Large Language Models in Software Engineering: A Critical Review and Future Research Directions

Ali Bektas

June, 2024

Abstract

The development of Transformer-based Large Language Models (LLMs) has led to a burgeoning interest in their applications within the Software Engineering (SE) domain, as evidenced by the surge in related publications. Existing surveys, such as those conducted by Zheng et al. [27] and Hou et al. [9], have documented the extensive utilization of LLMs in SE, exploring a variety of applications and methodologies. Fan et al. [5] describe the field of LLMs for SE as rapidly developing but still in an embryonic stage, highlighting the significant potential and the necessity for ongoing academic exploration. Their survey specifically aims to identify challenges and open problems, presenting a critical analysis of the early stages of this emerging field. In light of the rapid development of this field, which complicates a comprehensive review, this thesis is aimed at further exploring the field of LLM-based Software Engineering. It focuses on identifying new open problems and challenges and delineating potential future directions to provide additional guidance for the community in this dynamically evolving field.

To guide this investigation, the study is framed around two research questions:

- **RQ1:** How do open problems, applied methodologies, challenges, and limitations in the selected studies vary across different SE tasks?
- **RQ2:** Given the task-specific and methodological contexts, how can the identified open problems, challenges, and limitations be addressed, and what potential directions can be proposed?

This research, through a detailed review of the literature identified by Hou et al. [9], extracts, compares, and analyzes data to identify specific challenges and insights. It highlights data-related challenges in SE tasks related to planning and effort estimation, underlining the need for training datasets that adequately represent a diverse array of projects and programming languages. Additionally, it points out that while some research papers leverage the non-deterministic nature of LLMs in their evaluation strategies, many studies lack this consideration in their methodologies. In activities such as code generation, debugging, and bug fixing, several papers have recognized the trade-off between the advantages of language models like ChatGPT and the effort required to provide additional information for debugging purposes. There is ongoing research in this direction, indicating potential areas for extension.

The thesis employs a structured methodological framework comprising predefined attributes — Task Context, Evaluation Criteria, Methodological Approach, and Insights and Reflections. This framework facilitates a systematic comparative analysis aimed at elucidating the variety of open problems, challenges, and limitations in the practical application of LLMs across different SE tasks and methodologies. Through this analysis, the thesis contributes to a more nuanced understanding of these issues, highlighting significant areas where the field may develop and improve.

The aim of this research is to analyze and synthesize the broad spectrum of existing problems and limitations in the integration of LLMs into SE tasks, with a focus on deriving actionable insights and proposing potential research directions. These insights are intended to guide future studies, thereby improving the practical application of LLMs in SE and advancing the field with innovative solutions.

Zusammenfassung

Die Entwicklung von auf Transformern basierenden großen Sprachmodellen (LLMs) hat zu einem wachsenden Interesse an deren Anwendungen im Bereich des Software Engineerings (SE) geführt, wie der Anstieg der damit verbundenen Veröffentlichungen zeigt. Bestehende Untersuchungen, wie die von Zheng et al. [27] und Hou et al. [9] durchgeführten, haben die umfangreiche Nutzung von LLMs im SE dokumentiert und eine Vielzahl von Anwendungen und Methoden erforscht. Fan et al. [5] beschreiben das Feld der LLMs für SE als sich rasch entwickelnd, jedoch noch in einem frühen Stadium, und heben das bedeutende Potenzial und die Notwendigkeit einer fortlaufenden akademischen Erforschung hervor. Ihre Untersuchung zielt speziell darauf ab, Herausforderungen und offene Probleme zu identifizieren und präsentiert eine kritische Analyse der frühen Entwicklungsstadien dieses aufkommenden Bereichs. Angesichts der schnellen Entwicklung dieses Fachgebiets, die eine vollständige Überprüfung erschwert, zielt diese Arbeit darauf ab, das Feld des auf LLM basierenden Software Engineerings weiter zu erkunden. Sie konzentriert sich darauf, neue offene Probleme und Herausforderungen zu identifizieren und mögliche zukünftige Richtungen aufzuzeigen, um der Gemeinschaft zusätzliche Orientierung in diesem dynamischen Bereich zu bieten.

Um diese Untersuchung zu leiten, ist diese Arbeit um zwei Forschungsfragen strukturiert:

- **RQ1:** Wie variieren offene Probleme, angewandte Methoden, Herausforderungen und Einschränkungen in den ausgewählten Studien über verschiedene SE-Aufgaben?
- **RQ2:** Angesichts der aufgabenspezifischen und methodischen Kontexte, wie können die identifizierten offenen Probleme, Herausforderungen und Einschränkungen adressiert werden, und welche potenziellen Forschungsrichtungen können vorgeschlagen werden?

Diese Arbeit extrahiert, vergleicht und analysiert durch eine detaillierte Überprüfung der von Hou et al. [9] identifizierten Literatur Daten, um spezifische Herausforderungen und Erkenntnisse zu identifizieren. Sie beleuchtet datenbezogene Herausforderungen bei SE-Aufgaben im Zusammenhang mit Planung und Aufwandsschätzung und betont die Notwendigkeit von Trainingsdatensätzen, die eine vielfältige Auswahl an Projekten und Programmiersprachen adäquat repräsentieren. Darüber hinaus wird darauf hingewiesen, dass einige

Forschungspapiere die nicht-deterministische Natur von LLMs in ihren Evaluationsstrategien nutzen, viele Studien jedoch diesen Aspekt in ihren Methoden nicht berücksichtigen. In Aktivitäten wie Codegenerierung, Debugging und Fehlerbehebung haben mehrere Arbeiten den Kompromiss zwischen den Vorteilen von Sprachmodellen wie ChatGPT und dem Aufwand, der notwendig ist, um zusätzliche Informationen für Debugging-Zwecke bereitzustellen, erkannt. Es gibt laufende Forschungen in diese Richtung, die potenzielle Erweiterungsbereiche aufzeigen.

Diese Arbeit verwendet einen strukturierten methodischen Rahmen, der vordefinierte Attribute umfasst – Aufgaben-Kontext, Evaluationskriterien, methodischer Ansatz und Erkenntnisse und Reflexionen. Dieser Rahmen ermöglicht eine systematische vergleichende Analyse, die darauf abzielt, die Vielfalt der offenen Probleme, Herausforderungen und Einschränkungen bei der praktischen Anwendung von LLMs über verschiedene SE-Aufgaben und Methoden zu klären. Durch diese Analyse trägt die Arbeit zu einem differenzierteren Verständnis dieser Probleme bei und hebt bedeutende Bereiche hervor, in denen sich das Feld entwickeln und verbessern kann.

Das Ziel dieser Arbeit ist es, das breite Spektrum bestehender Probleme und Einschränkungen bei der Integration von LLMs in SE-Aufgaben zu analysieren und zu synthetisieren, um umsetzbare Erkenntnisse zu gewinnen und potenzielle Forschungsrichtungen vorzuschlagen. Diese Erkenntnisse sollen zukünftige Studien leiten und damit die praktische Anwendung von LLMs in SE verbessern und das Feld mit innovativen Lösungen vorantreiben.

1 Introduction

The introduction of transformer-based models has significantly influenced the field of natural language processing (NLP), primarily through the introduction of attention mechanisms. These mechanisms allow models to process input sequences in their entirety rather than sequentially, greatly enhancing their ability to understand context and relationships within text. This advancement has not only overcome limitations related to long-range dependencies and contextual consistency but has also catalyzed the adoption of Large Language Models (LLMs) across a variety of applications.

As the use of transformer-based LLMs expands into the domain of Software Engineering (SE), a surge in scholarly activity and practical applications has been noted. The number of studies involving LLMs in SE tasks has shown remarkable growth, as documented by Hou et al. [9], indicating a robust interest and perceived potential within this intersection.

However, alongside these advancements, a range of challenges and open problems persist that complicate their application in software engineering. These issues include the scalability of model sizes, the dependency on extensive and diverse datasets, and the challenge of code generation ambiguity, which often leads to syntactically correct but functionally inadequate outputs. Furthermore, the generalizability of LLMs across different SE tasks, their interpretability, and the ethical implications of their use remain significant concerns [9].

Fan et al. [5] further emphasizes the underexplored areas within this field, particularly in dynamic adaptation and the fine-tuning of LLMs specifically for SE tasks, suggesting a pressing need for tailored methodologies that leverage the unique aspects of software as opposed to general language processing.

This thesis aims to address these challenges by focusing on the identification and analysis of open problems and challenges in the utilization of LLMs for SE tasks. By exploring how these issues vary across different SE tasks and methodologies applied in the selected studies, this research seeks to provide a structured overview of the field and propose potential solutions that could guide future research efforts. This approach is especially pertinent given the rapid evolution of LLMs and their increasingly complex integration into SE tasks.

The structure of the thesis is as follows:

1. **State of the Art:** This section reviews essential concepts and current trends in both software engineering and LLM applications.

2. **Methodology:** Here, the systematic approach adopted for literature review and data extraction is described, which is crucial for addressing the research questions.

3. **Results:** Key findings are discussed in this section, highlighting how LLMs are applied and evaluated within SE.

4. **Conclusions:** The thesis concludes with a synthesis of insights that could influence future research and practice in the field.

1.1 Motivation

The integration of LLMs into SE reflects a rapidly evolving landscape where theoretical concepts meet practical application. My involvement with this field stems from direct experiences in university projects and student employment, where the nuanced challenge of evaluating machine learning models first became evident to me. Particularly revealing was the realization that even for seemingly straightforward classification tasks, like those involving imbalanced data, choosing the right evaluation metrics and crafting a sound evaluation strategy to accurately assess the solution's quality for the specified task is a important and complex task.

Building upon the insights from [9], it is recognized that while LLMs are increasingly deployed across various SE tasks—regression, classification, recommendation, and generation—the conventional metrics like Accuracy, Recall, or F1-score often fall short of providing a comprehensive assessment of a model's performance. These metrics might not fully capture the nuanced effects and impacts LLMs have within specific SE tasks, potentially overlooking aspects like interpretability, robustness, or error sensitivity. This observation aligns with broader findings from [3] and [7], which emphasize the need for evolving evaluation methodologies alongside LLM advancements. They call for more inclusive, adaptive, and enhancement-oriented evaluation frameworks that not only benchmark performance but also identify and address areas for improvement, ensuring LLMs' alignment with ethical standards and practical utility in SE.

1.2 Related Work and Research Corpora

This thesis builds upon a comprehensive literature review detailed in the foundational paper by [9]. This pivotal review thoroughly explored the broader application of LLMs in SE, examining the types of LLMs utilized, data handling methods, optimization and evaluation techniques employed, and the specific tasks these models have been applied to. By delving deeper into the methodology and findings of this foundational literature review, this subsection provides crucial insights into the landscape from which this research sources its corpus.

The identification process in the basis paper [9] followed a carefully structured approach, adhering to the SLR methodologies established by Kitchenham et al. [11] [10].

The five-step seach strategy started by selecting six prominent SE venues—ICSE, ESEC/FSE, ASE, ISSTA, TOSEM, and TSE—for manual searches and extended their scope to include seven databases: IEEE Xplore, ACM Digital Library, ScienceDirect, Web of Science, Springer, arXiv, and DBLP for automated searches. A quasi-gold standard was developed by screening papers from manual search based on specific inclusion and exclusion criteria, followed by the creation of a search string informed by domain expertise for the automated search. The process concluded with a snowballing search, integrating findings from both manual and automated searches to ensure a comprehensive collection of pertinent studies for subsequent analysis.

The manual and automated search resulted in 164,366 papers which was reduced to a focused set of 218 foundational studies through a series of structured steps applied in the study selection process. After an initial automated filter based on paper length trimmed the count to 63,404, a thorough title, abstract, and keyword review further reduced the pool to 4,341. At this juncture, the researchers identified the publication venues to discern the source quality, which, along with subsequent deduplication and in-depth full-text reviews considering relevance and rigor, distilled the collection to 548. Additional exclusion of papers was done by applying quality assessments and manually scoring papers based on content relevance and publication type/ venue quality finally narrowed it down to 218 papers. A subsequent snowballing search, executed on these selected studies, yielded an additional 11 papers, thereby establishing a final set of 229 papers after deduplication and verification.

The distribution of the 229 relevant papers identified demonstrates that 38% were published in peer-reviewed venues like ICSE, TSE, ICSME, and SANER, while 62% appeared on arXiv, reflecting the field's swift evolution and the prepeer review status of much Large Language Models for Software Engineering (LLM4SE) research. A temporal analysis shows a significant surge in publications from 2020 to 2023, highlighting escalating interest and research activity in LLM4SE.

1.3 Research Questions

In this work, the focus is on *evaluating methodologies* applied to *solutions that incorporate Large Language Models (LLMs)* for SE tasks. Particular emphasis is placed on studies presenting and assessing LLM-based solutions for specific SE challenges, as these represent the majority of the examined works. Additionally, the analysis includes comparative studies contrasting LLMs with traditional models and investigations that concentrate on the unique properties of LLMs within SE contexts. The research questions (RQ1 and RQ2) aim to identify the diversity and specificity of evaluation methods across different SE tasks and methodologies and to explore how these methods can be optimized to address the unique requirements and challenges, thereby enhancing understanding of the effectiveness and practical applicability of LLMs in SE.

- **RQ1:** How do evaluation methods vary across different software engineering tasks and methodologies in the selected studies?
- **RQ2:** Given their task and methodology-specific contexts, how can the identified evaluation methods be enhanced to address their respective challenges?

In the research questions of this master thesis, "Evaluation Methods" specifically refers to the techniques and approaches used for assessing the overall effectiveness and efficiency of the complete solution proposed for solving the SE

Task, and not to the methods used for evaluating the Language Model (LLM) itself.

2 State of the Art

2.1 Large Language Models

Natural Language Processing (NLP) is a domain within artificial intelligence that focuses on the interaction between computers and humans through natural language. The objective is to enable computers to understand, interpret, and generate human language to support tasks like translation, summarization, information retrieval, sentiment analysis, and more. Within this field, a language model is a computational tool that predicts the likelihood of a sequence of words or phrases, capturing the essence of language syntax and semantics based on vast amounts of training text data.

The evolution of language models has unfolded through distinct phases, each marked by significant technological milestones. In the 1950s and 1960s, the field was dominated by rule-based models, relying on manually crafted linguistic rules to process language. The 1980s and especially the 1990s witnessed a pivotal shift to statistical models that used large text corpora to infer language patterns, enhancing the adaptability and scalability of language processing.

The late 1990s and early 2000s introduced neural network-based models, notably Recurrent Neural Networks (RNNs) [12], which excelled in sequential data processing, laying the groundwork for sophisticated text understanding and generation. The evolution continued into the mid-2010s with the emergence of pre-trained models, paving the way for the development of Large Language Models (LLMs). A transformative moment occurred around 2017 with the introduction of transformer models in the ”Attention Is All You Need” paper [23], which for the first time relied solely on self-attention mechanisms, eliminating the need for RNNs or convolutional layers. This marked a significant paradigm shift from earlier models that combined self-attention with other architectures, to a new era where self-attention alone could drive deep, contextually aware language understanding, propelling the field of NLP into new frontiers of capability and flexibility.

Sequential models, particularly Recurrent Neural Networks (RNNs) and their advanced variants like Long Short-Term Memory (LSTM) networks, have been pivotal in processing language data due to their inherent design to handle sequential information. They process text one token at a time, maintaining a hidden state that theoretically encapsulates the information from all previously seen tokens, thus creating a sense of memory. However, these models often struggle with long-range dependencies due to issues like vanishing or exploding gradients.

A pivotal feature of the Transformer architecture is its self-attention mechanism, which significantly enhances the model’s ability to address the limitations inherent in sequential processing. Unlike sequential models that process data in order, self-attention allows the model to weigh and relate any two tokens in the input, regardless of their positions. This mechanism computes the representation of each token by considering how it relates to every other token in the sequence, enabling the model to capture dependencies without being constrained by the distance between tokens. As a result, self-attention provides a more flexible and context-aware way to encode the semantics of the input sequence, enhancing the model's ability to understand and generate language with greater nuance and coherence.

The self-attention mechanism within large language models represents a paradigm shift in how models ascertain and encode the relationships between tokens in a sequence. Specifically, self-attention calculates alignment scores, typically through a scaled dot-product attention process (Fig. 2 left), where each token's representation is dynamically influenced by computing how much focus it should place on every other token in the sequence. These scores determine the extent to which each token should attend to every other token across the sequence, enabling the weighted aggregation of token representations that effectively contextualizes each word within its surrounding linguistic environment.

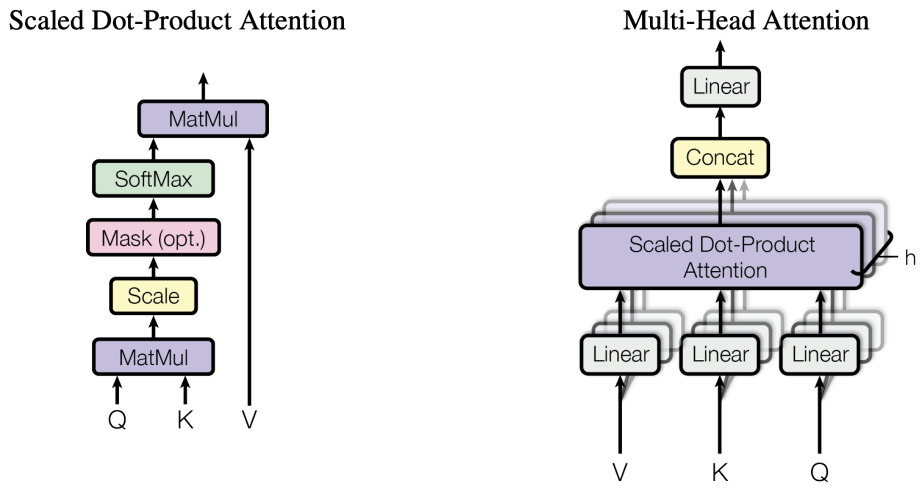

Figure 1: Illustration of the Scaled Dot-Product Attention and Multi-Head Attention mechanisms, adapted from [23]. The left part of the image demonstrates the computation within a single attention head, focusing on the scaled dot-product attention process, while the right part delineates the aggregation of multiple such heads in the multi-head attention framework.

Diving deeper into the multi-head attention feature (Fig. 1 right), we understand that it subdivides the attention mechanism into multiple "heads," allowing the model to concurrently explore different dimensions of the data. Each head can potentially capture distinct types of relationships, such as varying syntactic or semantic connections, by computing separate sets of alignment scores. After this parallel processing, the diverse outputs from all heads are concatenated and linearly transformed, yielding a rich, integrated representation that encapsulates various linguistic nuances and facets, thereby enhancing the model's linguistic comprehension.

The attention mechanism inherent in transformer models not only enhances their performance but also offers a level of interpretability that is crucial for understanding their inner workings. Recent works [24] [26] have delved into exploring these models' attention mechanisms through visualization techniques, shedding light on how these models process and prioritize different parts of the input data.

In Figure 2, we observe a visualization that elucidates the functionality of two attention heads processing the word 'its' within an input sequence. It becomes clear from this visualization that these attention heads are adept at anaphora resolution, showcasing their ability to link referential expressions with their antecedents effectively. Additionally, Figure 3 provides insights to the scaled dot-product attention mechanism within a single attention head, focused on the token '.'. This visualization intuitively demonstrates how the attention values evolve, emphasizing a pattern where attention diminishes as the distance from the sentence-ending token increases.

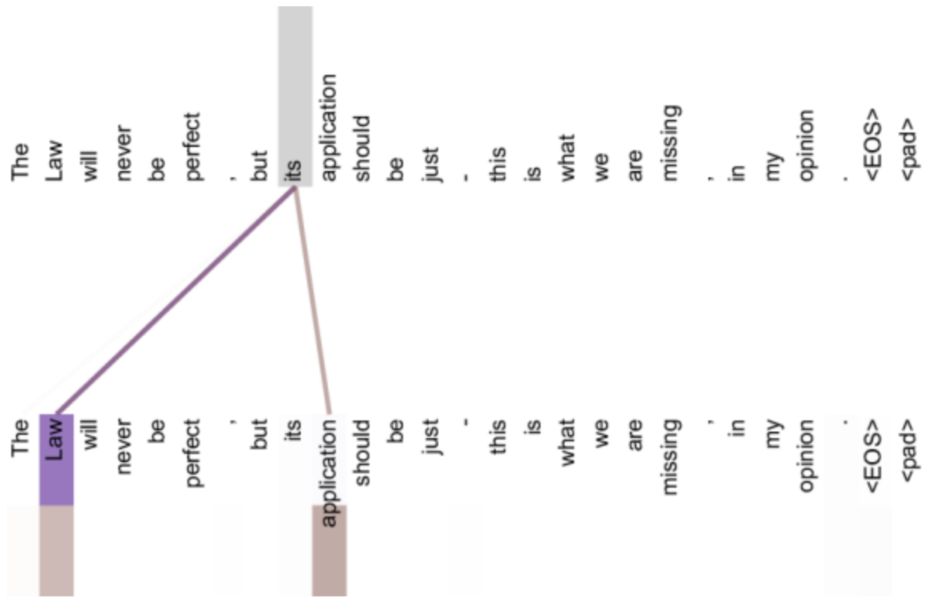

Figure 2: Illustration , adapted from [23]. Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word 'Law' .

Typically, a model might employ 8 or 16 attention heads, with the choice of

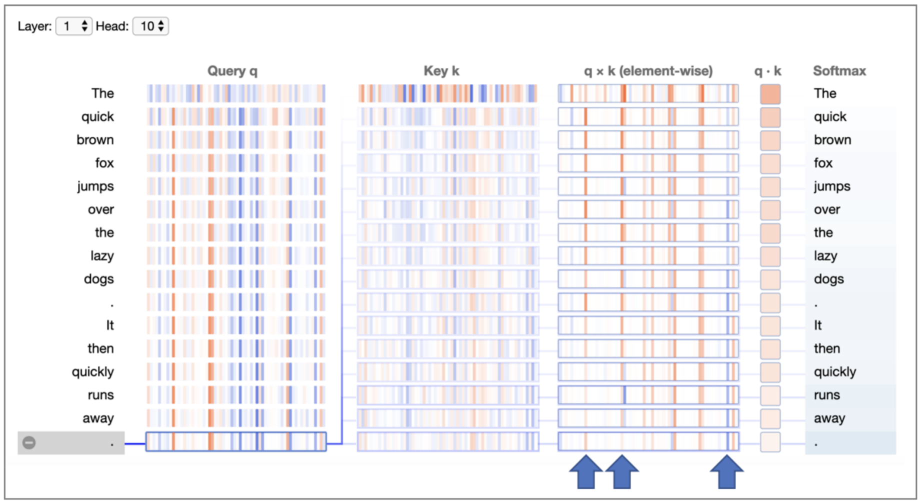

Figure 3: Illustration adapted from [24]. Neuron view of GPT-2 for layer 1, head 10 with last token selected. Positive and negative values are colored blue and orange, respectively, with color saturation based on magnitude of the value. As with the attention-head view, connecting lines are weighted based on attention between the words. Blue arrows mark positions in the element-wise products where values decrease with increasing distance from the source token (becoming darker orange or lighter blue).

this parameter influencing the granularity and scope of relationships the model can capture, thereby affecting its performance and interpretability [24] [26]. This feature contrasts with the sequential nature of RNNs, where information processing is inherently linear and time-dependent. Self-attention's parallelizable structure not only mitigates these limitations but also facilitates significantly faster training times and improved handling of longer sequences.

In the context of NLP tasks, these architectural innovations have proven instrumental across a range of applications, from translation to content generation, underpinning the versatile and powerful capabilities of large language models. However, it is also crucial to acknowledge inherent challenges, such as increased computational demands and potential difficulties in model interpretability, particularly in dissecting the specific contributions of individual attention heads or understanding the model's decision-making process in detail. Acknowledging these limitations alongside the benefits provides a comprehensive understanding of the self-attention and multi-head mechanisms within the larger narrative of language model evolution.

Following the introduction of transformer models, a notable development in the field of NLP is the application of Reinforcement Learning from Human Feedback (RLHF) to enhance the performance of large language models. The study by Ziegler et al. [29] exemplifies this approach, applying RLHF to refine pre-trained language models for specific tasks like text continuation and summarization, informed by human feedback.

In their research, Ziegler et al. [29] demonstrate how RLHF can be used to adjust language models based on human evaluations, aiming to produce outputs that are more aligned with human judgments of quality and relevance. This process involves training a reward model from human preferences, which is then used to guide the fine-tuning of language models, enhancing their ability to generate text that resonates more aligned to with human readers preference. The integration of RLHF represents a methodological advancement in the field, enabling language models to evolve based on direct human input, thus improving their applicability and effectiveness in real-world tasks. This development underscores the shift toward more interactive and adaptive systems in NLP, where human feedback plays a vital role in refining AI outputs. Through such advancements, language models continue to evolve, offering more sophisticated tools for a variety of NLP applications, illustrating a commitment to aligning AI systems more closely with human expectations and standards, and showcasing the potential for these models to become more nuanced and context-aware in their language generation capabilities.

2.2 Large Language Models for Software Engineering Tasks

Software Engineering (SE) emerged as a distinct discipline during a pivotal period in the mid-20th century, marked by the rapid expansion of computing technology. As hardware capabilities advanced, significant disparities in software development methodologies became evident, leading to what is known as the "software crisis." This crisis highlighted difficulties in developing reliable, maintainable, and efficient software within acceptable timeframes and budgets. In response, the seminal NATO conferences of the late 1960s were convened. These conferences played a crucial role in defining SE as a critical field aimed at addressing the complexities of software creation. Due to its abstract nature, software allows for substantial customization and wide applicability but also introduces significant complexities that challenge the automation of development processes. This dynamic requires a careful balance between specialized solutions and broadly applicable approaches in software design, underscoring the nuanced and continuously evolving nature of the field. The discipline of SE continues to adapt, applying established principles to new challenges across a diverse range of application domains [16, 2].

SE orchestrates all aspects of software production by applying engineering principles to develop functional, dependable, and efficient software within financial and temporal constraints. Unlike computer science, which explores theoretical aspects, or systems engineering, which addresses broader systems issues, SE focuses specifically on software solutions. It incorporates the Software Development Lifecycle (SDLC), a systematic process that includes phases such as planning, design, implementation, testing, deployment, and maintenance. This lifecycle helps manage and streamline the creation and maintenance of software, accommodating a diverse array of software types and usage scenarios. While no single methodology fits every project, leading practices vary from stand-alone applications to embedded systems. However, principles like effective process management, dependability, clear requirement gathering, and efficient resource use remain universal. These principles ensure the delivery of reliable software that meets various user and market demands, amidst ongoing technological and business challenges [20].

Building on the advancements in artificial intelligence research within SE, such as the optimization techniques employed in Search-Based Software Engineering (SBSE) which also involved natural language processing [21, 13], an intriguing new focus has emerged: the application of Large Language Models (LLMs) in SE . This area explores the potential of LLMs to manage tasks that involve understanding and generating human language, now directed toward the complexities of software development processes.

Since 2019, LLMs have garnered significant attention for their applications in SE, reflecting a broader trend across computer science disciplines. According to Fan et al. (2023), this focus has led to the recognition of LLM-based SE as an emerging subdiscipline [5]. The authors highlight the substantial growth in publications related to LLMs in SE, demonstrating the community's rapid adoption of this technology.

Supporting this observation, Hou et al. [9] document an exponential increase in publications that integrate LLMs with SE tasks—from 7 papers in 2020 to an impressive 160 in just the first half of 2023 [9]. This surge in research activity has spurred the development of the term Large Language Models for Software Engineering (LLM4SE), defining this vibrant area of study. According to Fan et al. [5], since 2022, more than 10% of all LLM-related publications have focused specifically on SE applications, underscoring the significant interest in this field [5].

Additionally, Fan et al. [5] also note that while LLMs are extensively explored in the domain of software development, certain subdomains such as Requirements Engineering and Design, and Refactoring are notably under-represented. These areas, which rely heavily upon linguistic forms of analysis and the recognition and prediction of patterns, are identified as surprisingly ripe for consideration, presenting an opportunity for future research and application of LLMs [5].

From the detailed analysis by Hou et al. [9], it is evident that LLMs are currently being utilized across a spectrum of SE tasks. They are categorized based on their architecture into encoder-only, encoder-decoder, and decoder-only models, with the latter being the most utilized in SE, particularly for tasks involving code generation and program repair. The popularity of decoder-only models in SE underscores their effectiveness in tasks requiring deep syntactic and semantic understanding of code. This effectiveness is attributed to the autoregressive nature of decoder-only models, which allows them to generate sequences of text by predicting the next token based on the context of the preceding tokens. This capability is crucial for code generation and repair tasks, where understanding the sequential and contextual relationships within the code is essential for producing accurate and coherent outputs.

Data handling is another area of focus; the predominant use of open-source datasets, which constitute approximately 59.35% of cases, shows a reliance on widely accessible resources for training these models. Data types within these datasets are primarily code-based and text-based, aligning with the strengths of LLMs in handling complex structured data, which is crucial for SE tasks.

Moreover, Hou et al. [9] outlines specific optimization techniques like fine-tuning and the Adam optimizer, which are commonly used to enhance the performance of LLMs in SE. The use of prompt engineering, particularly in data-scarce scenarios, further enhances these models' adaptability and effectiveness, allowing them to perform optimally across various SE tasks.

3 Method

3.1 Approach

In addressing the research questions, this research adopts a structured and iterative approach to examine the applied methodologies, evaluation strategies and how the results are interpreted in the corpus of research papers utilizing Large Language Models (LLMs) for solving Software Engineering (SE) tasks. This approach is designed to extract, analyze, and refine data from the corpus identified in the literature review by [9]. The intent is to extract the variability of applied methodologies, evaluation practices challenges and limitations across different SE tasks to forge targeted recommendations for their enhancement. The following steps of the approach, visualized in Figure 4, are explained in more detail:

Step 1: Paper and Attribute Selection

- Define a comprehensive set of attributes for extraction from the selected papers, pivotal for elucidating the interplay between SE tasks, methodologies, evaluation methods and practical usability of the proposed solution. These attributes should be capable of:
  - Enhance understanding of SE task characteristics for practical usability:
    - Identify all SE Tasks the research paper covers.
    - Highlight critical characteristics for practical usability. For instance, in Code Search, it may be important to determine if the correct match needs to be within the top 10 recommendations, rather than strictly at the first rank.
    - Consider task-specific needs. For example for vulnerability detection, weigh the importance of avoiding False Positive Predictions (False Alarms) against covering all possible threats, even at the cost of False Alarms.
  - Understand the applied methodology, evaluation strategy and its impacts for the practical usability of the solution:

    - Data sampling: From which sources had data been sourced. Assess real-world task representativeness of data. Understanding if the data helps in interpreting the results and accessing the appropriateness of the evaluation strategy.
    - Consider training strategy effects: For example, the choice between fine-tuning a smaller model with task-specific data versus using a larger model trained on a broader corpus could mean different genralizability capability of a model. If generalizability is important for the SE Task then this needs consideration in the evaluation strategy and in the interpretation of the results.
    - Are the benchmarks used realistic to real world scenario? For example in addressing code generation tasks, is different difficulty and complexity levels of code considered? Is different programming languages considered?
  - Offer insights into outcomes, limitations, and future directions:
    - Recognize study limitations and challenges.
    - Summarize results.
    - How does the interpretation of the results vary across studies? Do different studies interpret the same evaluation outcome for similar task differently? Are there patterns?

- Select an initial batch of 10 papers, representing the spectrum of SE activities and ML tasks detailed in [9].

Step 2: In-depth Review and Data Extraction

- Systematically review the chosen papers to extract the designated attributes, especially focusing on elucidating the evaluation methods and methodologies within varied SE tasks with respect to the practical usability.

- Cross-validate the extracted data with the supplementary materials from the replication package [1] in [9] to ensure completeness.

- Continuously assess and refine the attribute set to encapsulate a holistic view of the evaluation landscape, adapting to emergent insights.

Step 3: Iterative Refinement and Expansion

- Adjust the dataset iterative, incorporating new attributes identified during the review to maintain a robust and coherent analytical framework across all papers.

Step 4: Synthesis and Pattern Identification

[1] `https://docs.google.com/spreadsheets/d/1iomMvoDL2znNDQ_J4aGnqb3BhZpEMlfz/edit#gid=1471652439`

- Synthesize the extracted data to delineate prevalent and outlier evaluation strategies, applied methodologies, challenges and limitations discerning patterns and variances linked to different SE tasks and methodologies, aligning this analysis with **RQ1**.

Step 5: Comprehensive Review

- Extend the systematic application of the preceding steps across the entire suite of 230 papers cataloged by [9], ensuring a thorough and representative analysis.

Step 6: Analysis and Recommendation Formulation

- Interpret the identified patterns and insights, correlating them with the contextual demands of various SE tasks and methodologies to derive targeted improvements, addressing **RQ2**. This step aims to refine the methods applied in the selected studies and proposes directions for future investigative pursuits.

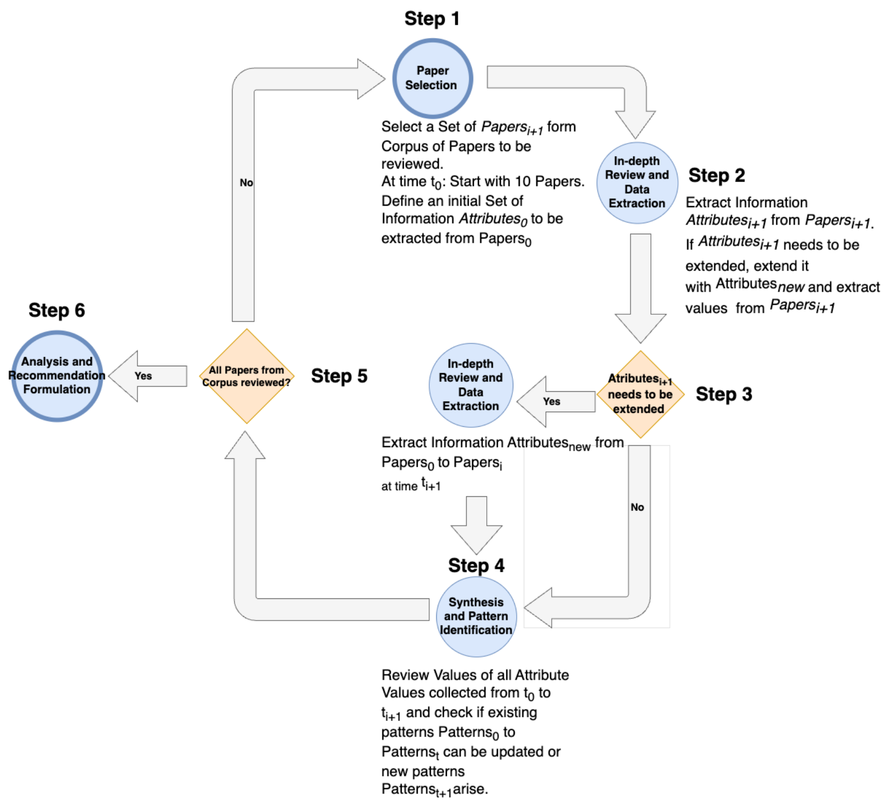
Step 1
Paper Selection
Select a Set of $Papers_{i+1}$ form Corpus of Papers to be reviewed.
At time t_0: Start with 10 Papers.
Define an initial Set of Information $Attributes_0$ to be extracted from $Papers_0$
Step 2
In-depth Review and Data Extraction
Extract Information $Attributes_{i+1}$ from $Papers_{i+1}$.
If $Attributes_{i+1}$ needs to be extended, extend it with $Attributes_{new}$ and extract values from $Papers_{i+1}$
Step 3
$Atributes_{i+1}$ needs to be extended
Yes
No
In-depth Review and Data Extraction
Extract Information $Attributes_{new}$ from $Papers_0$ to $Papers_i$ at time t_{i+1}
Step 4
Synthesis and Pattern Identification
Review Values of all Attribute Values collected from t_0 to t_{i+1} and check if existing patterns $Patterns_0$ to $Patterns_t$ can be updated or new patterns $Patterns_{t+1}$ arise.
Step 5
All Papers from Corpus reviewed?
No
Yes
Step 6
Analysis and Recommendation Formulation

Figure 4: Methodology flowchart outlining the steps for analyzing and enhancing LLM evaluation in SE tasks.

3.2 Exploration of Replication Package from Literature Review

In this section, we delineate the review process applied to the contents of the replication package[2] of [9] as of February 2024. Our objective is to scrutinize the available data within the package to identify the papers that constitute the literature review from [9] and to discern which information pertaining to these reviewed papers is structured and accessible. This careful examination aims to uncover potential facilitators for our analysis, ensuring we leverage any structured information to augment our understanding and streamline our research process.

The package comprises five Excel sheets: *QAC*, *RQ1*, *RQ2*, *RQ3*, and *RQ4*. A preliminary review of the column names and entries revealed that these labels stand for *Quality Assurance Criteria* and *Research Questions 1-4*, each sheet

[2]`https://docs.google.com/spreadsheets/d/1iomMvoDL2znNDQ_J4aGnqb3BhZpEMlfz/edit#gid=1471652439`

containing relevant data.

Each sheet adheres to a uniform structure, presenting identifier information for the reviewed papers, including *Title*, *URL*, *Year*, *Venue*, and *Abstract*, succeeded by sheet-specific values.

QAC: In their inclusion criteria, the authors of [9] evaluated the quality of each review paper, as detailed in the *Study Selection* section of [9], attributing a score to 10 Quality Assurance Criteria (QAC). The *QAC* sheet encapsulates the scores for these 10 criteria.

The Quality Assurance Criteria collectively assess the relevance, methodology, clarity, and impact of the studies within the scope of software engineering tasks, specifically examining their use of large language models (LLMs), publication venue prestige, motivation clarity, technique description, experimental detail, findings confirmation, discussion on contributions and limitations, and overall contribution to the academic or industrial community.

The sheets *RQ1* to *RQ4* provide structured documentation of the data extracted by [9] to address their respective Research Questions.

RQ1: This sheet includes information on the *utilized large language models (LLMs)* and their respective *Transformer Types* for each study reviewed. Columns are designated with names of *LLM Models*, covering all within the GPT and BERT series, and feature three additional columns to identify Transformer Types: Decoder only, Encoder only, or Decoder-Encoder. The inclusion of a model in a study is indicated by a column value of '1' or detailed further with specific information, such as the model's number of parameters.

RQ2: This sheet organizes dataset-specific details, identifying the *source* of the dataset as *open-source*, *collected*, *constructed*, or an *industry dataset*. It also categorizes the *type* of data utilized in the studies, distinguishing among *text-based*, *code-based*, *graph-based*, *software-repository-based*, or *combined* datasets. The *input types* for LLMs are outlined as *token-based*, *pixel-based*, *tree/graph-based*, or *hybrid* input types, with a '1' denoting usage or additional details provided. A column on *data preprocessing* lists specific preprocessing steps applied.

RQ3: This sheet provides information on the methods and metrics employed for optimizing and evaluating the large language models (LLMs) as described in the research studies. It includes a column for *Parameter Optimization Algorithms*, indicating the techniques utilized, such as *hyperparameter tuning*, *fine-tuning*, or a combination of both. It also details the specific optimizers used, for instance, *Adam*, *AdamW*, or *ZeRO*. For selected studies, it documents strategies to prevent overfitting within the *combat overfitting* column, mentioning approaches like *early stopping* and *data augmentation*. The sheet categorizes the machine learning problem in the *problem type* column into *classification*, *regression*, *generation*, or *recommendation*. Additionally, a *Metrics* column records the metrics applied by the research papers to assess their findings.

RQ4: This sheet organizes the research papers according to SE-Activities and SE-Tasks. It features a dedicated column for each SE-Activity: *Software Requirements*, *Software Design*, *Software Development*, *Software Testing*, *Software Management*, and *Software Maintenance*. Within these columns, the corresponding *SE-Task(s)* addressed by the research are listed as values. It is observed that some papers could not be associated with a specific SE-Task, marked by the value *other*. Additionally, there are instances where certain studies are not aligned with any SE-Activity.

Upon reviewing the list of paper titles retrieved, the paper [28] titled *Large language models are human-level prompt engineers* stood out as it seemed unrelated to the application of LLMs in addressing a Software Engineering task. This paper is listed in Table 6 of [9] among papers using *Text-based datasets* and in Table 9 among those using *Text as Tokens* as input for the LLMs.

However, upon closer examination of the content, it was found to violate the inclusion criterion 2) in Table 3, which states, *"The paper claims that the study involves an SE task,"* of the survey paper [9]. It appears this paper uses the SE task of *Program Synthesis* to address the topic of automatic prompt engineering for LLMs. For this reason, this paper is excluded from the investigation in this work.

3.3 Review Process of the Survey Papers

Following the exploration of the replication package of the basic literature review [9] and the identification of the papers belonging to the literature review, the first iteration of *Step 1* of the Approach described in Chapter 3.1 was conducted.

A subset of 11 papers covering a similar distribution of **SE-Activity** and **ML-Task** as the proportion in [9] was selected. A first set of attributes to be extracted from the papers under review was specified and refined during the review of the subset of papers.

All papers were reviewed by a single reader, the author of this thesis, and the following sections were reviewed as described below:

- **Title, Abstract, and Introduction:** Provided the most relevant information to understand the SE-Task and the proposed solutions, with the Abstract offering an initial glance at the applied methods, results, and conclusions.
- **Method or Approach Sections:** Initially glanced over to gain an understanding of the methods and approaches employed, with thorough review conducted if necessary to fully capture the details.
- **Dataset or Sampling Strategy Sections:** Initially reviewed to understand how data was handled and utilized, with further detailed examination performed as required.
- **Evaluation Strategy Sections:** Thoroughly reviewed as needed to facilitate a comprehensive understanding of the evaluation methodologies used.
- **Tables, Figures, and Diagrams:** Reviewed to provide clear insights into the Method, Evaluation Strategy, and Results, offering a concise representation of the research frameworks and findings.

- **Results, Conclusions, and Limitations Sections:** Reviewed to understand the outcomes, their interpretation, and any noted challenges or limitations. Insights gained included:
  - The outcomes of the studies.
  - The authors' interpretations of these outcomes.
  - Challenges and limitations encountered during the research.
- **Data Summarization:** Information from each paper was summarized for specific attributes to ensure good comparability between the papers:
  - **Categorical Data:** Such as metrics used, was stored for better comparability.
  - **Descriptive Data:** Where a text summary was required (e.g. results or the authors interpretations), a concise summary was created and stored.

By systematically reviewing these sections of the papers and summarizing the relevant information, a comprehensive understanding of each study's objectives, methodologies, evaluation strategies, and findings was obtained. This process ensured a robust analysis aligned with the structured approach outlined in Chapter 3.1.

Following the review, an analysis was conducted to identify how the extracted attributes could be grouped and to examine the term frequencies of their values. This analysis aimed to reveal patterns, trends, and outstanding approaches within the data, enhancing the understanding of the scope and impact of LLM applications in SE.

Analysis of Extracted Attributes:

- **Task Context:** The *SE Task Category* was chosen to delineate the range of Software Engineering tasks explored within the studies. This categorization aids in identifying and grouping papers that address similar SE challenges, facilitating an understanding of the common and divergent problem domains that LLMs are applied to in SE research.
- **Evaluation Criteria:** Comprising *Metrics* and *Benchmarks*, this grouping was selected to outline the standards and reference frameworks employed in evaluating LLM performance within SE tasks. It serves to standardize the varied evaluation approaches, enabling a comparison across studies based on their assessment methodologies.
- **Methodological Approach:** Incorporating *Training Approach*, *Sampling Strategy*, and *Evaluation Strategy*, this classification elucidates the procedural nuances of the studies. The rationale for selecting these attributes is to capture the diversity in methodological approaches, including model training, data sampling, and the blend of quantitative and qualitative evaluation strategies. This facilitates an analytical perspective on the methodological rigor and innovation within the LLM applications in SE.

- **Insights and Reflections:** Encompassing *Challenges and Limitations*, *Results and Findings*, and *Authors' Interpretation*, this assembly was delineated to provide an overarching view of the research outcomes. It aims to crystallize the significant discoveries, the obstacles encountered, and the authors' own reflections on their findings. By examining these attributes, the analysis seeks to underscore the practical implications and future research directions emerging from the current body of work.

After extracting structured information from the reviewed papers, term frequencies for the attributes have been summarized to identify trends, outstanding approaches, and patterns.

For the attribute *SE Task Category*, research papers focusing on *Code Generation* lead with 5 occurrences, followed by *Requirements Classification*, *Software Effort Estimation*, *Vulnerability Detection*, and *Debugging*, each with 2 occurrences. A paper can appear in multiple SE Task categories, for example, in *Code Generation* for a *Debugging* task.

The most common metrics used are *Recall* (5 occurrences), *F1-score* and *Accuracy* (referred to as Success Rate, both 4 occurrences), and *Precision* (3 occurrences). These metrics are widely used for quantitative and comparative evaluations in classification tasks with categorical results. *AUC* and *FPR* appeared in 2 studies each, while other metrics like statistical significance tests, MAE, cosine similarity, SHAP (Explainability), NDCG, and various similarities each have 1 occurrence.

For training approaches, the majority of the papers utilize a *Pre-trained foundational model* (9 occurrences). Four papers use foundational models via API access, such as ChatGPT. Two works adopt the *Fine-tuning* approach with domain-specific data. Other approaches, including transfer learning, prompt learning, self-learning, and combining LLMs with explainable AI or traditional ML models, each occur once.

As the attributes *Challenges and Limitations*, *Results and Findings*, *Authors' Interpretation*, and *Sampling Strategy* involve capturing more summarized descriptions than categorical values, the term frequencies are shown in separate word clouds. These word clouds emphasize emerging words within each attribute, highlighting key trends and insights visually depicted in Figure 5.

With this approach, it is intended to reveal insights that are not immediately visible in individual papers but become evident through the aggregated analysis of data from multiple studies. The systematic categorization and attribute selection process is designed to be iterative, allowing for the identification of thematic clusters and methodological trends as the review progresses. Importantly, this method is aimed at recognizing common challenges and delineating limitations and weaknesses in the evaluation and interpretation methodologies across the reviewed studies. Consequently, this comparative analysis is designed not only to deepen the understanding of prevalent issues in the field but also to identify areas requiring methodological refinement and further research in the application of LLMs to Software Engineering tasks.

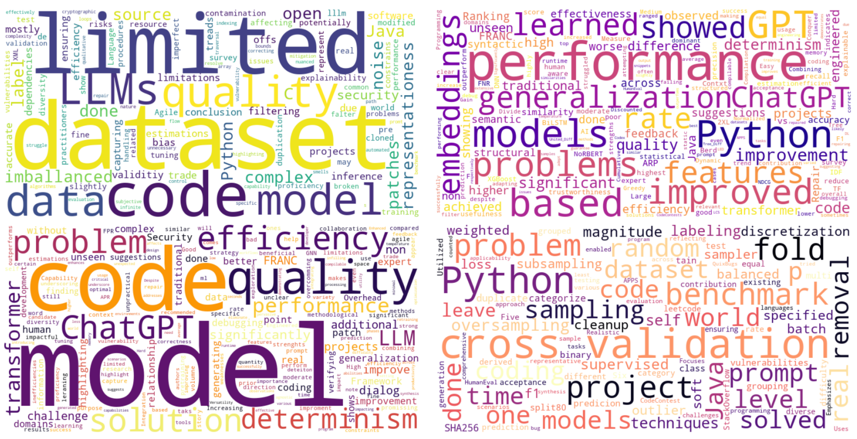
Figure 5: Word cloud visualization of term frequencies across four attributes: "Challenges and Limitations" (upper left), "Results and Findings" (upper right), "Authors' Interpretation" (lower left), and "Sampling Strategy" (lower right), illustrating key thematic focuses in the reviewed literature.

3.4 Ongoing identification of Patterns

In the process of systematically examining and analyzing the attribute values outlined in Subsection 3.3, discernible patterns emerged, elucidating the core challenges targeted by distinct groups of research papers. Notably, certain groups are actively addressing issues related to the limited representativeness of training data, implementing a variety of data sampling strategies to alleviate these concerns. Concurrently, other groups concentrate on exploiting the inherent characteristics of Large Language Models (LLMs), particularly their non-deterministic output. These thematic clusters signify the diverse methodological approaches and areas of emphasis within the realm of software engineering research involving LLMs. The subsequent discussion introduces these groups in detail, highlighting their common objectives, challenges faced, and the innovative methods applied to navigate these obstacles.

Group 1: *Different data sampling strategies* for coping *Data Limitation Challenges* in AI-Enhanced *Planning and Estimation Tasks*:

The studies within this group address prevalent *data-related challenges*, including imbalance [8], label noise [14], representativeness to real world projects and bias [1] [6]. They are distinguished by their use of diverse and *specialized sampling strategies* such as project-level cross-validation [8], self-supervised labeling [14], outlier removal, and magnitude discretization [6].

- NoRBERT: Transfer Learning for Requirements Classification [8]
- PRCBERT: Prompt Learning for Requirement Classification using BERT-

based Pretrained Language Models [14]

- Evaluation of Context-Aware Language Models and Experts for Effort Estimation of Software Maintenance Issues [1]
- GPT2SP: A Transformer-Based Agile Story Point Estimation Approach [6]

Group 2: Software Quality and Security Evaluation Focused on Robustness and Generalizability:

This group delves into enhancing software systems' quality and security, prioritizing the evaluation of model performance on unseen projects to gauge generalization to new data and emphasizing the significance of False Positive rates for reliable security and quality assessments. By focusing on these aspects, the studies aim to ensure the robustness and applicability of their methodologies across diverse and previously unseen software projects, highlighting efforts to minimize false alarms while effectively detecting genuine issues.

- DiverseVul: A Novel Vulnerable Source Code Dataset for Deep Learning-Based Vulnerability Detection [4]
- Transformer-Based Language Models for Software Vulnerability Detection [22]

Group 3: LLM *Reliability and Consistency* in *Code Generation* and *Debugging*:

Group 3 delves into the challenges and opportunities presented by the non-determinism of LLM outputs in code generation and debugging. While one study explicitly examines non-determinism's effects on code quality and reliability [15], the others implicitly leverage this characteristic by employing strategies that involve multiple attempts or iterations. Specifically, one approach involves attempting bug fixing multiple times [19], and another generates several code snippet variants to filter, rank, and refine [18].

- A Lightweight Framework for High-Quality Code Generation [18]
- LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation [15]
- An Analysis of the Automatic Bug Fixing Performance of ChatGPT [19]
- Extending the Frontier of ChatGPT: Code Generation and Debugging [17]

Group 4: LLMs in Dialog mode - Framework in automatic Feedback for Code Generation Tasks:

This group explores LLMs' dialogue capabilities in tasks like bug fixing, automatic program repair (ARP), and debugging. The research contrasts single-prompt generation with the improved outcomes from providing feedback, noting performance enhancements but also the need for balance due to the additional feedback effort . Future studies on automated feedback are encouraged [19]. [18] introduces a framework for high-quality code generation by automatically providing feedback on potential code or security smells. [25] demonstrate the efficacy of iterative feedback in ARP tasks, showcasing conversational AI's potential. However, [17] identifies limited adaptability in refining solutions based on feedback, with a success rate of 36.7% in revised attempts, indicating room for improvement in LLMs' debugging and learning from errors.

- A Lightweight Framework for High-Quality Code Generation [18]
- Conversational automated program repair [25]
- Extending the Frontier of ChatGPT: Code Generation and Debugging [17]
- An Analysis of the Automatic Bug Fixing Performance of ChatGPT [19]

4 Results

5 Conclustions

References

[1] Mohammed Alhamed and Tim Storer. Evaluation of context-aware language models and experts for effort estimation of software maintenance issues. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 129–138. IEEE, 2022.

[2] Friedrich L Bauer. Software engineering—wie es begann. *Historische Notizen zur Informatik*, pages 72–75, 2009.

[3] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 2023.

[4] Yizheng Chen, Zhoujie Ding, Lamya Alowain, Xinyun Chen, and David Wagner. Diversevul: A new vulnerable source code dataset for deep learning based vulnerability detection. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 654–668, 2023.

[5] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. Large language models for software

engineering: Survey and open problems. *arXiv preprint arXiv:2310.03533*, 2023.

[6] Michael Fu and Chakkrit Tantithamthavorn. Gpt2sp: A transformer-based agile story point estimation approach. *IEEE Transactions on Software Engineering*, 49(2):611–625, 2022.

[7] Zishan Guo, Renren Jin, Chuang Liu, Yufei Huang, Dan Shi, Linhao Yu, Yan Liu, Jiaxuan Li, Bojian Xiong, Deyi Xiong, et al. Evaluating large language models: A comprehensive survey. *arXiv preprint arXiv:2310.19736*, 2023.

[8] Tobias Hey, Jan Keim, Anne Koziolek, and Walter F Tichy. Norbert: Transfer learning for requirements classification. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 169–179. IEEE, 2020.

[9] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *arXiv preprint arXiv:2308.10620*, 2023.

[10] Staffs Keele et al. Guidelines for performing systematic literature reviews in software engineering, 2007.

[11] Barbara Kitchenham, Lech Madeyski, and David Budgen. Segress: Software engineering guidelines for reporting secondary studies. *IEEE Transactions on Software Engineering*, 49(3):1273–1298, 2022.

[12] Stefan Kombrink, Tomas Mikolov, Martin Karafiát, and Lukás Burget. Recurrent neural network based language modeling in meeting recognition. In *Interspeech*, volume 11, pages 2877–2880, 2011.

[13] Caroline Lemieux, Jeevana Priya Inala, Shuvendu K Lahiri, and Siddhartha Sen. Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 919–931. IEEE, 2023.

[14] Xianchang Luo, Yinxing Xue, Zhenchang Xing, and Jiamou Sun. Prcbert: Prompt learning for requirement classification using bert-based pretrained language models. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–13, 2022.

[15] Shuyin Ouyang, Jie M Zhang, Mark Harman, and Meng Wang. Llm is like a box of chocolates: the non-determinism of chatgpt in code generation. *arXiv preprint arXiv:2308.02828*, 2023.

[16] Brian Randell. The 1968/69 nato software engineering reports. *History of software engineering*, 37, 1996.

[17] Fardin Ahsan Sakib, Saadat Hasan Khan, and AHM Karim. Extending the frontier of chatgpt: Code generation and debugging. *arXiv preprint arXiv:2307.08260*, 2023.

[18] Mohammed Latif Siddiq, Beatrice Casey, and Joanna Santos. A lightweight framework for high-quality code generation. *arXiv preprint arXiv:2307.08220*, 2023.

[19] Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. An analysis of the automatic bug fixing performance of chatgpt. *arXiv preprint arXiv:2301.08653*, 2023.

[20] Ian Sommerville. *Software Engineering*, pages 5–13. Publisher Name, 9 edition, 2011.

[21] Yutian Tang, Zhijie Liu, Zhichao Zhou, and Xiapu Luo. Chatgpt vs sbst: A comparative assessment of unit test suite generation. *IEEE Transactions on Software Engineering*, 2024.

[22] Chandra Thapa, Seung Ick Jang, Muhammad Ejaz Ahmed, Seyit Camtepe, Josef Pieprzyk, and Surya Nepal. Transformer-based language models for software vulnerability detection. In *Proceedings of the 38th Annual Computer Security Applications Conference*, pages 481–496, 2022.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[24] Jesse Vig. A multiscale visualization of attention in the transformer model. *arXiv preprint arXiv:1906.05714*, 2019.

[25] Chunqiu Steven Xia and Lingming Zhang. Conversational automated program repair. *arXiv preprint arXiv:2301.13246*, 2023.

[26] Catherine Yeh, Yida Chen, Aoyu Wu, Cynthia Chen, Fernanda Viégas, and Martin Wattenberg. Attentionviz: A global view of transformer attention. *IEEE Transactions on Visualization and Computer Graphics*, 2023.

[27] Zibin Zheng, Kaiwen Ning, Jiachi Chen, Yanlin Wang, Wenqing Chen, Lianghong Guo, and Weicheng Wang. Towards an understanding of large language models in software engineering tasks. *arXiv preprint arXiv:2308.11396*, 2023.

[28] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.

[29] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.