COSMIC Read, the MSE is 0.0601 with an MAE of 0.0999, while COSMIC Write achieves an MSE of 0.0507 and an MAE of 0.082. Similarly, COSMIC Exit displays an MSE of 0.0574 and an MAE of 0.0949. These results underscore the model's efficacy in accurately predicting COSMIC movement types, showcasing its reliability and performance across different scenarios. (see Figure 4.5)

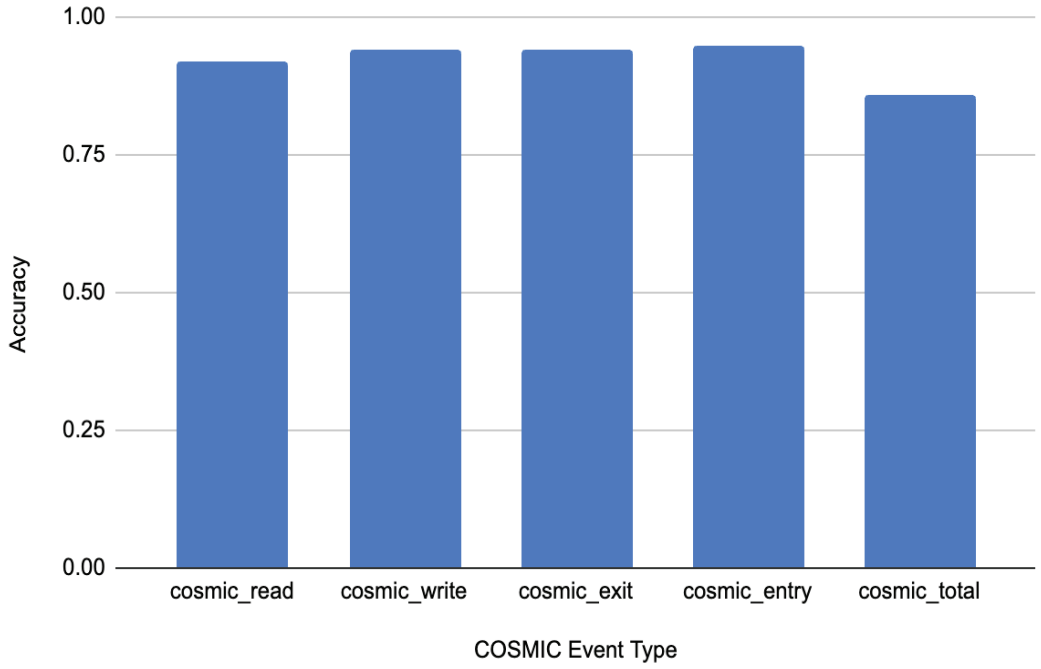

Figure 4.5. COSMIC Point Prediction Accuracies

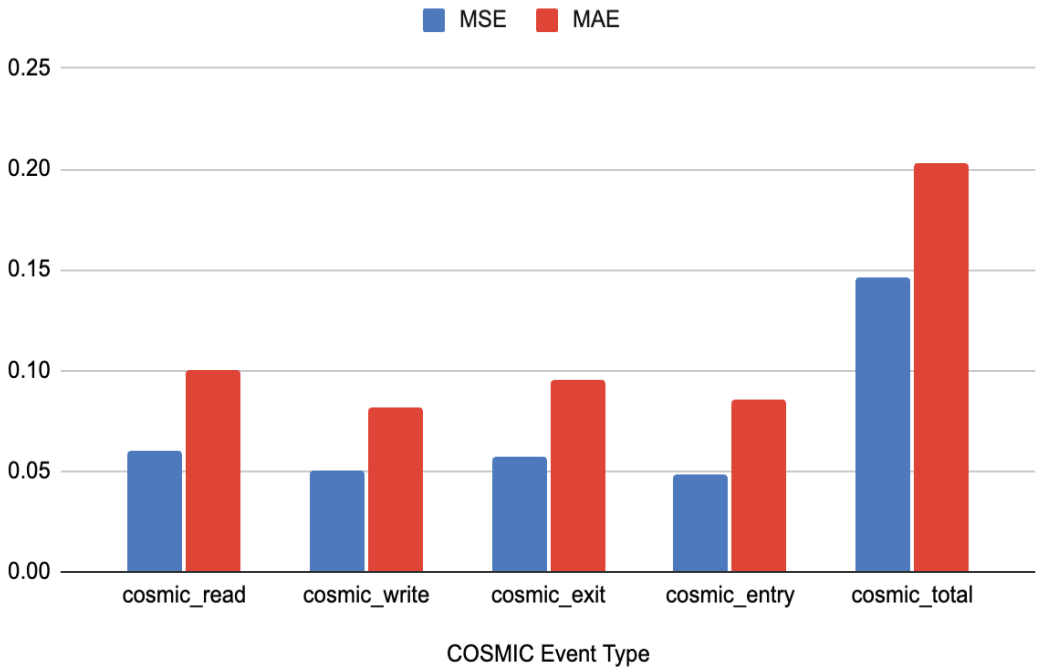

Figure 4.6. COSMIC Points Training Model MSE and MAE Average results.

3. Event Points Total and COSMIC Point Totals

In comparing the results, for Event Total prediction, the regression model achieved an accuracy of 0.81, while the aggregation approach performed notably better

with an accuracy of 0.92, indicating the superiority of aggregation in this context. Conversely, for COSMIC Total prediction, the regression model outperformed the aggregation method with an accuracy of 0.86 compared to 0.83, suggesting that the regression model is more suitable for predicting COSMIC Total function points.

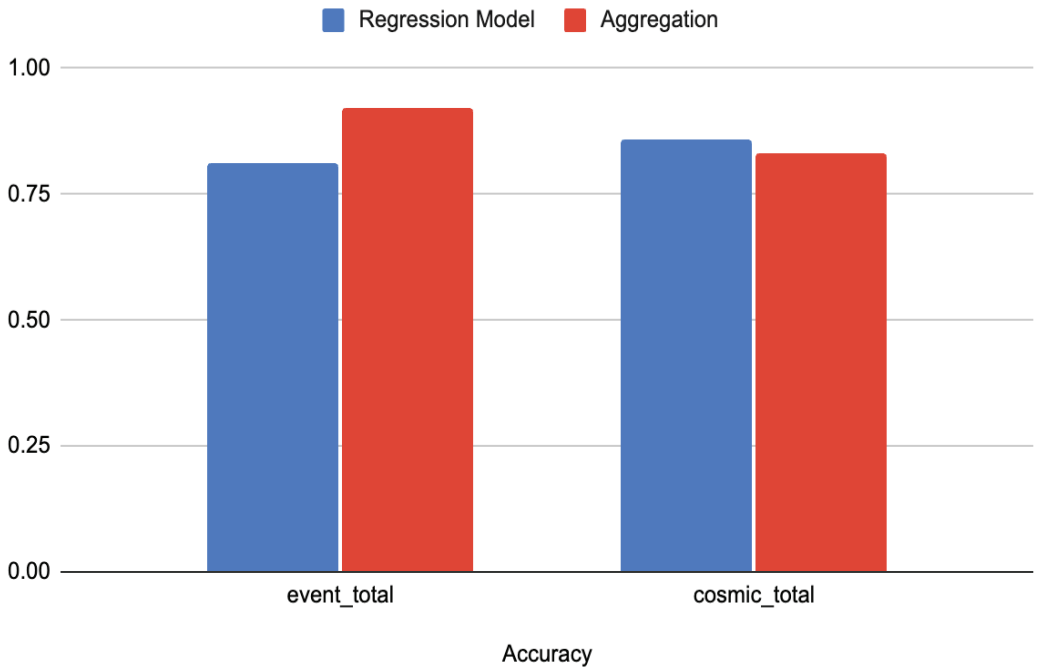

Figure 4.7. Comparison between Regression vs Aggregation for totals.

4. Results of BERT_SE

The BERT_SE model achieved strong accuracies across various COSMIC movement types, with COSMIC Read at 0.93, COSMIC Write at 0.94, COSMIC Exit at 0.94, and COSMIC Entry at 0.95. For event-specific accuracy, Event Process scored the highest at 0.94, Event Interaction also performed well at 0.92. However, Event Communication had a slightly lower accuracy of 0.89. In terms of overall performance, Event Total lagged at 0.82.

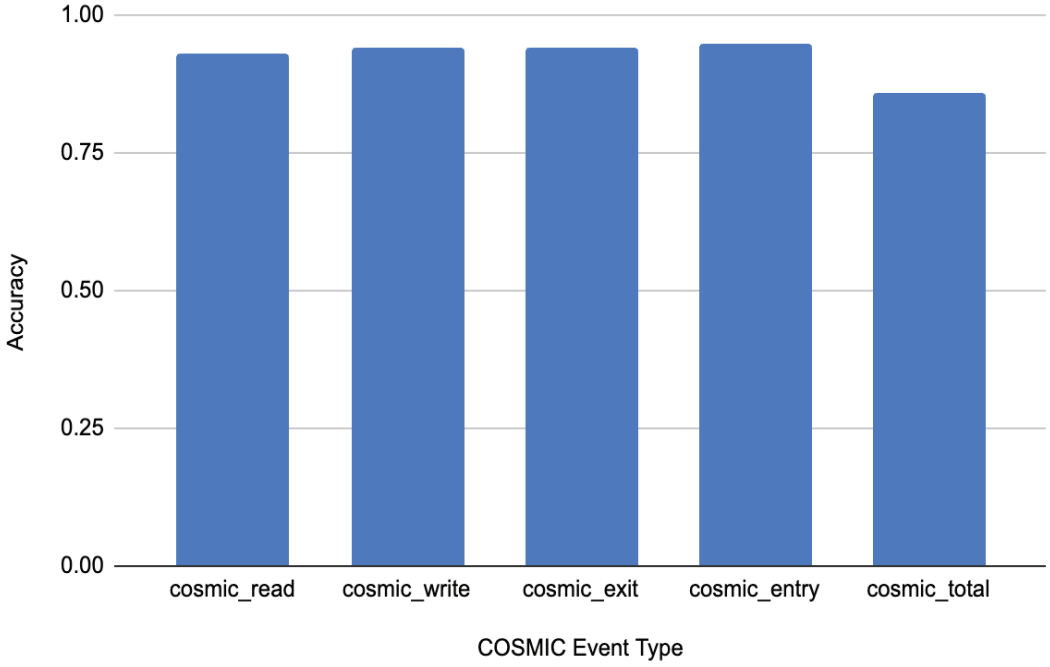

Figure 4.8. COSMIC Point Prediction Accuracies using BERT_SE.

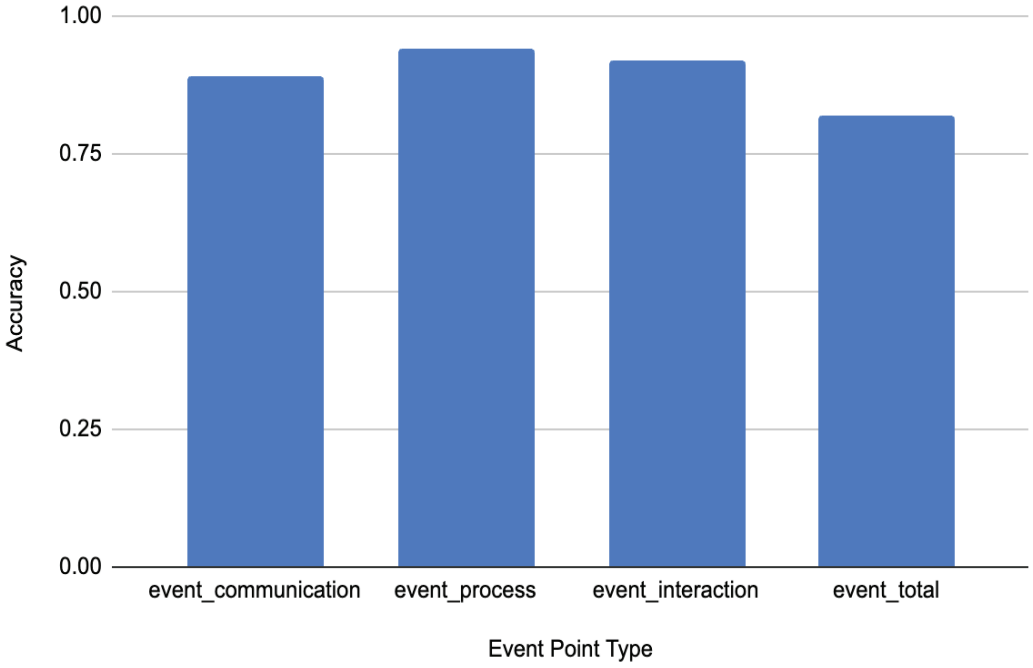

Figure 4.9. Prediction Accuracies for Event Points using BERT_SE

4.2. Results of Summarization Model

Across the five epochs, a discernible pattern emerges in the training loss, which progressively diminishes from an initial 1.045 to a final 0.968738. Conversely, the validation loss appears to stabilize, fluctuating moderately within the range of 0.7047 to 0.7314.

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
1	No log	1.045254	0.704700	0.439300	0.674100	0.673600	5.803900
2	No log	0.998770	0.716900	0.475400	0.684500	0.684200	6.225500
3	No log	0.979030	0.730600	0.485300	0.692700	0.693600	6.078400
4	No log	0.968738	0.731400	0.488000	0.699100	0.698500	6.039200
5	0.997200	0.964659	0.730600	0.493300	0.701600	0.701000	6.000000

Figure 4.10. T5 model fine tuning metrics

Notably, manual inspection of the results reveals an excellent performance. The Rouge scores exhibit incremental enhancements over successive epochs, with Rouge1 values ascending from 0.4393 to 0.7306, Rouge2 from 0.6741 to 0.4933, and RougeL from 0.6736 to 0.7016. Similarly, RougeLsum displays a slight amelioration from 5.8039 to 6.0392. Impressively, the generated length remains consistent at approximately 6.

Table 4.1. Comparison between original Use Case Line, original event name related to it, and the Predicted Event Name using summarization model.

No	Use Case Line	Original Event Name	Predicted Event Name
1	Website redirects the applicant to a relevant success page. Website renders the assessment form.	Website renders the assessment form	Website redirected applicant to success page
2	Open display albums page. Click on specific album.	Album clicked	Album clicked
3	Display all Al Fakhoora house visits. Click delete button for specific Al Fakhoora house visits.	Delete house visits confirmed Delete house visits confirmed	Delete house visits confirmed Delete house visits confirmed

(cont. on the next page)

Table 4.1 (cont.)

	Confirm message is shown. Click Ok button.		
4	The web app retrieves user data from the Database. The web app displays the data of the selected waiter	Web app displayed selected waiter	Web app displayed waiter data
5	The Editor selects a reviewer for the article.	Editor selected an author	Editor selected review

4.3 Results Of LLM Model

The assessment of the Object of Interests (OOIs) generation using Mistral revealed an exact match accuracy of 0.79. Exact match accuracy, in this context, denotes the percentage of generated OOIs that precisely match the ground truth labels. It quantifies the fidelity of the model's outputs by measuring the frequency with which the generated labels align exactly with the expected ones, without any disparities. For instance, if an image is annotated with the label "dog," an exact match occurs when the model's prediction also identifies the object as "dog." Thus, the attained exact match accuracy of 0.79 signifies that approximately 79% of the model's generated OOIs correspond precisely to the expected annotations.

It's notable that the accuracy discussed earlier could be improved through post-processing of the outcomes. For instance, we observed instances where the initial Object of Interest was labeled as "Schedule," but our model identified it simply as "WeeklySchedule." While both designations may be considered valid, our emphasis on exact matches led to a reduction in overall accuracy due to such discrepancies.

CHAPTER 5

Conclusions and Future work

In this conclusion, we will first address the core research questions identified at the outset of this study. By revisiting these questions, we aim to summarize how our findings provide answers and contribute to the broader understanding of the topic. Following this, we will discuss the efficiency and significance of the methods employed and the results obtained. This evaluation will highlight the practical implications and the impact of our research on the field. Finally, we will outline potential directions for future work, suggesting areas where further investigation could enhance or build upon our findings.

5.1. Answers for the Proposed Research Questions

1. Can a Machine Learning model achieve a high predictive accuracy in discerning the granular categories of COSMIC Points (i.e., write, read, exit, entry) and Event Point types (i.e., Interaction, Communication, Processing) and the cumulative points COSMIC Points Total and Event Points Total?

As outlined in the preceding chapter, the regression models developed in this study demonstrated remarkable accuracy, notwithstanding the dataset's size limitations. With an average accuracy rate of 92%, this research provides substantial validation for the initial research inquiry posited.

2. Can a Summarization Model adeptly generate syntactically correct event names?

The initial model exhibited proficiency in summarizing provided text; however, its output primarily comprised summaries rather than appropriately crafted event names. Conversely, in certain instances, the model faltered, failing to generate a summarized form (event name) altogether. For instance, when tasked with summarizing the phrase

"Delete house visits confirmed" the resulting output mirrored the input verbatim, differing only in the presence of an additional period at the conclusion.

Upon scrutiny of Table 3, our refined model yielded exemplary outcomes when subjected to test data, notwithstanding its retraining on a comparably modest dataset, denoted herein as 2041 instances. Notably, the predictions predominantly mirrored the original text, attesting to the model's proficiency. Moreover, in select cases, the model demonstrated competence in generating cohesive and linguistically accurate event names, characterized by appropriate grammatical structure, notably favoring the passive voice.

The discerned performance of our refined model not only underscores its efficacy but also serves to validate the core research question posited. This affirmative validation accentuates the model's capacity to fulfill the intended function of generating apt event names, thereby lending credence to the overarching objectives of our study.

3. Is it viable for a Language Model to accurately generate Objects of Interest based on provided Use Cases?

The obtained exact match accuracy in this thesis marks a promising beginning for utilizing Language Models (LLMs) in Object of Interest extraction. Although the achieved result in this aspect may appear lower compared to other facets of this study, it does not diminish its significance or impact. This is primarily due to the inherent differences between the tasks of categorization and token generation, rendering them incomparable. Furthermore, as discussed in the preceding chapter, the accuracy can be further improved through straightforward post-processing techniques. Hence, the affirmative answer to the research question emerges: Yes, LLMs are capable of accurately generating Objects of Interest.

5.2. Efficiency and Significancy of usage of proposed estimation method

The reluctance of companies to engage in SSM processes is primarily attributable to the substantial manual effort and specialized expertise required, which not all organizations can afford. As highlighted in the introduction, and in literature review, this presents a significant barrier to entry for many companies, hindering their ability to leverage SSM tools effectively.

Moreover, the temporal demands of manual SSM approaches, as evidenced in the preparation of the dataset of this thesis, often entail protracted durations spanning several weeks or even months. Such prolonged timelines are antithetical to the dynamic pace of modern software development endeavors, impeding agility and responsiveness in project planning and execution. The utilization of artificial intelligence (AI) in SSM, as proposed in this thesis, offers a transformative solution to mitigate these challenges.

By leveraging AI-driven models, the time required for size measurement is dramatically reduced from weeks or months to mere seconds, streamlining the estimation process and enhancing operational efficiency. Furthermore, the inherent automation facilitated by AI eliminates the necessity for specialized expertise in SSM, democratizing access to size estimation capabilities across diverse organizational contexts.

Consequently, the integration of AI into SSM practices not only circumvents the barriers posed by manual effort and expertise but also catalyzes broader adoption and utilization of SSM tools within the software development landscape.

5.3. Future work

In delineating avenues for future research and development, several key areas emerge for further exploration and refinement based on the findings and implications of this thesis.

First and foremost, there exists a considerable opportunity to advance the automation capabilities of the models proposed herein, particularly regarding tailoring them to the unique requirements and contexts of individual organizations. By integrating customizable datasets into the model architectures, organizations can effectively adapt and configure the models to align with their specific software management frameworks and objectives.

Moreover, efforts to enhance the accessibility and usability of these models within existing software management tools, such as JIRA, hold significant promise for facilitating seamless integration and adoption within organizational workflows.

Furthermore, while this thesis primarily focused on the utilization of Large Language Model (LLM) and regression models for specific tasks within software size estimation, there remains ample scope for investigating the broader applicability of LLM

across diverse tasks. Notably, future research endeavors could explore the efficacy of LLM models in other tasks size estimation, other than just Object of Interest extraction. Given the demonstrated versatility and efficacy of LLM models in natural language processing tasks, it is conceivable that LLM-based approaches may obviate the need for distinct regression or summarization models, potentially streamlining the estimation process and enhancing predictive accuracy even more.

Additionally, future work should consider the possibility of repeating all tasks done in this thesis using LLM and compare the results with current findings. This comparative analysis could provide valuable insights into the strengths and limitations of LLM models in software size measurement, further informing best practices and guiding future research in this area.

Additionally, the dataset employed in this thesis, while qualitatively robust, was relatively small in scale. Future research endeavors could entail the expansion and enrichment of this dataset with a more extensive corpus of software projects, maintaining the same rigorous standards of quality and relevance. By augmenting the dataset with a broader diversity of projects, spanning various domains, sizes, and complexities, researchers can garner deeper insights into the generalizability and robustness of the proposed models across diverse software development contexts.

Lastly, extending the application of the proposed methodology to encompass other languages represents a promising avenue for future exploration. While this thesis primarily focused on software requirements written in English, the scalability and adaptability of the proposed models to accommodate multiple languages warrant further investigation. By extending the methodology to encompass a broader linguistic repertoire, researchers can broaden the applicability and utility of the proposed models, enhancing their relevance and impact within the broader software engineering community.

REFERENCES

1. Clancy, T. The Standish Group Report. *Chaos report* **1995**.
2. Jadhav, A.; Kaur, M.; Akter, F. Evolution of Software Development Effort and Cost Estimation Techniques: Five Decades Study Using Automated Text Mining Approach. *Math Probl Eng* **2022**, *2022*, 1–17.
3. Wilkie, F. G.; McChesney, I. R.; Morrow, P.; Tuxworth, C.; Lester, N. G. The Value of Software Sizing. *Inf Softw Technol* **2011**, *53* (11), 1236–1249.
4. Jørgensen, M.; Boehm, B.; Rifkin, S. Software Development Effort Estimation: Formal Models or Expert Judgment? *IEEE Softw* **2009**, *26* (2), 14–19.
5. Albrecht, A. J.; Gaffney, J. E. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE transactions on software engineering* **1983**, No. 6, 639–648.
6. Ünlü, H.; Hacaloglu, T.; Büber, F.; Berrak, K.; Leblebici, O.; Demirörs, O. Utilization of Three Software Size Measures for Effort Estimation in Agile World: A Case Study. In *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*; IEEE, 2022; pp 239–246.
7. Miranda, E. Improving Subjective Estimates Using Paired Comparisons. *IEEE Softw* **2001**, *18* (1), 87–91. https://doi.org/10.1109/52.903173.
8. Thangaratinam, S.; Redman, C. W. E. The Delphi Technique. *The obstetrician & gynaecologist* **2005**, *7* (2), 120–125.
9. Mahnič, V.; Hovelja, T. On Using Planning Poker for Estimating User Stories. *Journal of Systems and Software* **2012**, *85* (9), 2086–2095.
10. Zaw, T.; Hlaing, S. Z.; Lwin, M. M.; Ochimizu, K. The Measurement of Software Size Based on Generation Model Using COSMIC FSM. In *2019 23rd International Computer Science and Engineering Conference (ICSEC)*; 2019; pp 373–378. https://doi.org/10.1109/ICSEC47112.2019.8974688.
11. Commeyne, C.; Abran, A.; Djouab, R. Effort Estimation with Story Points and Cosmic Function Points-an Industry Case Study. *Software Measurement News* **2016**, *21* (1), 25–36.
12. Hacaloğlu, T. Event Points: A Software Size Measurement Model. **2021**.

13. Lavazza, L. On the Effort Required by Function Point Measurement Phases. *International Journal on Advances in Software* **2017**, *10* (1).

14. Lavazza, L. A.; Liu, G. An Empirical Evaluation of Simplified Function Point Measurement Processes. *International Journal on Advances in Software* **2013**, *6* (1–2), 1–13.

15. Hacaloğlu, T.; Ünlü, H.; Yıldız, A.; Demirörs, O. Software Size Measurement: Bridging Research and Practice. *IEEE Softw* **2024**, *41* (3), 49–58. https://doi.org/10.1109/MS.2024.3358079.

16. Minkiewicz, A. F. The Evolution of Software Size: A Search for Value. *Software Engineering Technology* **2009**, 23–26.

17. Demirors, O.; Gencel, C. Conceptual Association of Functional Size Measurement Methods. *IEEE Softw* **2009**, *26* (3), 71–78.

18. *Second Generation - Cosmic Sizing*. https://cosmic-sizing.org/cosmic-sizing/functional-size-measurement/second-generation/ (accessed 2024-04-09).

19. Bundschuh, M.; Dekkers, C. The IFPUG Function Point Counting Method. *The IT Measurement Compendium: Estimating and Benchmarking Success with Functional Size Measurement* **2008**, 323–363.

20. Engelhart, J.; Langbroek, P. *Function Point Analysis (FPA) for Software Enhancement*; Nesma, 2009.

21. Desharnais, J.; Buglione, L.; Kocaturk, B. Improving Agile Software Projects Planning Using the Cosmic Method. In *workshop on Managing Client Value Creation Process in Agile Projects (Torre Cane, Italy*; 2011.

22. Buglione, L.; Trudel, S. Guideline for Sizing Agile Projects with COSMIC. *Proceedings of the IWSM/MetriKon/Mensura* **2010**.

23. Laigner, R.; Kalinowski, M.; Diniz, P.; Barros, L.; Cassino, C.; Lemos, M.; Arruda, D.; Lifschitz, S.; Zhou, Y. From a Monolithic Big Data System to a Microservices Event-Driven Architecture. In *2020 46th Euromicro conference on software engineering and advanced applications (SEAA)*; IEEE, 2020; pp 213–220.

24. Michelson, B. M. Event-Driven Architecture Overview. *Patricia Seybold Group* **2006**, *2* (12), 10–1571.

25. Singhal, K.; Azizi, S.; Tu, T.; Mahdavi, S. S.; Wei, J.; Chung, H. W.; Scales, N.; Tanwani, A.; Cole-Lewis, H.; Pfohl, S. Large Language Models Encode Clinical Knowledge. *Nature* **2023**, *620* (7972), 172–180.

26. Shen, T.; Quach, V.; Barzilay, R.; Jaakkola, T. Blank Language Models. *arXiv preprint arXiv:2002.03079* **2020**.

27. Salazar, J.; Liang, D.; Nguyen, T. Q.; Kirchhoff, K. Masked Language Model Scoring. *arXiv preprint arXiv:1910.14659* **2019**.

28. Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv preprint arXiv:1406.1078* **2014**.

29. Xu, F. F.; Alon, U.; Neubig, G.; Hellendoorn, V. J. A Systematic Evaluation of Large Language Models of Code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*; 2022; pp 1–10.

30. Wu, T.; He, S.; Liu, J.; Sun, S.; Liu, K.; Han, Q.-L.; Tang, Y. A Brief Overview of ChatGPT: The History, Status Quo and Potential Future Development. *IEEE/CAA Journal of Automatica Sinica* **2023**, *10* (5), 1122–1136.

31. Paulus, R.; Xiong, C.; Socher, R. A Deep Reinforced Model for Abstractive Summarization. *arXiv preprint arXiv:1705.04304* **2017**.

32. Fabbri, A. R.; Kryściński, W.; McCann, B.; Xiong, C.; Socher, R.; Radev, D. Summeval: Re-Evaluating Summarization Evaluation. *Trans Assoc Comput Linguist* **2021**, *9*, 391–409.

33. Regolin, E. N.; Souza, G. A. de; Pozo, A. R. T.; Vergilio, S. R. Exploring Machine Learning Techniques for Software Size Estimation. In *23rd International Conference of the Chilean Computer Science Society, 2003. SCCC 2003. Proceedings.*; 2003; pp 130–136. https://doi.org/10.1109/SCCC.2003.1245453.

34. Afshari, M.; Gandomani, T. J. Enhancing Software Effort Estimation with Ant Colony Optimization Algorithm and Fuzzy-Neural Networks. In *2024 Third International Conference on Distributed Computing and High Performance Computing (DCHPC)*; 2024; pp 1–6. https://doi.org/10.1109/DCHPC60845.2024.10454085.

35. Mokri, F. D.; Molani, M. Software Cost Estimation Using Adaptive Neuro Fuzzy Inference System. *Int. J. Acad. Res. Comput. Eng* **2016**, *1* (1), 34–39.

36. Qin, M.; Shen, L.; Zhang, D.; Zhao, L. Deep Learning Model for Function Point Based Software Cost Estimation-an Industry Case Study. In *2019 International Conference on Intelligent Computing, Automation and Systems (ICICAS)*; IEEE, 2019; pp 768–772.

37. Mahmood, Y.; Kama, N.; Azmi, A.; Khan, A. S.; Ali, M. Software Effort Estimation Accuracy Prediction of Machine Learning Techniques: A Systematic Performance Evaluation. *Softw Pract Exp* **2022**, *52* (1), 39–65.

38. Baskeles, B.; Turhan, B.; Bener, A. Software Effort Estimation Using Machine Learning Methods. In *2007 22nd international symposium on computer and information sciences*; IEEE, 2007; pp 1–6.

39. Boehm, B.; Clark, B.; Horowitz, E.; Westland, C.; Madachy, R.; Selby, R. Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. *Annals of software engineering* **1995**, *1*, 57–94.

40. Ochodek, M.; Kopczyńska, S.; Staron, M. Deep Learning Model for End-to-End Approximation of COSMIC Functional Size Based on Use-Case Names. *Inf Softw Technol* **2020**, *123*, 106310.

41. Choetkiertikul, M.; Dam, H. K.; Tran, T.; Pham, T.; Ghose, A.; Menzies, T. A Deep Learning Model for Estimating Story Points. *IEEE Transactions on Software Engineering* **2018**, *45* (7), 637–656.

42. Fávero, E. M. D. B.; Casanova, D.; Pimentel, A. R. SE3M: A Model for Software Effort Estimation Using Pre-Trained Embedding Models. *Inf Softw Technol* **2022**, *147*, 106886.

43. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. Bert: Pre-Training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* **2018**.

44. Xiong, R.; Yang, Y.; He, D.; Zheng, K.; Zheng, S.; Xing, C.; Zhang, H.; Lan, Y.; Wang, L.; Liu, T. On Layer Normalization in the Transformer Architecture. In *International Conference on Machine Learning*; PMLR, 2020; pp 10524–10533.

45. Fávero, E. M. D. B.; Casanova, D. Bert_se: A Pre-Trained Language Representation Model for Software Engineering. *arXiv preprint arXiv:2112.00699* **2021**.

46. Sikka, G.; Kaur, A.; Uddin, M. Estimating Function Points: Using Machine Learning and Regression Models. In *2010 2nd International Conference on Education Technology and Computer*; IEEE, 2010; Vol. 3, pp V3-52.

47. Bilal, M.; Almazroi, A. A. Effectiveness of Fine-Tuned BERT Model in Classification of Helpful and Unhelpful Online Customer Reviews. *Electronic Commerce Research* **2023**, *23* (4), 2737–2757. https://doi.org/10.1007/s10660-022-09560-w.

48. *BertForSequenceClassification*. https://huggingface.co/docs/transformers/v4.40.0/en/model_doc/bert#transformers.BertForSequenceClassification (accessed 2024-04-21).

49. Loshchilov, I.; Hutter, F. Decoupled Weight Decay Regularization. *arXiv preprint arXiv:1711.05101* **2017**.

50. Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P. J. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of machine learning research* **2020**, *21* (140), 1–67.

51. *T5*. https://huggingface.co/docs/transformers/en/model_doc/t5 (accessed 2024-04-21).

52. Lin, C.-Y. Rouge: A Package for Automatic Evaluation of Summaries. In *Text summarization branches out*; 2004; pp 74–81.

53. Jiang, A. Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D. S.; Casas, D. de las; Bressand, F.; Lengyel, G.; Lample, G.; Saulnier, L. Mistral 7B. *arXiv preprint arXiv:2310.06825* **2023**.

54. Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; Chen, W. Lora: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685* **2021**.

55. *Mistral 7b Announcement*. https://mistral.ai/news/announcing-mistral-7b/ (accessed 2024-04-24).