Utrecht University

Enhancing the ECSER pipeline: Evaluating Large Language Model Classifiers in SE Research

Ruben van der Zeijden (7081111)

Supervisor: Dr. F.B. Aydemir
Second Reader: Dr. D. Dell'Anna

Abstract

The various research papers in the field of Software Engineering (SE) that use classification algorithms, LLMs, or other machine learning methods to obtain their results differ in how many and which evaluation metrics are reported, whether or not significance tests are performed, and which steps are taken to aid reproducibility. The ECSER pipeline was designed to mitigate this issue by providing a step-by-step pipeline that researchers can use to report their results when classification algorithms are used, and the pipeline was empirically shown to be effective in replicating the findings of several studies, as well as producing additional findings and occasionally contradicting the findings of the original papers. However, the ECSER pipeline is designed for evaluating simple classifiers and gives no specific recommendations for LLM classifiers, despite LLMs being an increasingly popular choice for classification tasks. The goal of this thesis is to expand the ECSER pipeline for the use of LLMs by adding recommendations from LLM4SE research and related fields. First, an exploratory mapping study was done of SE studies released in 2024 that use LLM classifiers, summarising which steps were taken and which metrics were (or weren't) reported, in order to get an overview of the current state of LLM4SE research. Subsequently, we designed the *ECSER-LLM* pipeline, an enhanced version of the ECSER pipeline for the use of LLMs, including recommendations for prompt engineering, evaluating the fairness and robustness of models, and more. Lastly, in order to evaluate the comprehensiveness and ease-of-use of the new pipeline, two replication studies were conducted using the pipeline to test its ability to strengthen or contradict the findings of the original papers.

MSc. Artificial Intelligence
30EC
The Ethics and Privacy Quick Scan of the Utrecht University Research Institute of Information and Computing Sciences classified this research as low-risk with no fuller ethics review or privacy assessment required.

Contents

1 Introduction

In recent years, large language models (LLMs) have become an increasingly relevant tool in software engineering (SE), allowing researchers to accomplish complex tasks usually reserved for humans or other machine learning (ML) methods, such as code generation, summarization and classification. However, few guidelines exist for conducting and reporting on LLM research, leaving researchers on their own to decide what choices to make when designing prompts, what evaluation metrics to report in their results and the level of statistical validation needed for their comparisons. Because of these lack of guidelines, researchers using LLMs often omit important details, such as the F1 score or the significance of their results [1]. This problem is not new or unique to LLMs; many researchers have reported that there is a poor standard of empirical software engineering, which inhibits the usefulness of results and impairs reproducibility [2, 3].

To treat the observed poor standard of empirical SE, some researchers have called for a more systematic approach to conducting and evaluating experiments. The ECSER (Evaluating Classifiers in Software Engineering) pipeline, introduced by Dell'Anna et al. [4], provides a systemic pipeline for training, validating and testing ML classifiers and analysing their results. ECSER is based on recommendations from the field of machine learning for software engineering (ML4SE) and is designed to be easy to use for SE researchers, regardless of expertise with machine learning methods. The ECSER pipeline has seen some empirical validation in the form of two replication studies, in requirements engineering and software testing, where Dell'Anna et al. were able to confirm and strengthen some findings, as well as discover additional findings or findings that contradict the original ones. Figure 1 shows the steps of the ECSER pipeline.

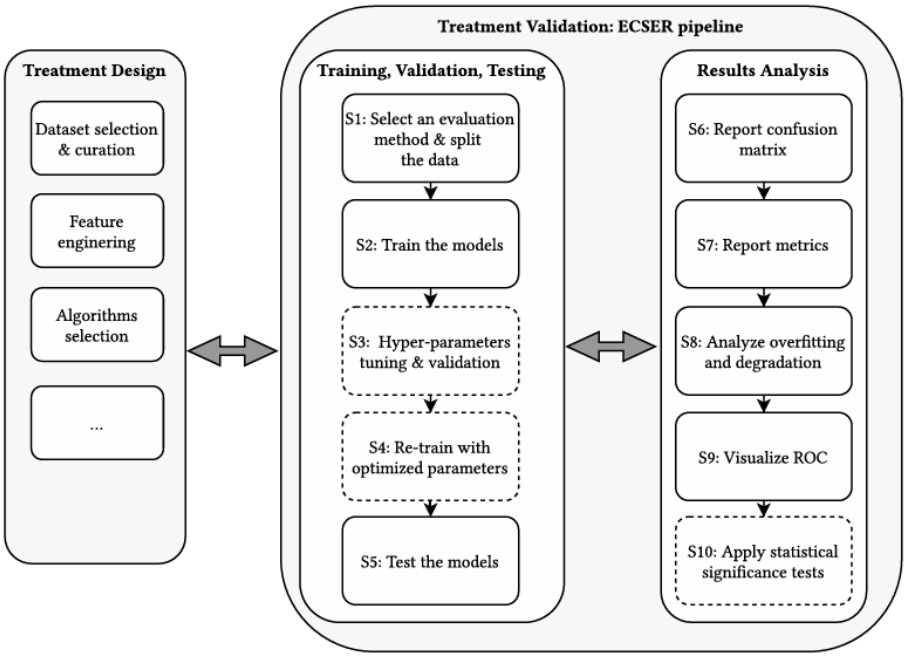

Figure 1 The ECSER pipeline. Reproduced from Dell'Anna et al. [4]. Steps with a dashed border are optional. The ↔ arrows indicate feedback loops between phases.

However, while ECSER is tailored to traditional ML classifiers (such as logistic regression, SVMs and decision trees), it includes no specific recommendations for LLM-based classifiers, despite LLMs showing

promise for various classification tasks [5, 6]. Thus, the ECSER pipeline is not sufficient for LLM research, since using LLMs for classification comes with unique considerations that are not present for traditional classifiers. For one, it is common to use pre-trained, generalised LLMs and give task-specific instructions to get the desired classification on the input data. This means that designing these instructions, also called prompts, becomes a crucial part to conducting LLM research [7]. There are also unique considerations when it comes to evaluating the fairness and robustness of the results: does the model behave in accordance with human values and are the results resilient to adversarial prompts, such as those containing typos [8, 9]?

The goal of this thesis is to address the lack of LLM-specific guidelines in the ECSER pipeline, by expanding it to include specific recommendations for conducting LLM classifier research. In doing so, we made the following contributions:

- We conducted a mapping study to assess the current state of reporting in LLM4SE classification research.
- We developed the *ECSER-LLM* pipeline, an expanded version of the ECSER pipeline that includes specific recommendations for conducting LLM classifier research, such as prompt engineering and the evaluation of model fairness and robustness.
- We conducted two replication studies of existing LLM classifier papers using the *ECSER-LLM* pipeline, in order to evaluate the pipeline's applicability and comprehensiveness.

Supplemental materials, including all code that was used, the full intermediary and final results and the source for this document can be found on our GitHub page[1].

The remainder of this thesis is structured as follows. In Section 2, we provide background information about LLMs. Section 3 describes the research approach and research questions. Lastly, Section 4 discusses related work.

2 Background

Language Models (LMs) are computational models designed to model the generative likelihood of word sequences [10]. In other words, given a sequence of input text, they predict the next word in the sequence based on what is deemed most likely by the model. One of the oldest examples of language models are N-grams, which model the likelihood of a word given the previous N tokens [11]. Early statistical models like N-gram models suffer from several problems, including limited context and difficulty with unseen words.

Pre-trained Language Models (PLMs) are LMs that are designed to take in vast amounts of unannotated text in order to learn knowledge about language and the world [12]. The knowledge that is contained in these models can then be used on new tasks by way of **transfer learning** [13]. PLMs require much more information to be contained within than statistical language models such as N-grams could provide, which has led researchers to design more sophisticated models that rely on neural networks to model language, starting with recurrent neural networks (RNNs) and later Long Short-Term Memory (LSTM) models [14, 15]. The biggest recent breakthrough came with the invention of **transformers**, which performed better than previous models, were easier to train and allowed for the parallel processing of inputs [16]. Transformers typically consist of an **encoder**, which embeds the input sequence into a hidden (latent) space and a **decoder**, which translates the abstract representation of the hidden space to the target text. Both the encoder and decoder use a self-attention mechanism to weigh the contribution of each token to the output [16, 17].

[1] `https://github.com/rubenvdz/Master-Thesis-Enhancing-the-ECSER-pipeline`

Large Language Models (LLMs) were introduced as a way to refer to PLMs with massive parameter sizes, since it was found that larger parameter sizes lead to better performance and emergent abilities [18, 19]. Almost all LLMs use transformer models with the self-attention mechanism, but not all of them use both the encoder and decoder component [20]:

- **Encoder-decoder LLMs**, such as BART [21] and T5X [22], are good at both understanding and generating language [20]. As such they are useful for tasks like summarization and translation [23, 24].
- **Encoder-only LLMs**, such as BERT and its variants [25–28], are tailored towards language understanding [20]. As such they are good at tasks like classification and inference [29].
- **Decoder-only LLMs**, such as the GPT-series [30–32], are tailored towards language generation [20]. As such they are good at all generation tasks, including code generation [33] and creating conversational agents like ChatGPT [34] and LLama 2 [35].

Prompt Engineering is the process of systematically designing natural language instructions (or **prompts**) to guide LLMs towards a desired output [36]. Prompts condition the outputs of the LLM on the given instructions, so that the model generates the next tokens with both the prompt and previously generated tokens in mind [12]. The most common approaches to prompt engineering are:

- **Zero-shot Prompting**. The model is given no examples for a task, forcing it to rely entirely on the prompt [37].
- **Few-shot Prompting**. A limited number of examples for a task are given, which the model can learn from [38].
- **Chain-of-Thought Prompting**. The prompt instructs the model to follow coherent reasoning steps [39].

Another less common approach is automatic prompt engineering, where instructions are automatically generated and selected [40].

3 Research Questions

The usage of LLMs in SE research has increased massively over the past few years, but the research of how to best conduct studies and report results in this field remains very limited. In order to approach the problem of LLM4SE reporting standards and design meaningful guidelines, we follow Wieringa's design cycle of *problem investigation*, *treatment design* and *treatment validation* [41]. This leads us to the following research questions:

RQ1: What is the current state of reporting in LLM4SE classification research?
RQ1 fulfils the step of *problem investigation* and is both descriptive and evaluative: what do researchers report in their studies and does this level of reporting contribute or detract from the accuracy and reproducibility of the results. The aim of this question is to provide insights into the potential shortcomings of existing research by conducting a mapping study of the reporting practices of LLM4SE research, in order to provide a basis for additional guidelines that help researchers avoid these shortcomings. Section 5 presents the research approach of RQ1 and the results of the mapping study.

RQ2: What resource can be developed to assist researchers in conducting and reporting on LLM4SE classification research?
Following the problem investigation of RQ1, RQ2 fulfils the step of *treatment design*. The purpose of RQ2 is to design a tangible artefact that is comprehensive and widely applicable to the field of LLM4SE, with the

intent of helping researchers avoid any problems identified in RQ1. This research question was designed to guide the modification of the existing ECSER pipeline [4], taking into consideration recommendations from the fields of LLM4SE, ML4SE and Design Science. Section 6 presents the research approach of RQ2 and the resulting *ECSER-LLM* pipeline.

RQ3: How applicable is the proposed pipeline in assisting LLM4SE classification research?
After designing the *ECSER-LLM* pipeline, RQ3 follow the step of *treatment validation*: does the proposed pipeline achieve its goals of applicability and comprehensiveness. The purpose of RQ3 is to apply the pipeline to existing LLM4SE classification research by way of a replication study, in order to evaluate whether the pipeline is able to replicate the research and whether there can be added benefit to using the pipeline. Section 7 presents the research approach of RQ3 and the results of the replication study.

4 Related Work

The main focus of this thesis is to study the usage and evaluation of LLM Classifiers in Software Engineering, but in doing so we touch on several related research fields, including Empirical Software Engineering, Machine Learning for Software Engineering (ML4SE), Large Language Models for Software Engineering (LLM4SE) and Prompt Engineering. Additionally, each of the three research questions has related work that is relevant to their methodologies. Hence, we have split the related work into several sections based on different areas of study to aid readability.

First, we discuss challenges in empirical software engineering which motivated the creation of the ECSER pipeline and further motivated our mapping study of LLM4SE. Second, we discuss the topic of replication in SE, which is relevant to our replication studies and the field of empirical SE. Third, we discuss how Large Language Models have been used in SE, with a focus on classification tasks. Fourth, we discuss related work in Prompt Engineering, as it has become a vital part to using LLMs and informed our guidelines for prompt engineering in the *ECSER-LLM* pipeline. Finally, we discuss related work on the evaluation of LLMs and classifiers informed our guidelines for the evaluation of LLM Classifiers.

4.1 Challenges in Empirical Software Engineering

Over the past couple decades, the use of machine learning methods has become ubiquitous in every field of science, among them software engineering. Zhang and Tsai [42] summarized seven main activities that use machine learning in SE: prediction, property/model discovery, transformation, generation, library construction and maintenance, acquisition of specifications and development knowledge management. In addition, they list specific tasks belonging to each activity, such as *predicting* software quality and *generating* test cases. The field of classification research falls almost exclusively in the prediction category.

With the increased popularity of machine learning methods, many software packages have been developed to allow anyone to use these methods without having to write any code or even understand how these methods work. This leads to many researchers using machine learning without fully understanding the underlying assumptions of the methods they are using and assuming the results are correct without proper (statistical) analysis. This problem is not new: Kitchenham et al. [2] reported at the start of this century that there was a poor standard of empirical software engineering, mostly caused by a lack of statistical expertise from both the authors and reviewers. They go on to provide extensive guidelines for designing and conducting experiments as well as analysing, presenting and interpreting results. In another paper, Kitchenham et al. [43] discuss the benefits of evidence-based software engineering approach by drawing an analogy to evidence-based medicine, but note that the infrastructure needed for widespread adoption of evidence-based SE isn't there yet, which means that SE experiments are vulnerable to subject and experimenter bias.

Methodological problems in SE research can result in problems with the reproducibility of results, which severely reduces the real-world usefulness of said results. Menzies and Shepperd [3] argue that the main goal of science is *conclusion stability*, which means that when you discover an effect, it holds true irrespective of the situation. However, they note that in the software engineering literature there is often as much evidence *for* a given effect, as *against*. This lack of conclusion stability is caused by sources of bias and variance that are introduced at various steps: sampling, pre-processing, training, algorithm selection and the analysis of results. If these sources of bias and variance are accurately analysed and reported, that would make it easier for the authors and other researchers to interpret the conclusions.

Despite being a known issue, the problems of improper analysis and reporting have persisted over the years. In a mapping study of the proceedings of the International Conference of Software Engineering (ICSE) from 2019 to 2021, Dell'Anna et al. [4] analysed 60 SE papers related to classification and found that most papers (36) reported on the precision and recall of their experiment, fewer reported the F-score (27) and accuracy (24), but only 14 papers explicitly justified why they chose the reported metrics. The full confusion matrix was only included in six papers. Other metrics like the receiver operator characteristic (ROC) curve and the area under the curve (AUC) were mostly absent with 3 and 9 respective mentions. Of course, as we know from statistics, it is not enough to report a metric and claim it's better than that of other models, since it could be coincidental, so statistical significance testing is required to make such claims. However, only six papers analysed the statistical significance of their results. Evidently, the situation is dire, which is why Dell'Anna et al. developed the ECSER pipeline to give SE researchers a systematic approach to conducting classification experiments and reporting and analysing the results.

4.2 Replications in SE

Replication studies are a vital part of empirical software engineering, allowing researchers to investigate the effects of alternative values for important attributes, vary the strategy with which hypotheses are investigated and make up for certain threats to validity [44]. The importance of replication is not unique to software engineering, after all the so-called "replication crisis" was a massive topic of debate in psychology research in recent years, since it was found that many previous results in the field could not be replicated, even if the original findings were statistically significant [45–47]. Cruz et al. [48] conducted a systematic literature review of replication studies in SE between 2013 and 2018 and found that the number of published replication studies has been steadily increasing, though there are research gaps in certain fields, suggesting that the SE community is taking an increased interest in replicating previous studies.

Shull et al. [49] identify two types of replication studies: *exact replications*, in which the procedures of the original study are strictly followed; and *conceptual replications*, in which the research problem of the original study is evaluated using a different procedure. Additionally, exact replications are subdivided into *dependent* and *independent* replications. Dependent replications keep most of the conditions of the experiment (close to) the same, which allows one to test the effect of specific variables on the results; whereas independent replications vary the original conditions of the experiment significantly, which allows one to test the robustness of the original results under different conditions. Juristo and Vegas [50] underline the value of non-exact replications, which they define similarly to how Shull et al. define independent exact replications, by conducting a multiple-case replication study using a newly proposed four-phase process, to show that non-identical replications can be used to learn new information about the original results.

While there is a consensus in SE that replication is important, Shepperd et al. [51] show that replication studies often suffer from the same problems as original studies, such as incomplete reporting, low power or potential researcher bias. Notably, many replication studies in their survey did not report any details on the dispersion (e.g., variance) of the response variables. In another paper, Shepperd [52] shows that because

of wide prediction intervals, almost all replications are confirmatory, which means the added knowledge is negligible. Because of these problems, they suggest that researchers should focus more on broader meta-analyses instead of replication studies. Other common problems in conducting replication research include difficulty in getting them published, lack of guidelines and the unavailability of replication packages [48, 53].

Communication can also play a factor in successful replication: Vegas et al. [54] investigated the role of communication in successful replication of previous experiments and found that there should be some level of communication, orally or in writing, between the previous authors and authors of the replication study, to avoid unnecessary changes to the experiments.

Given these challenges, guidelines for conducting replications are clearly needed. Despite this, the number of existing guidelines remains low. Guidelines were proposed by Carver [55], which are still commonly used in replications, but these seem to be the only replication guidelines for software engineering and were intended as a starting point for future guidelines. Other artifacts do exist to aid replications: Gómez et al. [56] proposed a classification of replications in SE, differentiating between literal, operational and conceptual replications, that can be used to identify which changes can be made in each type of replication and understand the level or verification needed for that type.

Abualhaija et al. [57] proposed an artifact, referred to as the ID-Card, for summarizing papers in RE research that use NLP techniques. The intention behind the ID-Card is that all the relevant information needed for replication is presented in an easy to read format and can be used for both replication and educational purposes. Lastly, Brandt et al. [58] developed the "Replication Recipe" for psychology research, which lists 36 questions that should be addressed when conducting a replication, most of which are also directly applicable to empirical SE.

4.3 Large Language Models for SE

A recent development in SE research is the increased relevance and use of LLMs. Fan et al. [59] looked at all preprints on arXiv categorised under computer science whose title included "LLM", "Large Language Model" or "GPT" and found the number of SE papers that mentioned LLMs grew exponentially from 0 in 2019, 5 in 2020, to 181 in 2023. Naturally, this is only an approximation of the total number of preprints on arXiv that use LLMs, but it undoubtedly signifies a substantial increase. Fan et al. also conducted an extensive survey of how LLMs have been applied to various software development activities (such as code generation and testing) and research domains (such as human-computer interaction and education), but the survey is almost exclusively focused on the generative use of LLMs and does not go into depth on the use of LLMs in classification research. Many other good surveys and analyses of the use of LLMs in Software Engineering have been written, but most of them similarly do not mention any applications to classification tasks and instead exclusively focus on generation [60–62].

Surveys have also been written about specific domains of software engineering: Wang et al. [63] did a comprehensive review of 102 studies that use LLMs for software testing and found that LLMs have commonly been applied to various tasks, including test case generation, test input generation, debugging, program repair and more. They also found that while most papers used LLMs to address the entire task, many others combined LLMs with additional techniques to optimize the outputs of the LLM, including mutation testing, differential testing, syntactic checking, program analysis, statistical analysis and other techniques. With these extra techniques, researchers were able to generate more diverse and complex code and overcome some of the limitations of LLMs.

While LLMs are most frequently used to generate text (and code), they have seen considerable success in classification tasks. Guo et al. [5] have shown that LLMs can outperform other methods like SVMs for health-

related text classification. Fields et al. [6] present an in-depth survey of text classification using transformers across domains and found that LLMs can perform remarkably well on many (but not all) classification tasks. However, Chen et al. [64] compared the performance of various LLMs to traditional ML methods in clinical prediction and found that LLMs could *not* beat traditional methods in this case. Vajjala and Shimangaud [65] also show that there are large performance disparities between languages in classification tasks. Thus, LLMs might not always be a better choice than traditional ML methods, but they certainly show potential.

Software engineers have also started using LLMs for classification. Hou et al. [20] conducted a systematic literature review of 395 software engineering studies that use LLMs and found that around 21.61% involve classification tasks. The most common classification tasks where LLMs have been used include vulnerability detection, requirements classification, bug prediction and review/commit/code classification, with many other classification tasks having been attempted using LLMs. Zhang et al. [66] conducted a systematic survey of 947 SE studies and summarized 62 unique LLMs of code, including six LLMs specifically fine-tuned for the tasks of code classification, clone detection and defect detection.

Recently, Baltes et al. [67] presented a list of community-driven guidelines for using and reporting on LLMs, providing eight detailed guidelines. These guidelines include: (1) declaring LLM usage and role; (2) reporting model versions, configurations and fine-tuning; (3) documenting tool architectures; (4) disclosing prompts and interaction logs; (5) using human validation; (6) employing an open LLM as baseline; (7) using suitable baselines, benchmarks and metrics; and (8) openly articulating limitations and mitigations. These guidelines were designed to increase the transparency and reproducibility of LLM research and are actively maintained on `https://llm-guidelines.org/`. The guidelines in this thesis serve a similar purpose of helping researchers conduct LLM4SE research; however, their guidelines have a broader focus, since they include all LLM applications, whereas our guidelines are specifically tailored to classification research. Furthermore, unlike their paper, this thesis also includes a mapping study to assess the state of LLM4SE classification reporting and provides a replication study to evaluate the guidelines. Despite the difference in scope, many of their recommendations directly apply to our research and were taken into account for the *ECSER-LLM* pipeline where relevant.

4.4 Prompt Engineering

Designing a good prompt for a particular task is vital to ensuring the adequate performance of LLMs, which is why much research has gone into the systematic designing of prompts, or prompt engineering. Marvin et al. [7] provide an overview of prompt engineering principles and techniques and outline the main steps involved in the process of prompt engineering: 1) define the goal of the prompt; 2) understand the model's capabilities; 3) choose the right prompting format; 4) provide context to the LLM and 5) test and refine the prompt based on the goal. Several resources exist to make it easier to choose the right prompting approach: White et al. [68] introduce prompt patterns, which offer reusable solutions to common prompting challenges, and provide a framework for designing and documenting these patterns in terms of the intent, motivation, structure and consequences of the patterns. For example, the persona pattern can be used to make the LLM take the point of view of an expert in the field, which is especially useful if the user itself is not an expert. The paper provides a catalogue of this pattern and other successfully applied patterns with example implementations. Further, Ekin [69] provides an accessible (AI-generated) guide to prompt engineering.

Prompt engineering is an important consideration when using LLMs: Sclar et al. [70] show that choices in prompt design, even if minor, can have a very large impact on the performance of several popular LLMs, and this sensitivity is present regardless of the size of the model or the number of examples provided. Because of this, they argue that researchers should forego using only a single prompt and switch to reporting the performance of their models using multiple plausible prompt patterns. They present a novel algorithm,

FormatSpread, to evaluate multiple prompts at the same time and measure the sensitivity of the LLM. Mizrahi et al. [71] have found a similar sensitivity of LLMs to prompt design and discuss metrics to evaluate multiple prompts. Lastly, Maia Polo et al. [72] propose PromptEval, an efficient method for evaluating a large number of prompts, and show it can accurately estimate the performance distribution of the prompts.

Prompt engineering research has also been conducted specifically for software engineering: Hou et al. [20] looked at what prompt engineering techniques are commonly applied in SE tasks and found the most common techniques involve few-shot prompting and zero-shot prompting, but the third largest group of studies had no explicit mention of prompting techniques or proposed their own strategies. Other techniques included chain-of-thought, automatic prompt engineering, chain-of-code, automatic chain-of-thought, modular-of-thought and structured chain-of-thought.

Arvidsson and Axell [73] collected prompt engineering recommendations for requirements engineering (RE) in a systematic literature review and classified the guidelines into various themes, after which they interviewed three RE experts to evaluate the guidelines. Ronanki et al. [36] evaluated the effectiveness of 5 prompting patterns (from the catalogue by White et al. [68]) on two RE tasks and proposed a framework for evaluating the effectiveness of prompting patterns for any RE task, providing five steps for conducting a comparison of patterns.

A recent development in prompt engineering is automatic prompt engineering (APE), which has shown good performance compared to baseline models, while avoiding the need for manually designing prompts [40, 74]. Zadenoori et al. [75] investigated the use of automatic prompt engineering in RE and found that, on average, APE outperforms traditional prompt engineering techniques. However, research is still limited.

4.5 Evaluating LLM Classifiers

Few papers have been written specifically about the evaluation of LLM-based classifiers, but we can look at the evaluation of classifiers and LLMs separately to get a picture. The ECSER pipeline is the most relevant for evaluating classifiers, since it gives in-depth recommendations about the conducting and reporting of classifier research [4]. Specific recommendations include the reporting of the full confusion matrix, reporting other metrics relevant to the domain such as specificity (true negative rate), analysing overfitting and degradation, visualising the ROC and applying statistical significance tests. Some researchers also recommend reporting the Matthews correlation coefficient (MCC), which combines all four metrics of the confusion matrix and can be more informative than the F1 score [76–78].

Hou et al. [20] looked at whether these metrics were reported in LLM-based classification research and found that out of 147 papers that use LLMs for classification, the most frequently reported metric is precision with 35 papers, followed by recall (34), F1-score (33) and accuracy (23), but the AUC was reported 9 times, the ROC only 4 times and the MCC only twice. Unfortunately, they did not count how many studies did significance testing. These results are very similar to Dell'Anna et al. [4]'s aforementioned mapping study of classifier research as a whole and suggest that many important metrics are under-reported in LLM-based classification research.

Using and evaluating LLMs comes with more considerations than just the accuracy of the results. Guo et al. [1] categorize the evaluation of LLMs into three types: knowledge and capability evaluation, which assesses the fundamental knowledge and reasoning capabilities of the LLM; alignment evaluation, which refers to evaluating ethical and moral considerations like bias; and safety evaluation, which focuses on the robustness of LLMs and risk evaluation. Liu et al. [79] have created guidelines specifically for evaluating LLM alignment in terms of reliability, safety, fairness, resistance to misuse, explainability and reasoning, adherence to social norms and robustness.

Another consideration is how well the LLM outputs adhere to the format specified in the prompt. For example, it is common to ask an LLM to format its answer as a valid JSON object, allowing for easier processing of the output. Limiting responses to a specified format can be beneficial: Tam et al. [80] compared the performance of several LLMs between free-form response and formatted responses and found that, while reasoning ability was weakened for formatted responses, classification accuracy was increased. However, LLMs do not always follow the format accurately. Long et al. [81] define several measures to analyse the level of adherence to the output format: SysE, which measures the performance of the LLM for answers that *meet the constraint*; TrueE, which measures the performance of all answers *regardless of whether the format is satisfied*; and BiasF, which measures the (mean squared) difference between SysE and TrueE. Li et al. [82] introduce the JScore measure, which measures the similarity between JSON objects and can be used to compare LLM generated JSON objects with the target JSON objects. Lastly, Xia et al. [83] introduce the FoFo benchmark for evaluating LLMs based on their ability to follow complex, domain-specific formats.

Chang et al. [84] conducted another survey of the evaluation of LLMs which focuses on downstream applications, resulting in a classification of four aspects of LLM evaluation: the accuracy, containing measures of correctness such as the F1 score; the calibration, which contains measures for the degree of alignment between the predicted probabilities and actual probabilities, such as the expected calibration error (ECE); the fairness, with measures pertaining to the equal treatment of different groups, such as the equalized odds difference (EOD); and robustness, with measures pertaining to the performance of a model in the face of challenging inputs and noise, such as performance drop rate (PDR). Further, they mention several ways in which human evaluation can be used to evaluate LLMs, such as the degree to which the model outputs align with human values.

Lastly, the model temperature, which regulates the randomness of the model, can also affect its performance. Peeperkorn et al. [85] looked at the effect of model temperature on the level of creativity of the outputs and found that LLMs generate slightly more novel outputs with higher temperature, but the temperature was also correlated with incoherence. No relationship was found between temperature and cohesion or typicality. Model temperature might also affect a model's susceptibility to security attacks: Yu et al. [86] found that some LLMs became more susceptible to jailbreaking attacks as temperature increased, whereas others had decreased susceptibility to jailbreaking with increased temperature, possibly due to the fact that the former models had a lower susceptibility at 0 temperature and the latter models had a higher susceptibility at 0 temperature.

5 RQ1: What is the current state of reporting in LLM4SE classification research?

In this section, we first discuss the methodology used to conduct a systematic mapping study on the state of reporting of LLM4SE classification research. Subsequently, we discuss the results from the mapping study and draw conclusions from these results.

5.1 Research Approach

To answer the research question, we conducted a mapping study of SE papers published in 2024 that use LLM classifiers using the guidelines by Kitchenham et al. [87, 88] and Petersen et al. [89], consisting of five steps:

1. Define the research question.
2. Conduct a search for primary studies.

3. Screen papers based on inclusion/exclusion criteria.
4. Classify the papers.
5. Extract data.

What follows is a detailed summary of the search strategy (steps 2 and 3) and the information that was obtained from the list of relevant papers (steps 4 and 5).

Search Strategy
To get a complete image of the state of LLM4SE classifier evaluation, we collected all 528 papers published in 2024 in three of the top SE conferences and journals (Table 1). These venues were narrowed down from the list of venues included in [20] based on their impact and relevance. Using the full list of 528 papers, an automatic keyword search was conducted on the title and abstract of each paper to identify the papers that were potentially relevant to LLM4SE.

The list of keywords was based on previous research ([20, 66]) and designed with the intent of high recall, since it is easier to remove false positives than to retrieve false negatives. Keyword matching was done on the lowercased versions of the title and abstract and allowed for partial matches (e.g., "llm" matches "LLM", "LLMs", "LLM4SE", etc.). The complete list of keywords is as follows:

- *llm, large language model, language model, code generation model, plm, pre-trained, pretrained, pre-training, pretraining, chatgpt, gpt, bert, t5, llama, bart.*

The keyword search resulted in 129 potentially relevant papers. After manually removing false positives, mostly caused by the authors comparing their non-LLM model to previous LLM models, the total number of identified LLM4SE papers was 116, representing 21.97% of total SE papers.

To narrow down the list from all LLM-related papers to those pertaining to SE classification research, the title and abstract of the remaining 116 papers were manually screened for the inclusion and exclusion criteria (Table 2). These criteria were designed to exclude any paper not relevant for answering the research question, such as meta studies and those not related to classification. During the manual search, 18 papers were identified that use LLMs for SE classification research and satisfy all other criteria.

Table 1 Publication venues included in the mapping study.

Acronym	Venue
ICSE	International Conference on Software Engineering
FSE	International Conference on the Foundations of Software Engineering
TSE	Transactions on Software Engineering

Table 2 Inclusion/Exclusion Criteria

Inclusion Criteria
The paper uses an LLM.
The paper focuses on a downstream application of the LLM.
The downstream application is a SE classification task.
The full text of the paper is accessible.
The paper was published in 2024.
Exclusion Criteria
The paper focuses on LLM architecture with no downstream application.
The paper or task is not relevant to SE.
The full text is inaccessible.
The paper is a survey or other meta-analysis.
The paper focuses on research methodology.
The paper is a reprint or different version of another paper.

While only classification papers were included in the final analysis, all the papers that applied LLMs to SE tasks (a total of 83 papers) were additionally classified based on the type of LLM application (classification, generation or recommendation) to get a view of the relative academic interest in each type (Figure 2). As can be seen, LLMs are most often used for generation, with 73.5% of papers using them in this way, including some that use generation alongside classification. However, classification represents the second largest application of LLMs, with 22.9% in total. Appendix A contains references for Figure 2 and other pie charts contained in this thesis.

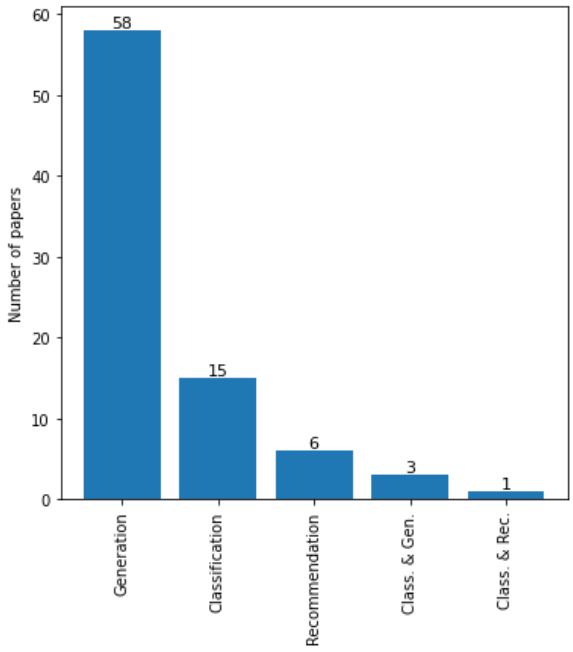

Figure 2 Distribution of LLM application types in collected studies.

Information Extraction

After completing the manual search and obtaining 18 relevant LLM4SE classification papers, the evaluation

and reporting practices of each paper were manually analysed and recorded. The information that was extracted from the papers was determined based on what is recommended to be included by the ECSER pipeline [4], metrics that are recommended by other authors (such as the Matthews correlation coefficient [76–78]) and recommendations from the field of LLM4SE. For the latter, we recorded which metrics (if any) were reported for each of Chang et al. [84]'s classification of general metrics: the accuracy, calibration, fairness & robustness. Since the accuracy was already covered by the ECSER pipeline, this resulted only in the addition of calibration, fairness & robustness to the final list of extracted information.

The calibration of an LLM refers to how well the predicted probabilities of the model align with actual probabilities [90, 91]. Evaluating the fairness of an LLM is done to ensure the alignment of LLM results with human values and prevent a difference in performance across groups of people [79]. The robustness refers to the performance of a model in the face of challenging inputs and can be analysed in several ways, including by making changes in the data distribution, the addition of noise to the data, and adversarial attacks [84].

In total, the following list of evaluation metrics or evaluation metric types were added: *precision, recall, accuracy, F-score, Area under curve (AUC), Matthews correlation coefficient (MCC), Confusion matrix, Receiver Operation Characteristic (ROC) plot, Calibration, Fairness* and *Robustness*.

Further, we looked at several factors related to the reproducibility and generalisation of results, particularly whether justification was given for why the chosen evaluation metrics were used, whether results were obtained for multiple datasets, whether the results were compared to external baselines and whether the statistical significance was analysed and the choice of significance test justified.

Lastly, for the papers that used a prompt-based approach for classification, we analysed whether the prompt approach and final prompt text were reported and justified, whether the results were evaluated over multiple prompts and whether the temperatures of the models were reported.

5.2 Mapping Study Results

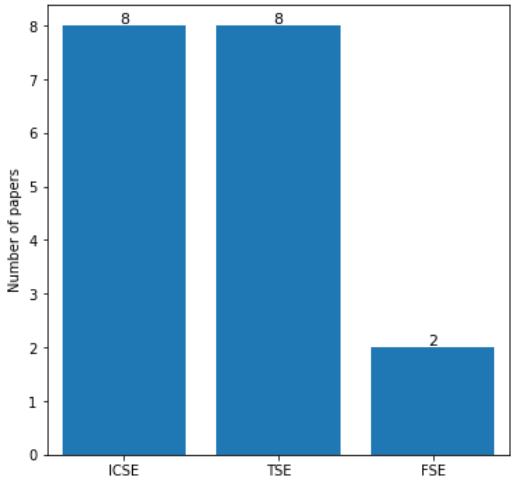

Figure 3 Distribution of venues in mapping study.

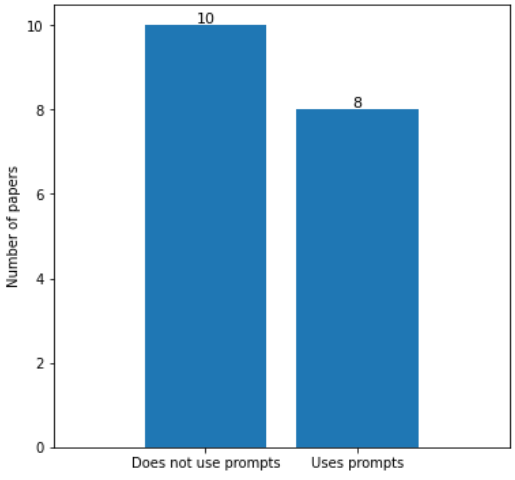

Figure 4 Distribution of prompt usage in mapping study.

During the search process, 18 papers were identified as relevant for answering the research question. Each venue included in the search strategy is represented in the final selection, with eight ICSE papers, eight TSE

papers and two FSE papers (Figure 3). By including papers from three of the most influential Software Engineering venues, we increase the likelihood that the results are representative of the field.

One of the first things we analysed was whether the papers used a prompt-based approach to LLM classification or another approach such as fine-tuning, since additional information about the prompt engineering approach was extracted from the papers where prompts were used. Figure 4 shows that 44.4% or 8 papers used a prompt-based approach, while 55.6% or 10 papers used a different approach. Note that additional references for the figures and the subsequent tables are included in Appendix A.

Table 3 Summary of mapping study results.

Category	**Total**
Total mapping study papers	18
Evaluation metrics	
Precision	17 (94.4%)
Recall	17 (94.4%)
Accuracy	11 (61.1%)
F-Score	17 (94.4%)
AUC	3 (16.7%)
ROC plots	0 (0.0%)
MCC	0 (0.0%)
Calibration	0 (0.0%)
Fairness	0 (0.0%)
Robustness	
Data distribution & Adversarial Attacks	1 (5.5%)
Noisy data	1 (5.5%)
Metrics justification	
No	9 (50.0%)
Implicit	0 (0.0%)
Previous work	5 (27.8%)
Yes	4 (22.2%)
Confusion matrix	4 (22.2%)
Evaluation over multiple data sets	5 (27.8%)
Type of baseline	
None	2 (11.1%)
Own	3 (16.7%)
External	4 (22.2%)
External and Own	9 (50.0%)
Analysis of statistical significance	5 (27.8%)
Statistical significance justification	
No	4 (80.0%)
Implicit	0 (0.0%)
Previous work	1 (20.0%)
Yes	0 (0.0%)

Table 3 shows a summary of the information that was extracted from each paper and how many papers reported each piece of information. This includes which evaluation metrics were reported; whether the calibration, fairness and robustness of the models were analysed; whether the confusion matrix was reported; the

level of justification for the reported metrics; whether the models were evaluated over multiple data sets; the type of baseline used; and whether the statistical significance of the results was reported and justified.

Starting with the evaluation metrics that are recommended by the ECSER pipeline [4], 17 papers reported the precision, recall and F-score of the results, while only 11 papers reported the accuracy and only four papers reported the full confusion matrix. Although less known, some authors also recommend reporting the Matthews Correlation Coefficient (MCC), since it takes the full confusion matrix into account and might therefore give a more balanced representation of the results than the F-score [76], but it was not reported in any of the papers. Further, the AUC, which some researchers consider to be a theoretically and empirically better metric for comparing classification models than the accuracy [92, 93], was reported in only three papers, while none provided a plot of the full ROC curve.

Apart from metrics related to the correctness of the classification results, we extracted three other types of LLM evaluation that are frequently used: the calibration, fairness and robustness [84]. None of the papers reported measures of the calibration or fairness of the LLMs. Only two papers tested the robustness of their models in one or more of these ways; one looked at both altering the data distribution as well as adversarial attacks and the other looked at the effect of noisy data. However, some papers tested something similar to robustness by performing ablation studies, which work by removing parts of the model to see the impact on the performance, which allows one to assess whether the model exhibits graceful degradation [94]. Ablation studies were not initially included in the mapping study but after seeing their successful implementation were added to the *ECSER-LLM* pipeline as an optional part of evaluating the robustness.

We also looked at whether justifications were provided in the papers for the selection of metrics they chose to include or exclude from the results and whether these justifications went beyond referring to previous research, which runs the risk of bad habits propagating through the field [4]. Only nine papers included a justification for their choice of metrics, with five of those exclusively referring to previous research as a justification, while four provided a more elaborate explanation.

Lastly, we looked at methodological information about the evaluation and comparison of models and found that only five papers evaluated their models over multiple data sets. Further, 13 papers compared their results to external baselines (nine of which compared to both external and their own baselines), though three papers only compared the results to their own baselines and two papers made no comparisons.

Despite almost all papers comparing their results to their own baselines or other research and usually making explicit claims about the superior performance of their models, only five papers did any statistical significance testing on their results. Some papers even claimed their results show a *significant* improvement despite not doing or not reporting on the significance testing. Further, of the five papers that did a statistical analysis of the results, only one justified the statistical test that was chosen, by referring to previous research.

Table 4 Summary of prompt reporting results in the mapping study.

Category	Total
Total prompt-based classification papers	8
Prompt approach	
Chain-of-verification	1 (12.5%)
Few-shot	2 (25.0%)
Mimic-in-the-background	1 (12.5%)
Not reported	1 (12.5%)
Zero-shot	3 (37.5%)
Final prompt	
Not reported	2 (25.0%)
Own	6 (75.0%)
Prompt approach justification	
No	5 (62.5%)
Yes	3 (37.5%)
Final prompt justification	
No	4 (50.0%)
Previous Work	1 (12.5%)
Yes	3 (37.5%)
Evaluation over multiple prompts	
No	8 (100.0%)
Temperature	
No	7 (87.5%)
Yes	1 (12.5%)

Eight of the papers used prompting to obtain results from pre-trained LLMs, which comes with additional considerations and challenges, such as what prompt approach to use (few-shot, zero-shot, etc.) and how to effectively write the text of the prompt. Table 4 shows a summary of the reported prompting information of the papers that used prompts in the mapping study.

Firstly, papers used various prompt design approaches; one used chain-of-verification, two used few-shot, one used a novel approach called mimic-in-the-background, three used zero-shot prompting, and one paper failed to report the prompt approach used or didn't use a systematic approach. However, only three papers provided justification for the chosen prompt approach.

Secondly, the final prompt, which refers to the actual prompt text used with possible placeholder values for data insertion, was only reported in six out of eight papers. Further, the authors of the six papers that provided the prompt wrote the prompts themselves without consulting established patterns, which means there is no prior evidence of the efficacy of the prompt pattern. Only four papers provided a justification for how the prompt was designed, one of which referred to previous work.

The performance of LLMs can be heavily impacted by minor changes in the prompt design, which is why Sclar et al. [70] recommend comparing multiple plausible prompts. However, none of the papers evaluated the performance of their models using multiple prompts, which leaves the vulnerability of the models to prompt variability unknown. Lastly, only one paper reported the temperature used for the LLM, despite this parameter having a potential impact on the performance of LLMs [85].

Conclusion
Evidently, the results confirm a pattern of incomplete reporting and unsubstantiated claims similar to that

found previously in the fields of ML4SE [4] and LLM4SE [20]. Metrics such as the AUC, ROC plots, the MCC, the full confusion matrix and metrics related to the calibration, fairness and robustness of LLMs go completely or almost completely unreported.

Since none of the included papers seemingly involve human participants or sensitive applications, this could be a valid reason for not evaluating the fairness, but these considerations are not discussed in any of the papers, nor do the papers explain the absence of calibration and robustness metrics.

Even when metrics *are* reported, there is often no explanation as to the why the metrics were chosen for the specific tasks. Further, many papers only use one data set and sometimes make no comparisons to external baselines. Further, many papers make unsubstantiated claims about the relative performance of their models without significant statistical backing and when significance testing is performed, little explanation is given to the choice of test and significance threshold.

For papers that use prompts for LLM classification, there is often little to no explanation for the chosen prompt approach and the design of the final prompt. Further, the prompt texts are generally not based on established patterns with empirical backing, but designed without a systematic approach. Lastly, none of the papers evaluated multiple prompts to avoid the potential sensitivity of LLMs to minor changes in wording, and the temperature of LLMs goes largely unreported as well.

Naturally, although multiple venues were considered and all papers from 2024 were considered from these venues, these results are still based on a limited set of papers. However, the results align with findings from a large base of previous research [1–4, 20], so it is not unexpected to find similar problems here, especially since there are still very few guidelines for conducting LLM research.

In conclusion, there is a clear need for easy to use and comprehensive guidelines to aid researchers in conducting LLM4SE classification research, especially for statistical significance testing and prompt engineering. We designed the *ECSER-LLM* pipeline (section 6) to take into account the mentioned common shortcomings of existing research and provide extra attention to these topics.

6 RQ2: What resource can be developed to assist researchers in conducting and reporting on LLM4SE classification research?

In this section, we first discuss the approach that was used to modify the ECSER pipeline with recommendations tailored to the conducting and reporting of LLM4SE classification research. Subsequently, we present the *ECSER-LLM* pipeline, which features a step by step guide to the implementation and results reporting of LLM-based classification research.

6.1 Research Approach

To answer the research question, we developed the *ECSER-LLM* pipeline by enhancing the existing ECSER pipeline to include the use of LLMs, taking into consideration recommendations from the fields of LLM4SE and ML4SE, as well as relevant statistical or methodological research in other fields (see section 4).

The ECSER pipeline was chosen as the basis for the development of a new LLM-specific resource, since most ECSER recommendations apply to all classification research, including LLM-based classification [4]. However, using LLMs comes with additional challenges that weren't considered during the development of the original pipeline, such as the use of prompts to guide pre-trained LLMs or the alignment of LLM behaviour with human values [79].

To enhance the ECSER pipeline with LLM-specific recommendations, we consulted various existing guidelines and surveys for using and evaluating LLMs, including Hou et al. [20]'s systematic literature review of LLM4SE, Chang et al. [84]'s survey on the evaluation of LLMs, Marvin et al. [7]'s chapter on prompt engineering and Baltes et al. [67]'s guidelines for using LLMs in SE, among many others. These higher level sources lead to more specific sources and recommendations that were included in the pipeline.

Additionally, the results of RQ1 were taken into account to ensure that common issues encountered in the mapping study received additional care in the new pipeline, including issues such as the lack of reporting for common evaluation metrics, the lack of significance testing and the lack of consistent reporting for prompt usage and development. The results of RQ1 also lead to the inclusion of ablation studies as part of the the pipeline (S8), which were overlooked during the initial literature review but were shown to be useful in assessing the graceful degradation of the models in the papers that used them [94].

In Table 5, we list each step of the original ECSER pipeline along with the new version of the step and the major changes that were made in the new *ECSER-LLM* pipeline, along with the primary source(s) that motivated these changes. Minor changes are not included in the table.

Table 5 Summary of changes and additions to the ECSER pipeline.

***ECSER* Step**	***ECSER-LLM* Step**	**Changes from *ECSER* to *ECSER-LLM***
S1. Select an evaluation method and split the data.	S1. Select an evaluation method and split the data.	Add specification of desired level of model alignment and safety [1].
S2. Train the model.	S2. Fine-tune the model	Change focus to fine-tuning LLMs. Add reporting of LLM configuration and interaction [67].
-	S3. Design Prompts.	Add prompt engineering guidelines [7, 73]. Add reporting of LLM configuration and interaction [67].
S3. Hyper-parameter tuning & validation.	S4a. Hyper-parameter tuning	Tune LLM model temperature and other hyper-parameters [67, 85].
-	S4b. Prompt comparison.	Compare similar prompts [36, 70]. Report prompt revisions and LLM interaction logs [67]
S4. Re-train with optimized parameters.	-	Merged into S4a.
S5. Test the models.	S5. Test the models.	-
S6. Report confusion matrix.	S6. Report confusion matrix.	-
S7. Report Metrics.	S7. Report Metrics.	Report Matthews Correlation Coefficient (MCC) [76–78]
-	S8. Evaluate calibration, fairness, robustness & sustainability.	New step that includes guidelines on how to evaluate the calibration, fairness, robustness and sustainability of LLMs [79, 84]. Conduct ablation studies [94].
S8. Analyse overfitting and degradation.	S9. Analyse overfitting and degradation.	-
S9. Visualize ROC.	S10. Visualize ROC.	-
S10. Apply statistical significance tests.	S11. Apply statistical significance tests.	Justify the significance test and significance threshold [95, 96].

6.2 The *ECSER-LLM* Pipeline

In this section we present the *ECSER-LLM* pipeline, a step-by-step guideline for SE researchers to conduct classification experiments using LLMs and evaluate the results, which serves to provide an answer to the research question. The pipeline encompasses the *treatment validation* step of Wieringa [41]'s design cycle, which happens after the *treatment design* step. Steps belonging to the *treatment design*, such as feature engineering and algorithms selection, are out of scope of the pipeline.

Figure 5 shows the full pipeline, consisting of 11 steps divided into two macro activities: (1) training, validation and testing and (2) results analysis. The steps are shown consecutively, but it is always possible to return to previous steps. For example, after finding optimal hyper-parameters in S4, one will likely return to S2 to retrain the models with these parameters. Further, there is a possible feedback loop between macro activities, including the *treatment design*. For example, when it is found that the model shows signs of overfitting (S9), one can reconsider the algorithms used (*treatment design*) or increase regularization (S2).

Some steps are marked as optional with a dashed border, since they depend on the *treatment design* or other considerations such as time and space constraints. In particular, when using a pre-trained LLM without fine-tuning, S2 (Fine-tuning) becomes obsolete since no training is performed, and when *not* using prompts to guide LLM outputs, S3 (Design prompts) and S4b (Prompt comparison) become obsolete since they provide guidelines for using prompts. Lastly, S4 is split into two steps: (a) hyper-parameter tuning and (b) prompt comparison. This is done to indicate that these steps can be done in parallel to find the best combination of prompt and/or hyper-parameters, but since both are marked as optional it is possible to do only one of the steps or neither.

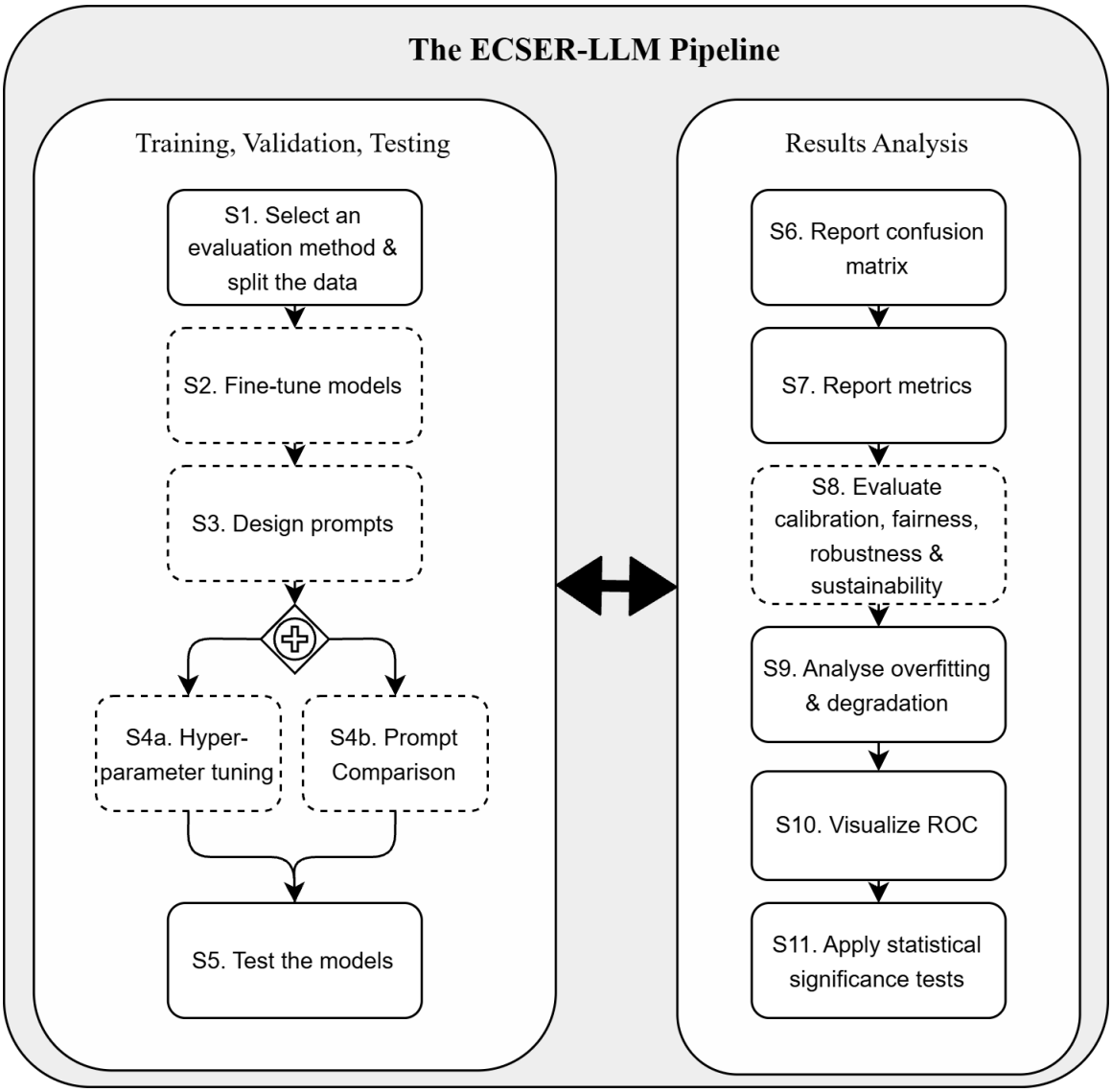

Figure 5 The *ECSER-LLM* pipeline. Dashed borders indicate optional steps. Double arrows show possible feedback loops between macro activities. The steps in S4. can be done in parallel.

6.2.1 Training, Validation, Testing

S1. Select an evaluation method and split the data.
In experimental software engineering, the treatment that is designed to answer a research question generally consists of a model that must be trained using a data set, but how well the trained model fits the training data is not a good indicator of the performance of the model on unseen data, since the training data is only a sample of the true data distribution. Therefore, some of the available data must be set aside to test whether the trained model generalises well to data that wasn't encountered in the process of training the model; and in the case of hyper-parameter tuning, some of the data must be used to evaluate the performance of models with different parameters.

When using a pre-trained LLM with no additional training, it is not needed to set aside data for training, but the data should still be split into validation and test sets. The validation set should be used to select hyper-parameters for the LLM, such as the temperature, and during the development and comparison of prompts [97]. The test set should only be used when the final prompt is developed (along with any other part of the model) to ensure the prompt design was not influenced by the test set.

Several methods exists to partition data into training, validation and test sets. The simplest is the *holdout method*, where the data is split into training, validation and test sets. The training set is used for fine-tuning (S2), the validation set is used for prompt development and hyper-parameter tuning (S3 & S4), and the test set is used for evaluating the model (S5). Generally, at least 50-70% of the data is used for training and validation, with the rest being used for testing [98, 99]. The holdout method does have a major issue: since only a portion of the training data is used for validation, the test error rate can be highly variable, leading to unreliable results [100, 101].

A more robust alternative is *resampling*, where models are fitted on multiple samples drawn from the training data. One example is *k-fold cross-validation*, where the training set is split into *k* groups (or folds) of equal size, chosen randomly from the data. The model is trained k times, once for each fold, but each time a different fold is used as the validation set, with the remaining $k - 1$ folds being used for training. The result of this process is k estimates of the test error, the average of which can be used for hyper-parameter selection [101]. A special case of k-fold cross-validation is *leave-one-out cross-validation* (LOOCV), where k is set to N, with N being the number of observations in the training set. This means that the model is trained N times on $N - 1$ observations, with a different observation being used as the validation set each time. LOOCV is more computationally expensive than k-fold cross-validation, although efficient algorithms exist for some applications that don't require training the model N times [102, 103]. In terms of performance, LOOCV has a lower bias than k-fold cross-validation, at the cost of a higher variance [102, 104].

It is also possible to use projects to partition the data using *leave-one-project-out cross-validation* (LOPO), where the training data is not split randomly, but grouped with all the data belonging to the same SE project. Each project is set aside in turn to serve as the validation set, while the other projects are used for training [105, 106]. The advantage of LOPO is that a different project is used for validation than what is used for training, which possibly enhances the generalisability of the results [107].

Which of these methods is most appropriate might depend on the available computational power and time, with methods like LOOCV requiring much more training than the holdout method; and whether the data is obtained from distinct projects. In all cases, 10-30% of the data should be set aside for testing and not touched until all parameters are tuned and prompts are finalised.

Finally, when using a pre-trained LLM, if the researcher uses open source data, there is a legitimate concern that the validation and testing sets were also used in training the LLM, leading to data leakage. LLMs have been shown to sometimes repeat training data verbatim [108]. If possible, this can be avoided by using testing

data that was released after the creation of the LLM or proprietary data.

S2. Fine-tune models.

> Optional
>
> This step does not apply when using a pre-trained LLM without fine-tuning or additional training, such as research that only uses prompts to guide the output of the LLM.

After splitting the data and deciding on the evaluation method, the training set is used to fine-tune the LLM or train a classification model on the LLM encodings. Training or fine-tuning a model entails changing the *parameters* of the model to produce outputs that are most similar to the supervised labels in the training data. The training process is usually influenced by *hyper-parameters* that are set before training and don't change during training. For example, when fine-tuning an LLM, the learning rate, batch size and epochs are examples of hyper-parameters that influence the training process, but the weights of the model are the parameters that are actually changed during training.

Since LLMs are often updated after release, the exact LLM version, configuration and experiment dates should be reported to make it easier for other researchers to replicate findings [67]. Additionally, when using fine-tuning, the fine-tuning data and weights should be shared whenever possible.

It should be noted that training a traditional classification model on LLM encodings also falls under the scope of the original ECSER pipeline [4], which deals exclusively with traditional classification models. However, additional information should still be included about the way the LLM is used, its version, experiment dates and configuration (including seed); and the additional evaluation of calibration, fairness, robustness & sustainability should still be considered when using LLM encodings.

See Parthasarathy et al. [109] for a detailed guide to fine-tuning LLMs.

S3. Design prompts.

> Optional
>
> This step only applies when using prompts to guide LLM outputs, instead of using only fine-tuning or using LLM embeddings with traditional classifiers.

Prompt engineering is an important part of interacting with LLMs, since the prompt design can have a substantial effect on the performance of the LLM. We split the process of designing a prompt into five steps, based on Marvin et al. [7].

1. *Define the goal of the prompt*: the first step of prompt engineering is to clearly define the goal of a prompt. Setting a goal for the type of output that you desire the LLM to produce helps inform the content of the prompt [7]. For example, having the goal of producing requirement classification results that resemble those of a human expert might inform the choice to add a stipulation to the prompt: "act like an expert in requirements engineering".

2. *Choose the right prompt engineering approach*: there are many approaches to prompt engineering, but the most notable are zero-shot prompting, few-shot prompting and chain-of-thought prompting. Of these, the most popular are few-shot and zero-shot in that order. When providing examples in the prompts, these

examples should be removed from the validation set before testing and refining the prompt or comparing different prompts (S4b). A relatively novel approach to prompt engineering is automatic prompt engineering (APE), which avoids the problem of manually having to write, test and refine prompts by automatically generating prompts [40, 75].

3. *Design the prompt*: writing a high quality prompt is an often overlooked aspect of prompt engineering, with most researchers writing prompts from scratch without potentially putting much thought into the exact wording, despite small changes in wording sometimes having a large effect on the performance [70]. Some resources exist that make it easier to write high quality prompts, such as White et al. [68]'s catalogue of prompt patterns. These patterns represent techniques that have been successfully used to solve common problems, put into reusable formats. For example, the Persona pattern tells the LLM to take up a certain role (e.g., "act as a software engineering researcher"). Some suitable patterns for binary classification are Question Refinement, Cognitive Verifier, Persona, Template and Context Manager [36]. Another approach is adapting prompts from related research, to more consistently compare results and provide empirical backing to the prompt design.

4. *Provide context*: providing context related to the specific domain can be a useful way to improve the performance of a prompt and avoid irrelevant responses [7]. Context can include examples like in few-shot prompting, but also information such as the desired content of the output, the format of the output or additional information about the topic.

5. *Test and refine*: it is vital to test and refine the designed prompt, with respect to the defined goal. The validation set should be used to fill in any placeholder data, after which the output of the LLM with respect to the prompt can be used to evaluate the performance and whether the LLM adheres to the desired output format. Any changes made to the prompt or insights gained from the evaluation of the prompt, as well as the full interaction logs of the prompts, should be made available, since this information can be useful for reproductions and replications [67].

Lastly, whenever an impactful prompt engineering decision is made, such as choosing to use few-shot prompting instead of another method, the choice should be motivated in the paper. This can be done by explaining the choice based on the specific nature of the task or by referencing previous research which made the same choice. If certain information, such as the prompt or interaction log, cannot be provided for privacy or confidentiality reasons, this should be clearly stated and the researcher should still provide a summary of this information [67].

S4a. Hyper-parameter tuning.

Optional

This step only applies when fine-tuning or when using other configurable parameters that influence LLM outputs, like the temperature.

Hyper-parameters are parameters that are set before training, as opposed to parameters that are optimized by training, such as the weights of an LLM. Some hyper-parameters are used during fine-tuning, such as the number of epochs, batch size and learning rate; whereas others influence the outputs of a pre-trained model, such as the temperature [85, 110]. Although hyper-parameters are not learned, they can have a substantial effect on the performance of an LLM [110]. For this reason, it is important to carefully select the hyper-parameters.

Several methods exist to choose hyper-parameters. The simplest is manual search, where the researcher

manually compares hyper-parameter values and develops an intuition for choosing the parameters, but this method makes it difficult to reproduce results [111]. Grid search automates the process by testing every combination of different subsets of the hyper-parameter space. For example, the researcher picks the subset {1e-5,1e-4,1e-3} for the learning rate and {5,10,20} for the number of epochs and then fine-tunes with all nine combinations of parameters and continues with the parameters that have the highest performance on the validation set. Since grid search brute-forces all combinations of the chosen subsets, it can become very computationally expensive [112].

A more efficient alternative to grid search is *random search*. Random search works by drawing multiple samples of the hyper-parameters from a discrete or continuous distribution of the hyper-parameter space and fine-tuning using these samples [111]. Another advantage of random search is that it can be stopped after any amount of trials, instead of only after the full grid has been explored. Lastly, some other less common methods for tuning are genetic algorithms [113] and DODGE [114].

After the optimal hyper-parameters are determined, the model is retrained using the full training data, including any data that was set aside for validation. This can also be done using nested cross-validation, where the final model is trained at the same time as tuning the hyper-parameters.

S4b. Prompt comparison.

Optional

This step only applies when using prompts to guide LLM outputs, instead of using only fine-tuning or using LLM embeddings with traditional classifiers.

The specific wording of a prompt can have a large impact on the performance of an LLM, even when the prompt retains its semantic content and intent [70, 71, 115]. This includes seemingly innocuous changes such as capitalisation. Thus, relying on a single human-written prompt could lead to unreliable results. One way to mitigate this problem is by writing multiple prompts that use different prompt patterns (such as Question Refinement, Cognitive Verifier and Persona) and compare the results using the validation set [36]. The comparison of different prompts can be done efficiently using PromptEval, a technique for estimating the performance of a large set of prompts, allowing one to pick the best prompt or evaluate the average performance over multiple prompts [72]. However, these techniques require manual prompt writing, which still leaves the risk of specific word choices or syntax affecting the performance of the prompts.

Alternatively, Sclar et al. [70] developed the algorithm FormatSpread, which is able to generate and evaluate a large set of plausible prompt formats, reporting the range of performances over these prompt formats. Some advantages of this technique include that it provides insight into the sensitivity of the original prompt design and that it gives a lower bound on the model's performance. A similar technique is Mixture of Formats (MOF), where the style of each few-shot example is modified by asking the LLM to change the style, leading to a mix of different styles being used, reinforcing model understanding [116].

The approach to prompt comparison that was used should be reported and justified in the paper, along with prompt revisions and interaction logs [67]. The result of prompt comparison could be a single prompt which has shown good performance, which will then be used for the final evaluation; or a set of plausible prompts, in which case the evaluation will be done over this set of prompts. If no prompt comparison is done, the reason should be explained in the paper.

S5. Test the models.

Finally, after fine-tuning and determining the optimal hyper-parameters and/or prompts, the performance of the LLM is evaluated on the test set. It is recommended to include multiple data sets into the test set, to assess the generalisability of the model and to run statistical tests across these multiple data sets (S11) [4].

6.2.2 Results Analysis

S6. Report the Confusion Matrix.

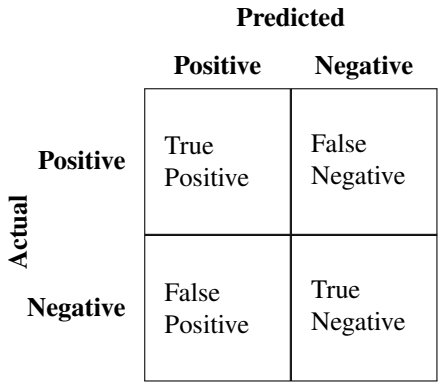

Figure 6 Generic confusion Matrix. All values add up to the number of data points.

While it is common to select a few metrics appropriate to a given task and report them, we recommend reporting the full confusion matrix so readers can get a more complete picture of the results, and be able to calculate many different metrics, even when they aren't explicitly reported (see S7). The confusion matrix consists of the number of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) classification results and are reported as whole numbers. Often, the sums of the rows or columns are also included, since adding up the numbers in the columns results in the total number of data points that were classified as positive or negative, while the row sums represent the number of data points that were actually positive or negative. Figure 6 shows a generic confusion matrix.

S7. Report Metrics.

Many metrics can be calculated from the confusion matrix, including the *Precision*, *Recall*, *Sensitivity*, *Accuracy*, F_1-score, F_β-score and Matthews correlation coefficient (MCC). Table 6 shows the calculation

Table 6 Calculation of common metrics for classifier performance.

Metric	Formula
Precision	$TP/(TP+FP)$
Recall (TPR)	$TP/(TP+FN)$
Specificity (TNR)	$TN/(TN+FP)$
Accuracy	$(TP+TN)/(TP+TN+FP+FN)$
F_1-score	$2 \cdot (\text{Precision} \cdot \text{Recall})/(\text{Precision} + \text{Recall})$
F_β-score	$(1+\beta^2)\text{Recall})/((\beta^2 \cdot \text{Precision}) + \text{Recall})$
MCC	$(TP \cdot TN - FP \cdot FN)/\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}$

of these metrics using the values from the confusion matrix. Which metrics are reported generally depends on the domain and the specific application. For example, the precision gives the percentage of positive classifications that are correct, so a high precision is desirable for applications where the consequences of false positives are high. On the other hand, the recall gives the ratio of actual positives that were correctly classified, so a high recall is desirable when the consequences of false negatives are high. In most applications, a good balance of precision and recall is preferred, which can be measured by the F_1-score, the harmonic mean of recall and precision. The F_β-score changes the weights of precision and recall in case one of the metrics is more important for the given task.

The specificity or true negative rate (TNR) is the ratio of actual negatives that are correctly classified as negative. The accuracy is the overall ratio of correct classifications and can be a useful metric when the classes are balanced, i.e. there is a similar amount of data points for positive and negative classes. If the classes are not balanced, the accuracy can be misleading [117, 118].

The F_1-score is commonly used in the field of LLM4SE, but it has several drawbacks: (1) the F_1-score varies when the classes are swapped (i.e., positive is renamed to negative and vice-versa) and (2) the F_1-score is independent of the number of data points correctly classified as negative (TN) [76, 119]. The Matthews correlation coefficient (MCC) is recommended by many researchers as a more balanced alternative to the F_1-score, since it balances all four categories of the confusion matrix and does so proportionally to the number of positive and negative data points [76–78]. The MCC ranges between -1 and 1, with -1 representing a total disagreement between the predicted and actual results, 0 representing no correlation and $+1$ representing a perfect agreement.

Why specific metrics were included or omitted from the results should be justified in the text [120]. Ideally, this can be done by explaining why the chosen metrics are mathematically or empirically appropriate for the given task. Otherwise, the researcher should refer to similar previous research that used these metrics. If certain metrics were omitted, the results should report enough information to allow the reader to calculate these metrics themselves, for example by reporting the full confusion matrix and providing the raw results as supplementary material.

The main metrics for comparing LLM classifications are obtained by evaluating on the test set. However, the researcher should also share the metrics that were obtained for the training and validation sets during fine-tuning, hyper-parameter tuning or prompt development/comparison, to give readers more insight into the development and aid replicability. Providing the mean and standard-deviation of the metrics for cross-validation can also provide insight into the performance. Additionally, when using a specific output format such as a JSON object, a distinction should be made between the performance of (1) the test cases where the format constraints were adhered to and (2) all test cases, regardless of format satisfaction [81].

S8. Evaluate calibration, fairness, robustness & sustainability.

Optional

Each of the aspects of LLM evaluation contained in this section is optional; but we recommend considering these aspects when dealing with human participants or users; or sensitive data and applications.

The previous step (S7) considered how to evaluate the correctness of classification results. However, the correctness does not tell the full story: an accurate model does not necessarily work equally well for all groups of users, nor does the accuracy take into account the resistance of the model to noisy data, changes

in distribution, adversarial prompts or the calibration of predicted probabilities [79, 84, 91]. In this step, we briefly discuss how (and why) to evaluate the calibration, fairness, robustness & sustainability of LLM classifiers.

Calibration
The calibration of an LLM refers to how well the confidence of the model aligns with the actual probability of getting a correct classification [1]. When the confidence is well calibrated, we can get an indication when model outputs are likely to be incorrect or whether we can trust the outputs. The confidence of a prediction can be estimated in several ways, as discussed in [121]. The most accurate estimate of the confidence can be obtained by calculating the conditional probability of the label given the prompt [122]. However, the probabilities of tokens can sometimes not be obtained for closed-source models, in which case the confidence can be estimated by sampling multiple responses and measuring the consistency or by prompting the model to give the confidence of the labels in the generated answer, which can be a numerical probability or a linguistic response (e.g., "Almost certain" or "Unlikely").

A common metric for measuring the calibration is the **expected calibration error (ECE)**, which is obtained by dividing predictions into M bins of equal size and calculating the average difference between the accuracy and confidence over all M bins [91]. For high-risk applications, the **maximum calibration error (MCE)** can be used to get the worst-case difference between the confidence and accuracy. See Nixon et al. [90] for a discussion of possible pitfalls when using the ECE and alternative metrics.

Fairness
The fairness of an LLM refers to the equal treatment across different groups of people [84]. Taking the fairness into account helps ensure the alignment of model behaviour to human intentions [123]. Liu et al. list four types of unfairness: injustice, stereotype bias, preference bias and disparate performance. Injustice arises when similar individual are not treated similarly by the model. For example, when two indistinguishable job applications are entered into a prompt, with the only difference being irrelevant group attributes (such as gender), we expect the LLM to classify both applications the same way. Stereotype bias refers to an LLM's prejudiced or misleading expectation about members of particular social groups, such as a presumed academic ability based on race or gender. Preference bias refers to a lack of neutrality concerning subjective topics such as politics, scientific interpretations, societal matters or product preference. Lastly, disparate performance refers to a difference in performance across groups of users. For example, LLMs often perform worse on languages other than English, which means that applications using LLMs might perform worse in certain cultures and societies [124].

To evaluate the fairness, Wang et al. [125] list two metrics:

- **Demographic Parity Difference (DPD)**: the DPD measures the difference between the probability of positive predictions given the presence of a sensitive attribute and the probability given the absence of that attribute. A high DPD means that there is a large difference in the positive predictions between groups. Note that the DPD does not consider the ground truth labels.
- **Equalized Odds Difference (EOD)**: the EOD takes the maximum difference between the true positive rate (recall) and false positive rate given the sensitive attribute. This means that a low EOD requires both classes to have similar prediction errors. By comparing the true positive rate and false positive rates, it also takes into account the ground truth labels.

Robustness
The robustness of an LLM refers to how well it deals with difficult or unexpected inputs [84]. There are multiple scenarios in which an LLM can encounter challenging inputs, including through changes in the data distribution, noise in the data, adversarial prompts and poisoning of the training data [79, 84]. These scenarios

can be caused by malicious actors (such as changing labels to poison the training data), user error (such as typos in the prompt) or other reasons (such as a change in user demographic).

Chang et al. [84] list two common robustness metrics:

- **Attack Success Rate (ASR)**: the ASR evaluates the success of adversarial attacks by calculating the rate with which the attack successfully produces adversarial examples [126].
- **Performance Drop Rate (PDR)**: the PDR evaluates the robustness of prompts by quantifying the drop in performance resulting from a prompt attack [9].

A different aspect of robustness is how well models perform when some parts of it are deliberately removed. Ablation studies work by removing parts of a model, such as layers in a transformer or input features, and seeing whether the performance changes [94]. These studies show whether the model exhibits a graceful degradation of the performance when parts are removed. In the field of LLM4SE, ablation studies are commonly used to find the performance benefit of each component in models that consist of multiple interacting components, thereby evaluating redundancy [127, 128].

Sustainability
The use of LLMs can come with a significant cost of time, space and energy compared to traditional ML models. Especially the training phase of LLMs can have a large energy footprint; but the increased integration of LLMs into various tools such as Google search could also have a major impact on energy consumption over time [129, 130]. Researchers in LLM4SE can reduce energy expenditure primarily by reducing LLM inferences, which can be done by restricting the number of queries, using smaller data-sets for fine-tuning or using smaller models [67]. By using more efficient algorithms, reducing LLM inferences does not necessarily reduce the accuracy of the results, although there can be a trade-off between accuracy and energy expenditure [130]. In the results analysis, researchers should report what steps (if any) were taken to reduce energy consumption and justify why LLMs were used instead of less costly alternatives.

S9. Analyse overfitting & degradation.
One of the fundamental problems in machine learning is the possible disconnect between the in-sample performance of a model on its training or validation sets and the out-of-sample performance on the unseen test set [131]. A high training accuracy does not guarantee a high test accuracy; in fact, a model with a lower training accuracy can outperform a model with a higher training accuracy when applied to the test set. This problem is known as overfitting and can occur when a model learns from noise in the data, has limited training data or has more parameters than needed. For example, if a limited set of examples is used for fine-tuning an LLM, the performance on the fine-tuned data could be great, but not generalize well to the actual real-life task that it aims to solve.

Overfitting can be quantified by measuring the difference between the performance on the test set and the train set using the metrics in S7. Let M be any chosen metric, then $overfitting = M_{test} - M_{train}$. If we use multiple data sets for testing, we can instead calculate the average overfitting, where $Test$ is the test set:

$$\textit{average overfitting} = \frac{1}{|Test|} \sum_{t \in Test} (M_{test} - M_{train}) \tag{1}$$

Apart from the metrics discussed in S7, such as accuracy, other metrics can also be used to quantify overfitting. This includes the metrics of zero-one loss for binary classifiers and mean squared error (MSE) for when class outputs contain scores or probabilities [4]. If a pre-trained LLM is used with prompting, it might be difficult or unnecessary to analyse the overfitting. However, the validation set that was used for developing

and selecting prompts or hyper-parameters can still lead to degradation, where the performance on the test set is lower than the performance on the validation set.

Degradation compares the performance of the test set with the performance of the validation set, using the metrics of S7. Its calculation is similar to overfitting: given performance metric M, we calculate $degradation = M_{test} - M_{validation}$. Equivalently, if multiple test sets are used or multiple validation sets in the case of k-fold cross-validation, we recommend calculating the average degradation, as in (1). In the event that both k-fold cross-validation and multiple test sets are used, then the distributions of the performance metrics for the validation and test sets can be statistically compared to see if there is a significance difference. For example, if the data is normally distributed, the independent samples T-Test can be used, whereas the Mann-Whitney's U test can be used as a non-parametric alternative. The original *ECSER* pipeline contains examples of how to apply these tests [4].

S10. Visualize ROC.
Metrics such as the accuracy and F-score depend heavily on the arbitrary choice of the decision threshold (i.e., the confidence value above which a data point is classified as positive). The receiver operating characteristic curve (ROC) gives a visual description of this effect by showing the trade-off between the true positive rate (recall) and the true negative rate (specificity) across threshold values [132]. Figure 7 shows an example of an ROC plot with true positive rate on the y axis and false positive rate on the x axis. The diagonal represents random chance. Visually, the closer a classifier is towards the top-left corner, the better the classification performance.

For discrete classifiers, the outputs will not be confidence values, but only class labels. In this case, the resulting true positive and false positive rate can be plotted as a single point in the ROC space [133]. Multiple discrete classifiers can thus be visually compared by plotting each result as a point in the ROC plot, or as a cluster of points when multiple data sets are used for each classifier.

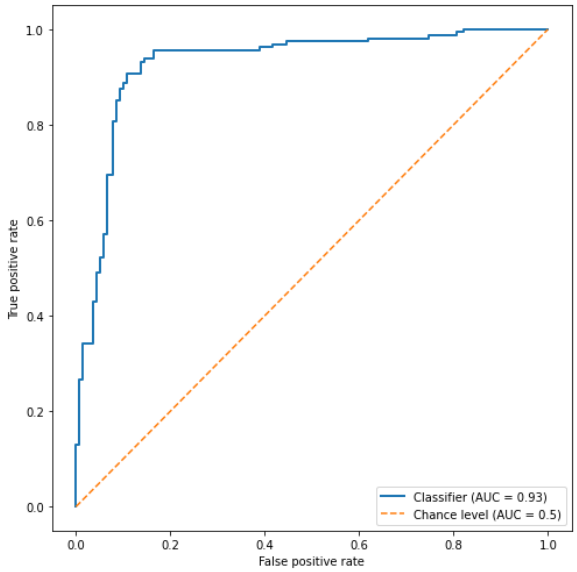

Figure 7 Example of an ROC plot comparing a classifier model (blue) to random chance (orange).

The area under the ROC curve (AUC) provides a summary of the ROC curve by taking the average across all

threshold values. Since the AUC of a random classifier is 0.5, the realistic range of AUC values is between 0.5 and 1 [133]. It has been suggested that the AUC provides a better basis for comparing classifiers than the accuracy, since it is independent of the decision threshold [92, 93].

S11. Apply statistical significance tests.
It is not enough to see a difference in performance metrics between two classification models to make the claim that one model is better than the other. There are many reasons why the performance of two models can differ, including randomness in the training data, evaluation data or training process [120]. Statistical significance testing assesses whether the obtained results are incompatible with a given null-hypothesis (e.g., "the two models have equal performance"). The smaller the p-value of a test, the less the results agree with the null-hypothesis, given that the assumptions of the chosen test are met [134]. It is general practice that a p-value below the significance threshold of 0.05 represent a statistically significant result. However, the significance threshold that is used should be justified based on the research topic, the number of hypotheses tested, the study design and the cost of errors [95]. Further, statistical significance is not a license for claiming a scientific finding, since the results depend on the quality of the research itself. Instead, conclusions should be based on non-automated informed judgement and replicability [96].

Many tests exist to evaluate the statistical significance between two or more classifiers, each with their own underlying assumptions. For example, parametric tests like the paired samples T-Test assume that the data is normally distributed, whereas non-parametric tests like the Wilcoxon's Signed-Rank test do not have this assumption. In general, LLM4SE researchers should consider non-parametric tests, since classifier results are generally not normally distributed [135]. Readers should refer to the original ECSER pipeline for a detailed discussion of how to perform statistical testing on single or multiple datasets, noting that the Wilcoxon Signed-Rank and the Friedman plus Nemenyi's post-hoc tests are recommended tests by the authors [4]. Another useful resource for choosing the significance test is Rainio et al. [136]'s flowchart, which considers the task, evaluation metric, the number of test sets and the number of models that are compared. In all cases, researchers should justify the chosen statistical test based on the assumptions it makes and its limitations.

7 RQ3: How applicable is the proposed pipeline in assisting LLM4SE classification research?

In this section, we first discuss the approach to evaluating the applicability of the *ECSER-LLM* pipeline by way of a replication study. Subsequently, we present the results of the replication study and use these results to analyse the applicability and the added benefit of using the pipeline.

7.1 Research Approach

In order for the *ECSER-LLM* pipeline to be considered a suitable treatment for the problems investigated in RQ1 (Section 5), the results obtained by applying the pipeline should avoid the shortcomings of existing LLM4SE research. For this reason, the choice was made to conduct a replication of an existing LLM4SE classification study by following the steps of the *ECSER-LLM* pipeline, in order to evaluate whether applying the pipeline is beneficial for enhancing the original findings. In conducting the replication study, we followed Carver [55]'s reporting guidelines.

Studies were considered for replication based on the following criteria:

1. The authors provide a ready-to-use replication package.

2. The study is recent (past five years) and represents current practices in LLM4SE well.
3. The study is peer-reviewed and published in a reputable journal.
4. The study does not already follow all the proposed guidelines, so that the added benefit of the pipeline can be evaluated.

These criteria were designed to select a study that is high-quality and easy to replicate, but does not follow all of the best practices in the field, since the purpose of the *ECSER-LLM* pipeline is to make it easier for researchers to follow these best practices.

We chose to replicate Hora [137]'s study on predicting Python test results without execution, since the paper included a repository with the data and LLM outputs, was published recently (2024), was published in a reputable journal (FSE) and did not follow all the proposed guidelines. Additionally, the paper uses prompts to guide predictions, making it well suited to test the prompt engineering recommendations in the pipeline, which represents one of the major contributions of the *ECSER-LLM* pipeline compared to the original ECSER pipeline. In the next section, we describe the paper in more detail and present and discuss the results of each step of the pipeline.

7.2 Case Study: Predicting Test Results without Execution

For our case study we consider the topic of predicting test results without execution, a novel area of research proposed by Hora [137], which aims to evaluate whether LLMs can understand code execution sufficiently to predict the results of running tests without the challenges associated with executing the code. The original study presents this problem as a binary classification task where the model predicts that executing the test either causes an exception ('fail') or runs without exceptions ('pass').

The original study investigated this problem by evaluating the performance of GPT-4 in predicting Python test results in Python version 3.10. The authors constructed a data set of Python test cases by selecting test cases from five different libraries of the Python Standard Library: ast, calender, csv, gzip and string. From each of these five libraries, one test was selected that was considered to be relatively simple and one that was considered more complex, for a total of 10 unique test cases. Then, each test case was manually modified to create 10 passing tests and 10 failing tests, resulting in a data set of 200 test cases (100 passing and 100 failing), with 100 simple and 100 complex cases and 40 cases for each test suite (ast, calender, csv, gzip and string).

For our replication, we decided against using GPT-4 for predicting the results, since the usage of a closed-source model such as GPT-4 comes with several downsides, including cost, restricted access to model parameters and no guarantee of long-time availability. These downsides could hinder the replicability of the results [67]. Instead, we use the open-source LLama 3.1 with eight billion parameters and instruction tuning, since Llama 3 has shown competitive performance with GPT-4 on text classification [138]. However, we do not use the largest available model of 70 billion parameters, because of the available computational resources. Since we intentionally alter the procedure of the original study by applying the *ECSER-LLM* pipeline, our replication can be considered a non-exact or conceptual replication [49].

By applying the *ECSER-LLM* pipeline, we make several contributions to the literature:

1. We increase the replicability of the research by using an open-source LLM as well as providing all our code, prompts and the processed data set on our GitHub page[2].

[2] `https://github.com/rubenvdz/Master-Thesis-Enhancing-the-ECSER-pipeline`

2. We perform a more systematic approach to prompt engineering, including a comparison of different prompts patterns.
3. We provide more insight into the results by providing additional performance metrics and analysing the calibration, robustness and sustainability.
4. We conduct statistical tests that show that we cannot corroborate some of the original findings of the paper.

7.2.1 Training, Validation, Testing

S1. Select an evaluation method and split the data.
In order to get an unbiased estimate of the generalisability of the results, we have to avoid testing the model on the same data that was used for prompt engineering and prompt comparison. Since we do not perform hyper-parameter tuning or fine-tuning, we choose to use the holdout method to split the data evenly into a validation and test set, consisting of 100 samples each. These samples are chosen to provide an even split in terms of test suite, simple and complex test cases and label (pass or fail). This means that the validation and test set both consist of 50 passing and 50 failing tests, 50 simple and complex cases and 20 test cases per suite. The validation set is used for S3-S4, while the test set is exclusively used for S5-S11. The original study did not split the data into validation and test sets, which leaves open the possibility of the test data having influenced the design of their prompt.

S2. Fine-tune models.
Since the aim is to evaluate the performance of a pre-trained LLM on test result prediction, we do not perform S2 of the pipeline.

S3. Design prompts.
The prompt of the original study, although inspired by related research [139], was not accompanied by an explanation of how the prompt was designed, nor whether alternative approaches or prompts were considered. Instead, using the prompt of the original study as a blueprint, we follow the five steps of prompt design as follows:

1. *Define the goal of the prompt*: the goal of the prompt is to make the model understand that it is asked to provide a binary judgement of whether a given test will pass or fail if it were run in Python version 3.10.

2. *Choose the right prompt engineering approach*: for our approach, we selected the two most widely used prompt engineering approaches, zero-shot and few-shot, which we compare in S4b. Since the original study used a zero-shot approach, we based our zero-shot prompt on theirs, with the added output context mentioned in step 4 (see Figure 8). For the few-shot prompt, we selected four test cases from different suites, of which two were failing and two passing tests, and providing these tests and the desired outputs as part of the prompt. The desired outputs took the form of an explanation and a label as defined in step 4. The explanations were taken from the outputs of GPT-4 provided in the original study, making sure that these explanations were taken from correctly classified test cases, which were then shortened to prevent the context length from exceeding the allowed limit of 2048 tokens. The few-shot examples were removed from the validation set before testing.

3. *Design the prompt*: the prompt of the original study was not written using any established patterns with empirical backing. To enhance the text of the prompt, we selected three prompt patterns from White et al. [68]'s catalogue: the Persona, Cognitive Verifier and Question Refinement patterns. These patterns were chosen since they were the three highest performing prompt patterns in Ronanki et al. [36]'s comparison

of patterns for requirements classification. Each prompt pattern provides contextual statements that should be included in the prompt. For example, the Persona pattern has the contextual statements "Act as persona X" and "Provide outputs that persona X would create". The Question Refinement pattern asks the LLM to generate additional questions for it to answer before providing the label. The Question Refinement pattern tells the LLM to change the original question if needed and answer that question instead. For this last pattern we created two versions: one that asks LLama 3.1 to generate a better question for each instance and one that uses a pre-generated refined question that was generated by the more powerful GPT 5. All of these prompts, including the patternless zero-shot and few-shot prompts, can be found on our GitHub page and are compared in S4b.

4. *Provide context*: we provided context to our prompts by specifying the desired format of the LLM response: an explanation followed by the label "PASS" or "FAIL" on a separate line, allowing us to easily extract the predicted label from the generated output. Figure 8 shows an example of how we changed the original (zero-shot) prompt to add context for our zero-shot prompt. In order to ensure that the LLM followed this format and to avoid responses that couldn't be automatically parsed, we made use of the GBNF grammar; a formal grammar that constrains the output, designed for use with the llama.cpp library. In this grammar, we also restricted the length of the explanation to 2000 characters.

Original Prompt

Consider the following test of the Python Standard Library, version 3.10:
⟨test_case⟩
Your job is to figure out whether this test will pass or fail. If it fails, provide the rationale.

(a) Prompt from the original study [137].

Our Prompt (Zero-shot)

Consider the following test of the Python Standard Library, version 3.10:
⟨test_case⟩
Your job is to figure out whether this test will PASS or FAIL. Provide the rationale before giving an answer.

Answer in this exact format:
Explanation: ⟨free text⟩
Label: ⟨PASS | FAIL⟩

(b) Our zero-shot prompt.

Figure 8 The original prompt next to our zero-shot prompt with added context.

5. *Test and refine*: we tested all of our prompts on smaller partitions of the validation set to ensure that the responses resembled our expectations with regards to the content and structure, before using the full validation set for comparison. These smaller tests revealed that for some prompts, without a specified grammar, up to 25% of responses would not follow the specified format, which lead to our implementing a strict grammar for the responses in step 4, solving the issue of low format adherence. Further, our original few-shot prompt contained examples with explanations that exceeded LLama's context limits, which lead us to shorten the explanations to fit within the limits. Other common issues with the few-shot prompt were that the responses would sometimes repeat the few-shot examples instead of classifying the actual test case, which was solved by specifying explicitly that the LLM should not repeat the examples and only answer the actual test case. Lastly, for transparency and reproducibility, we included all interaction logs (i.e., prompts and responses) in our results.

Compared to the prompt design approach of the original paper, which was not explained apart from a ref-

erence to related research, we have followed a systematic approach to prompt design based on the available recommendations and empirical evidence. Further, our prompts have additional contextual information to guide the output into specific formats, using prompt patterns and grammars. The most important change is the development of multiple prompts, which allows us to compare the relative performance of these prompts, avoiding the unpredictability of using a single prompt [71].

S4a. Hyper-parameter tuning.
Since we used a pre-trained model, we do not have control over training-related parameters. For parameters that guide the generation process, we selected specific values to ensure consistent results, since we are more concerned with replicability than performance. The temperature could be tuned to find the value with the best performance, but we chose to set it to 0 in order to remove any randomness in favour of replicability. We set the context size of the model to 2048 tokens to ensure our few-shot prompt did not exceed the context size. All other parameters were kept at their default values[3]. The original study did not report the temperature or the value of any other parameter, which means we do not know how the temperature affected the randomness of their result or which parameter values to use if we wish to conduct an exact replication.

S4b. Prompt comparison.
Instead of designing a single prompt, we designed a total of six prompts. The first two, which we refer to as the zero-shot and few-shot prompts were made to be very similar to the prompt of the original paper, only with added context for both and few-shot examples for the latter prompt. The other four are modified versions of our zero-shot prompt that make use of established prompt patterns. This includes the Persona, Cognitive Verifier and two versions of the Question Refinement pattern. Each of these prompts were evaluated on the validation set in order to select a single prompt for the final evaluation (S5).

Table 7 shows the performance metrics for each prompt (calculated from the confusion matrices in Table 8). Note that for these results, we consider "fail" to be the positive class, as in the original paper, since we want test cases to fail in case there is a problem. Hence, a true positive (TP) represents a test case that is correctly classified as failing, whereas a true negative (TN) is a test that is correctly classified as passing. These results show that our Cognitive Verifier and Persona prompts perform the best on the validation set, with the Cognitive Verifier prompt having the highest precision (0.545), accuracy (0.550) and MCC (0.101); whereas the Persona prompt has the highest recall (0.740), F_1-score (0.607), F_2-score (0.680) and AUC (0.577). The few-shot prompt has the highest specificity (0.729), but this comes at the cost of a very low recall (0.255).

Table 7 Performance metrics for each prompt on the validation set. Highest values are in bold.

Prompt	Precision	Recall	Specificity	Accuracy	F1	F2	MCC	AUC
Zeroshot	0.531	0.680	0.400	0.540	0.596	0.644	0.083	0.549
Fewshot	0.480	0.255	**0.729**	0.495	0.333	0.282	-0.018	0.472
Cognitive Verifier	**0.545**	0.600	0.500	**0.550**	0.571	0.588	**0.101**	0.541
Persona	0.514	**0.740**	0.300	0.520	**0.607**	**0.680**	0.045	**0.577**
Q. Refinement (Llama)	0.508	0.620	0.400	0.510	0.559	0.594	0.021	0.458
Q. Refinement (GPT)	0.466	0.540	0.380	0.460	0.500	0.523	-0.081	0.511

Based on these results, we selected the Persona prompt for the final evaluation for two reasons. Firstly, it has the highest F_1-score, which signifies a good balance between precision and recall. Secondly, it has the

[3]See `https://llama-cpp-python.readthedocs.io/en/latest/api-reference/#llama_cpp.Llama.create_chat_completion`

highest recall and F_2-score, which we weigh more heavily than the precision, since the primary purpose of running test cases is to find code that fails, so that issues can be resolved, instead of finding code that runs as expected. Figure 9 shows the full Persona prompt as it is used in the rest of the pipeline. It should be noted that these results are specific to our implementations of the prompt patterns, the general wording of the prompt and the data set, which means we cannot make claims about which prompt pattern is better in the general case. For this reason, we recommend all researchers to conduct their own comparisons to find the best prompt for their case.

Since the original study used a single prompt, without comparing it to alternative prompt designs, and also did not use established prompt patterns, we have no way of knowing whether their results would have been different if they used other prompting approaches or if the Persona pattern would have also had the highest performance in their case.

Persona Prompt

Act as a Python expert and consider the following test of the Python Standard Library, version 3.10:
⟨test_case⟩
Your job is to figure out whether this test will PASS or FAIL. Provide the rationale before giving an answer.
Provide outputs that a Python expert would create.

Answer in this exact format:
Explanation: ⟨free text⟩
Label: ⟨PASS | FAIL⟩

Figure 9 The Persona prompt, which was selected from the prompt comparison and will be used in the rest of the pipeline. Contextual statements unique to the Persona prompt pattern are coloured red for emphasis.

S5. Test the models.
After developing multiple prompts and comparing the results on the validation set, we determined that the Persona prompt has the best performance. Next, we use this prompt with the test set to obtain the final results for S6-S11.

7.2.2 Results Analysis

S6. Report the Confusion Matrix.
Table 8 shows the confusion matrices obtained on the validation and test sets. These confusion matrices were used to calculate the metrics for each prompt on the validation set in S4b, and are used to calculate the metrics on the test set for the Persona prompt in S7. The benefit of providing the confusion matrix along with the calculated metrics is that if the reader is interested in any metrics that we did not consider, they have the ability to calculate these metrics themselves using the confusion matrix.

Table 8 Confusion matrices for validation and test sets. Persona prompt is bold for easier comparison between the validation and test set.

Data set	Prompt	TP	FP	FN	TN
Validation	Zeroshot	34	30	16	20
	Fewshot	12	13	35	35
	Cognitive Verifier	30	25	20	25
	Persona	**37**	**35**	**13**	**15**
	Question Refinement (Llama)	31	30	19	20
	Question Refinement (GPT)	27	31	23	19
Test	**Persona**	**40**	**35**	**10**	**15**

S7. Report Metrics.

Using the confusion matrix on the test set (S6), we calculated the performance metrics presented in Table 9. Overall, the results show that the model has a much higher recall (0.800) than precision (0.533), suggesting that while it finds most failing test cases, in the process it also misclassifies many passing test cases as failing test cases. This result is unsurprising, since we specifically favoured recall over precision when selecting the prompt in S4b, as the primary goal of running tests is to find incorrect code. As a result of the high difference in recall and precision, the model has a high F_2-score (0.727) and decent F_1-score (0.640), but a relatively low accuracy (0.550) and specificity (0.300). The MCC (0.115) shows that there is some level of agreement between the predicted and actual labels, but far from a perfect prediction (which would be 1).

Apart from the results across all test cases, we have included the results for simple and complex test cases and the results for each test suite (ast, calendar, csv, gzip and string), as in the original paper. These results show that the simple test cases have a higher or equal performance than the complex test cases on every metric. Further, there are considerably differences in the performance metrics depending on the test suite. However, these observations are not enough to claim that the model is more capable of classifying simple test cases than complex test cases or actually performs better on some test suites *in general*. We have to consider the possibility that these differences are coincidental, especially given the limited number of observations (50 each for simple and complex test cases, and 20 per test suite). Hence, we analyse the statistical significance of these results in S11 before drawing conclusions, a step that was not performed in the original study.

Table 9 Performance metrics across different parts of the test set.

Tests	Precision	Recall	Specificity	Accuracy	F1	F2	MCC	AUC
All	0.533	0.800	0.300	0.550	0.640	0.727	0.115	0.573
Simple	0.556	0.800	0.360	0.580	0.656	0.735	0.178	0.630
Complex	0.513	0.800	0.240	0.520	0.625	0.719	0.048	0.507
ast	0.643	0.900	0.500	0.700	0.750	0.833	0.436	0.640
calendar	0.563	0.900	0.300	0.600	0.692	0.804	0.250	0.580
csv	0.471	0.800	0.100	0.450	0.593	0.702	-0.140	0.360
gzip	0.500	0.800	0.200	0.500	0.615	0.714	0.000	0.650
string	0.500	0.600	0.400	0.500	0.545	0.577	0.000	0.650

Compared to the original paper, we have reported many common metrics that were not provided by the original authors, including the specificity, F_1-score, F_2-score, MCC and AUC. However, since they included the confusion matrix for the full test set, we were able to calculate that their model (GPT-4) had an F_1-score of 0.789, an F_2-score of 0.740 and an MCC of 0.633. These metrics are higher than ours, which is unsurprising

given that GPT-4 is a much larger model than our Llama model of 8 billion parameters. However, their results are much harder to replicate, since they used a closed-source model, did not explain how they designed their prompt and did not provide all the code they used to run the experiments and analyse the results. It is possible that with a more systematic approach to prompt design and prompt comparison, their model would have performed even better, given that our zero-shot prompt, which is very similar to their prompt, was not our highest performing prompt.

S8. Evaluate calibration, fairness, robustness & sustainability.
The evaluation of the correctness of the results does not give a complete picture of the trustworthiness of said results. In this step, we consider the calibration, fairness, robustness & sustainability of the results. These aspects of the evaluation provide additional insights into the real-world usefulness of the model.

Calibration
In order to calculate the calibration of the model, the confidence of each prediction has to be estimated. We used the key token probability to estimate the confidence, since it is considered one of the best methods for confidence elicitation [122]. This method works by focusing on the probability of result-specific tokens, which in our case means taking the conditional probability of the final label ("PASS" or "FAIL"), which was generated after the explanation. By not calculating the probability of the entire sequence, we remove the influence of the length of the explanation and any irrelevant tokens. We were able to easily obtain these probabilities, since we have used an open-source model. If we had used a closed-source model as in the original study, we would have had to use less reliable methods for estimating the confidence.

We calculated the expected calibration error (ECE) with 10 bins to measure the difference between the estimated confidences and the actual accuracy of the predictions, resulting in an ECE of 0.249. This means that we can expect a 25% difference between the average confidence and the actual accuracy, showing that the model is poorly calibrated. This result is unsurprising, since LLMs have been shown to be overconfident in their predictions, and it has been demonstrated that instruction tuning worsens the calibration [140]. However, the fact that the model is poorly calibrated means that we cannot rely on the model's confidence to determine whether we should trust its predictions. I.e., if the model is very confident that a given test case will pass, there could still be a high probability of it failing.

Since the original study did not evaluate the calibration of the model, we don't know how much we can trust the explanations provided by their model, even when the explanations express a high level of certainty of a particular test case passing or failing.

Fairness
In the pipeline, we consider four types of unfairness: injustice, stereotype bias, preference bias and disparate performance [79]. Since we do not use human participants, sensitive data, or subjective data; we do not perform this step of the pipeline.

Robustness
There are several ways to evaluate the robustness of an LLM when faced with challenging inputs, such as the presence of noise in the training data or changes in the data distribution [84]. Since we do not perform any training, we focus on another type of robustness: the robustness of the LLM to adversarial prompts. We can evaluate the robustness to adversarial prompts by testing whether the results would have been different if the prompt had been altered in some way, whether maliciously or due to user error, by using a prompt attack.

To exemplify how to evaluate the prompt robustness, we used the TextAttack framework to generate two adversarial prompts, one by making character-level changes to our original prompt and one by making word-level changes [141]. The character-level changes for the first prompt attack were made using the "charswap"

augmenter, which changes the text of the prompt by substituting, deleting, inserting and swapping adjacent characters. The word-level changes for the second prompt attack were made using the "embedding" augmenter, which changes the text of the prompt by replacing words with semantically similar words based on the word embeddings. For both prompt attacks, we made sure that certain parts of the prompt stayed intact, such as the placeholder "⟨test_case⟩" and other information related to the syntax of the output. For the word-level changes, we manually modified the resulting prompt to make sure that the changes kept the semantic content of the prompt intact, reverting word substitutions that were based on the wrong word sense (e.g., the word "following" was originally incorrectly replaced with "pursuit"). Figure 10 shows the generated adversarial prompts.

Charswap attack prompt

Act as a Python expert and consider the following etst of the Python Standard Library, version 3.10:
⟨test_case⟩
YOour job is to figure out whether this test will PASB or FAIL. Provide the rationale before givin an answer. Provide outputs that a PythLon expert would create.

Answer in this exact format:
Explanation: ⟨free text⟩
Label: ⟨PASS | FAIL⟩

(a) Character-level prompt attack using the charswap augmenter.

Embedding attack prompt

Act as a Python specialist and contemplate the following test of the Python Standard Library, version 3.10:
⟨test_case⟩
Your job is to figure out whether this test will PASS or FAIL. Offer the rationale before giving a response. Deliver outputs that a Python expert would create.

Respond in this exact format:
Explanation: ⟨free text⟩
Label: ⟨PASS | FAIL⟩

(b) Word-level prompt attack using the embedding augmenter

Figure 10 The generated adversarial prompts using the original Persona prompt (Figure 9). Changes made to the original prompt are highlighted in red.

The generated adversarial prompts were evaluated on the test set and used to calculate the performance drop rate (PDR) resulting from these prompt attacks by comparing the predictions with the predictions of the original prompt. The PDR's for the charswap and embedding attacks were 0.145 and 0.074 respectively. This means that the introduction of character-level changes (i.e., typos) into the prompt resulted in a 14.5% decrease in performance. Clearly, prompts should be carefully checked for typos before using them. However, introducing word-level changes into the prompt also decreased the accuracy of the predictions by 7.4%, even though these changes kept the original meaning of the prompt intact. This finding reinforces that seemingly innocuous changes to the prompt, such as the choice of words, can have a substantial effect on the performance of the prompt, again supporting our recommendation of using multiple prompts and comparing the results of these prompts (S4).

Sustainability

Using an LLM inherently comes with a significant energy footprint. However, several of the choices we made in our study have reduced the potential energy expenditure. In particular, our using a pre-trained model avoids the large footprint that results from the training phase of LLMs [129]. Further, we have used a relatively small model, run locally, since it was not necessary to use a larger model given our focus on replicability over raw

performance. Lastly, in order to keep inferences low, we used a small subset of the data for the initial testing of our prompts and used a method of confidence elicitation that does not require repeated inference. As a result of using a smaller model, we have likely used less energy than the original study, since Llama-3.1-8b has been estimated to use 0.017 Wh per query [142], whereas GPT 4 has been estimated to use between 0.3 and 3 Wh per query [130, 143].

S9. Analyse overfitting & degradation.
Since we have not performed training or fine-tuning, we do not have information on the performance of the model on this task during the training phase, which means that we cannot analyse the level of overfitting. However, since our prompts were selected and improved using the performance on the validation set, there is a possibility that we have incidentally tailored our prompts to the intricacies of the validation set as opposed to features of the general problem. To investigate this potential issue, we calculate the degradation ($M_{test} - M_{validation}$) of the performance between the validation and test sets, shown in Table 10.

The results show a positive degradation on all metrics, meaning that the performance has actually improved in the test set compared to the validation set. However, the differences are very small compared to the variations in performance we have seen on the validation set for different prompts and the performance on different test sets (Table 7 and 9). Therefore, we argue that the performance is quite stable between the validation and test set, suggesting that the steps of prompt design and prompt comparison did not inadvertently bias our prompt on the validation set. This result is unsurprising, since our final prompt does not incorporate any information unique to the validation set.

Table 10 Performance degradation from validation to test set.

	Precision	**Recall**	**Specificity**	**Accuracy**	**F1**	**F2**	**MCC**
Validation	0.514	0.740	0.300	0.520	0.607	0.680	0.045
Test	0.533	0.800	0.300	0.550	0.640	0.727	0.115
Degradation	0.019	0.060	0.000	0.030	0.033	0.047	0.071

S10. Visualize ROC.

S11. Apply statistical significance tests.

7.2.3 Summary of Findings

References

[1] Z. Guo et al., *Evaluating large language models: A comprehensive survey*, 2023. arXiv: 2310.19736 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2310.19736

[2] B. Kitchenham et al., "Preliminary guidelines for empirical research in software engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 721–734, 2002. DOI: 10.1109/TSE.2002.1027796

[3] T. Menzies and M. Shepperd, "Special issue on repeatable results in software engineering prediction," *Empirical Software Engineering*, vol. 17, no. 1–2, pp. 1–17, Jan. 2012, ISSN: 1573-7616. DOI: 10.1007/s10664-011-9193-5 [Online]. Available: http://dx.doi.org/10.1007/s10664-011-9193-5

[4] D. Dell'Anna, F. B. Aydemir, and F. Dalpiaz, "Evaluating classifiers in se research: The ecser pipeline and two replication studies," *Empirical Softw. Engg.*, vol. 28, no. 1, Nov. 2022, ISSN: 1382-3256. DOI: 10.1007/s10664-022-10243-1 [Online]. Available: https://doi.org/10.1007/s10664-022-10243-1

[5] Y. Guo, A. Ovadje, M. Al-garadi, and A. Sarker, "Evaluating large language models for health-related text classification tasks with public social media data," *Journal of the American Medical Informatics Association : JAMIA*, vol. 31, pp. 2181–2189, 2024. DOI: 10.1093/jamia/ocae210

[6] J. Fields, K. Chovanec, and P. Madiraju, "A survey of text classification with transformers: How wide? how large? how long? how accurate? how expensive? how safe?" *IEEE Access*, vol. 12, pp. 6518–6531, 2024. DOI: 10.1109/ACCESS.2024.3349952

[7] G. Marvin, N. Hellen, D. Jjingo, and J. Nakatumba-Nabende, "Prompt engineering in large language models," in *Data Intelligence and Cognitive Informatics*, I. J. Jacob, S. Piramuthu, and P. Falkowski-Gilski, Eds., Singapore: Springer Nature Singapore, 2024, pp. 387–402, ISBN: 978-981-99-7962-2.

[8] B. Woodworth, S. Gunasekar, M. I. Ohannessian, and N. Srebro, "Learning non-discriminatory predictors," in *Proceedings of the 2017 Conference on Learning Theory*, S. Kale and O. Shamir, Eds., ser. Proceedings of Machine Learning Research, vol. 65, PMLR, Jul. 2017, pp. 1920–1953. [Online]. Available: https://proceedings.mlr.press/v65/woodworth17a.html

[9] K. Zhu et al., *Promptrobust: Towards evaluating the robustness of large language models on adversarial prompts*, 2024. arXiv: 2306.04528 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2306.04528

[10] W. X. Zhao et al., *A survey of large language models*, 2025. arXiv: 2303.18223 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2303.18223

[11] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948. DOI: 10.1002/j.1538-7305.1948.tb01338.x

[12] D. Jurafsky and J. H. Martin, "Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition with language models," Online manuscript released January 12, 2025, 2025. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/

[13] H. Wang, J. Li, H. Wu, E. Hovy, and Y. Sun, "Pre-trained language models and their applications," *Engineering*, vol. 25, pp. 51–65, 2023, ISSN: 2095-8099. DOI: https://doi.org/10.1016/j.eng.2022.04.024 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2095809922006324

[14] T. Mikolov and G. Zweig, "Context dependent recurrent neural network language model," in *2012 IEEE Spoken Language Technology Workshop (SLT)*, IEEE, 2012, pp. 234–239.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] A. Vaswani et al., *Attention is all you need*, 2017. DOI: 10.48550/ARXIV.1706.03762 [Online]. Available: https://arxiv.org/abs/1706.03762

[17] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2016. arXiv: 1409.0473 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1409.0473

[18] J. Kaplan et al., *Scaling laws for neural language models*, 2020. arXiv: 2001.08361 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2001.08361

[19] J. Wei et al., *Emergent abilities of large language models*, 2022. arXiv: 2206.07682 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2206.07682

[20] X. Hou et al., *Large language models for software engineering: A systematic literature review*, 2024. arXiv: 2308.10620 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2308.10620

[21] M. Lewis et al., *Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension*, 2019. arXiv: 1910.13461 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1910.13461

[22] A. Roberts et al., "Scaling up models and data with t5x and seqio," *arXiv preprint arXiv:2203.17189*, 2022. [Online]. Available: https://arxiv.org/abs/2203.17189

[23] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, *On the properties of neural machine translation: Encoder-decoder approaches*, 2014. arXiv: 1409.1259 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1409.1259

[24] A. Asadi and R. Safabakhsh, "The encoder-decoder framework and its applications," in *Deep Learning: Concepts and Architectures*, W. Pedrycz and S.-M. Chen, Eds. Cham: Springer International Publishing, 2020, pp. 133–167, ISBN: 978-3-030-31756-0. DOI: 10.1007/978-3-030-31756-0_5 [Online]. Available: https://doi.org/10.1007/978-3-030-31756-0_5

[25] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. arXiv: 1810.04805 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1810.04805

[26] Y. Liu et al., *Roberta: A robustly optimized bert pretraining approach*, 2019. arXiv: 1907.11692 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1907.11692

[27] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[28] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, *Albert: A lite bert for self-supervised learning of language representations*, 2020. arXiv: 1909.11942 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1909.11942

[29] M. V. Koroteev, *Bert: A review of applications in natural language processing and understanding*, 2021. arXiv: 2103.11943 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2103.11943

[30] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al., "Improving language understanding by generative pre-training," 2018.

[31] T. B. Brown et al., *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2005.14165

[32] OpenAI et al., *Gpt-4 technical report*, 2024. arXiv: 2303.08774 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2303.08774

[33] R. A. Poldrack, T. Lu, and G. Beguš, *Ai-assisted coding: Experiments with gpt-4*, 2023. arXiv: 2304.13187 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2304.13187

[34] OpenAI, *Chatgpt*, https://chat.openai.com/, Available at https://chat.openai.com/, 2023.

[35] H. Touvron et al., *Llama 2: Open foundation and fine-tuned chat models*, 2023. arXiv: 2307.09288 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2307.09288

[36] K. Ronanki, B. Cabrero-Daniel, J. Horkoff, and C. Berger, *Requirements engineering using generative ai: Prompts and prompting patterns*, 2023. arXiv: 2311.03832 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2311.03832

[37] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[38] T. Brown et al., "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[39] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[40] Y. Zhou et al., *Large language models are human-level prompt engineers*, 2023. arXiv: 2211.01910 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2211.01910

[41] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. Springer Berlin Heidelberg, 2014, ISBN: 9783662438398. DOI: 10.1007/978-3-662-43839-8 [Online]. Available: http://dx.doi.org/10.1007/978-3-662-43839-8

[42] D. Zhang and J. Tsai, "Machine learning and software engineering," vol. 11, Feb. 2002, pp. 22–29, ISBN: 0-7695-1849-4. DOI: 10.1109/TAI.2002.1180784

[43] B. Kitchenham, T. Dybå, and M. Jorgensen, "Evidence-based software engineering," Jun. 2004, pp. 273–281, ISBN: 0-7695-2163-0. DOI: 10.1109/ICSE.2004.1317449

[44] V. Basili, F. Shull, and F. Lanubile, "Building knowledge through families of experiments," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 456–473, 1999. DOI: 10.1109/32.799939

[45] P. E. Shrout and J. L. Rodgers, "Psychology, science, and knowledge construction: Broadening perspectives from the replication crisis," *Annual Review of Psychology*, vol. 69, no. Volume 69, 2018, pp. 487–510, 2018, ISSN: 1545-2085. DOI: https://doi.org/10.1146/annurev-psych-122216-011845 [Online]. Available: https://www.annualreviews.org/content/journals/10.1146/annurev-psych-122216-011845

[46] S. E. Maxwell, M. Y.-K. Lau, and G. S. Howard, "Is psychology suffering from a replication crisis? what does "failure to replicate" really mean?" *The American psychologist*, vol. 70 6, pp. 487–98, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:11690879

[47] V. Amrhein, D. Trafimow, and S. Greenland, "Inferential statistics as descriptive statistics: There is no replication crisis if we don't expect replication," *The American Statistician*, vol. 73, pp. 262–270, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:56130495

[48] M. Cruz, B. Bernárdez, A. Durán, J. A. Galindo, and A. Ruiz-Cortés, "Replication of studies in empirical software engineering: A systematic mapping study, from 2013 to 2018," *IEEE Access*, vol. 8, pp. 26 773–26 791, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:209073520

[49] F. J. Shull, J. C. Carver, S. Vegas, and N. Juristo, "The role of replications in empirical software engineering," *Empirical Software Engineering*, vol. 13, no. 2, pp. 211–218, Jan. 2008, ISSN: 1573-7616. DOI: 10.1007/s10664-008-9060-1 [Online]. Available: http://dx.doi.org/10.1007/s10664-008-9060-1

[50] N. Juristo and S. Vegas, "The role of non-exact replications in software engineering experiments," *Empirical Software Engineering*, vol. 16, no. 3, pp. 295–324, Aug. 2010, ISSN: 1573-7616. DOI: 10.1007/s10664-010-9141-9 [Online]. Available: http://dx.doi.org/10.1007/s10664-010-9141-9

[51] M. Shepperd, N. Ajienka, and S. Counsell, "The role and value of replication in empirical software engineering results," *Information and Software Technology*, vol. 99, pp. 120–132, 2018, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2018.01.006 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584917304305

[52] M. Shepperd, "Replication studies considered harmful," in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, ser. ICSE-NIER '18, Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 73–76, ISBN: 9781450356626. DOI: 10.1145/3183399.3183423 [Online]. Available: https://doi.org/10.1145/3183399.3183423

[53] J. Siegmund, N. Siegmund, and S. Apel, "Views on internal and external validity in empirical software engineering," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, 2015, pp. 9–19. DOI: 10.1109/ICSE.2015.24

[54] S. Vegas, N. Juristo, A. Moreno, M. Solari, and P. Letelier, "Analysis of the influence of communication between researchers on experiment replication," in *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, ser. ISESE '06, Rio de Janeiro, Brazil: Association for Computing Machinery, 2006, pp. 28–37, ISBN: 1595932186. DOI: 10.1145/1159733.1159741 [Online]. Available: https://doi.org/10.1145/1159733.1159741

[55] J. C. Carver, "Towards reporting guidelines for experimental replications: A proposal," in *Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research (RESER)*, Held during ICSE 2010, Cape Town, South Africa, May 2010.

[56] O. S. Gómez, N. Juristo, and S. Vegas, "Understanding replication of experiments in software engineering: A classification," *Information and Software Technology*, vol. 56, no. 8, pp. 1033–1048, 2014, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2014.04.004 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584914000858

[57] S. Abualhaija et al., "Replication in requirements engineering: The nlp for re case," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 6, Jun. 2024, ISSN: 1049-331X. DOI: 10.1145/3658669 [Online]. Available: https://doi.org/10.1145/3658669

[58] M. J. Brandt et al., "The replication recipe: What makes for a convincing replication?" *Journal of Experimental Social Psychology*, vol. 50, pp. 217–224, 2014, ISSN: 0022-1031. DOI: https://doi.org/10.1016/j.jesp.2013.10.005 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022103113001819

[59] A. Fan et al., "Large language models for software engineering: Survey and open problems," in *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, 2023, pp. 31–53. DOI: 10.1109/ICSE-FoSE59343.2023.00008

[60] L. Belzner, T. Gabor, and M. Wirsing, "Large language model assisted software engineering: Prospects, challenges, and a case study," in *Bridging the Gap Between AI and Reality*, B. Steffen, Ed., Cham: Springer Nature Switzerland, 2024, pp. 355–374, ISBN: 978-3-031-46002-9.

[61] Z. Zheng et al., "Towards an understanding of large language models in software engineering tasks," *Empirical Software Engineering*, vol. 30, no. 2, Dec. 2024, ISSN: 1573-7616. DOI: 10.1007/s10664-024-10602-0 [Online]. Available: http://dx.doi.org/10.1007/s10664-024-10602-0

[62] I. Ozkaya, "Application of large language models to software engineering tasks: Opportunities, risks, and implications," *IEEE Software*, vol. 40, no. 3, pp. 4–8, 2023. DOI: 10.1109/MS.2023.3248401

[63] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, *Software testing with large language models: Survey, landscape, and vision*, 2024. arXiv: 2307.07221 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2307.07221

[64] C. Chen et al., "Clinicalbench: Can llms beat traditional ml models in clinical prediction?" *ArXiv*, vol. abs/2411.06469, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:273963111

[65] S. Vajjala and S. Shimangaud, *Text classification in the llm era – where do we stand?* 2025. arXiv: 2502.11830 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2502.11830

[66] Q. Zhang et al., *A survey on large language models for software engineering*, 2024. arXiv: 2312.15223 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2312.15223

[67] S. Baltes et al., *Guidelines for empirical studies in software engineering involving large language models*, 2025. arXiv: 2508.15503 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2508.15503

[68] J. White et al., *A prompt pattern catalog to enhance prompt engineering with chatgpt*, 2023. arXiv: 2302.11382 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2302.11382

[69] S. Ekin, "Prompt engineering for chatgpt: A quick guide to techniques, tips, and best practices," 2023. DOI: 10.36227/techrxiv.22683919.v2

[70] M. Sclar, Y. Choi, Y. Tsvetkov, and A. Suhr, *Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting*, 2024. arXiv: 2310.11324 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2310.11324

[71] M. Mizrahi, G. Kaplan, D. Malkin, R. Dror, D. Shahaf, and G. Stanovsky, *State of what art? a call for multi-prompt llm evaluation*, 2024. arXiv: 2401.00595 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2401.00595

[72] F. Maia Polo et al., *Efficient multi-prompt evaluation of llms*, May 2024. DOI: 10.48550/arXiv.2405.17202

[73] S. Arvidsson and J. Axell, "Prompt engineering guidelines for llms in requirements engineering," 2023.

[74] Q. Ye, M. Axmed, R. Pryzant, and F. Khani, *Prompt engineering a prompt engineer*, 2024. arXiv: 2311.05661 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2311.05661

[75] M. A. Zadenoori, L. Zhao, W. Alhoshan, and A. Ferrari, "Automatic prompt engineering: The case of requirements classification," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*, Springer, 2025, pp. 217–225.

[76] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, no. 1, Jan. 2020, ISSN: 1471-2164. DOI: 10.1186/s12864-019-6413-7 [Online]. Available: http://dx.doi.org/10.1186/s12864-019-6413-7

[77] G. M. Foody, "Challenges in the real world use of classification accuracy metrics: From recall and precision to the matthews correlation coefficient," *PLOS ONE*, vol. 18, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:263668955

[78] J. Yao and M. J. Shepperd, "Assessing software defection prediction performance: Why using the matthews correlation coefficient matters," *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:211818318

[79] Y. Liu et al., *Trustworthy llms: A survey and guideline for evaluating large language models' alignment*, 2024. arXiv: 2308.05374 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2308.05374

[80] Z. R. Tam, C.-K. Wu, Y.-L. Tsai, C.-Y. Lin, H.-y. Lee, and Y.-N. Chen, "Let me speak freely? a study on the impact of format restrictions on large language model performance.," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, F. Dernoncourt, D. Preoţiuc-Pietro, and A. Shimorina, Eds., Miami, Florida, US: Association for Computational Linguistics, Nov. 2024, pp. 1218–1236. DOI: 10.18653/v1/2024.emnlp-industry.91 [Online]. Available: https://aclanthology.org/2024.emnlp-industry.91/

[81] D. X. Long et al., *Llms are biased towards output formats! systematically evaluating and mitigating output format bias of llms*, 2025. arXiv: 2408.08656 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2408.08656

[82] D. Li, Y. Zhao, Z. Wang, C. Jung, and Z. Zhang, "Large language model-driven structured output: A comprehensive benchmark and spatial data generation framework," *ISPRS International Journal of Geo-Information*, vol. 13, no. 11, 2024, ISSN: 2220-9964. DOI: 10.3390/ijgi13110405 [Online]. Available: https://www.mdpi.com/2220-9964/13/11/405

[83] C. Xia et al., "FOFO: A benchmark to evaluate LLMs' format-following capability," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, L.-W. Ku, A. Martins, and V. Srikumar, Eds., Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 680–699. DOI: 10.18653/v1/2024.acl-long.40 [Online]. Available: https://aclanthology.org/2024.acl-long.40/

[84] Y. Chang et al., *A survey on evaluation of large language models*, 2023. arXiv: 2307.03109 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2307.03109

[85] M. Peeperkorn, T. Kouwenhoven, D. Brown, and A. Jordanous, *Is temperature the creativity parameter of large language models?* 2024. arXiv: 2405.00492 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2405.00492

[86] C. X. Yu, J. James, C. Si Yu, D. David, C. B. Hoe, and P. Hui-Li Phyllis, "Can llms have a fever? investigating the effects of temperature on llm security," in *Proceedings of the International Conference on Industrial Engineering and Operations Management*, ser. South American '24, IEOM Society International, May 2024. DOI: 10.46254/sa05.20240024 [Online]. Available: http://dx.doi.org/10.46254/SA05.20240024

[87] B. A. Kitchenham, D. Budgen, and O. Pearl Brereton, "Using mapping studies as the basis for further research – a participant-observer case study," *Information and Software Technology*, vol. 53, no. 6, pp. 638–651, 2011, Special Section: Best papers from the APSEC, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2010.12.011 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584910002272

[88] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," vol. 2, Jan. 2007.

[89] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, vol. 17, Jun. 2008.

[90] J. Nixon, M. W. Dusenberry, L. Zhang, G. Jerfel, and D. Tran, "Measuring calibration in deep learning.," in *CVPR workshops*, vol. 2, 2019.

[91] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *International conference on machine learning*, PMLR, 2017, pp. 1321–1330.

[92] J. Huang and C. X. Ling, "Using auc and accuracy in evaluating learning algorithms," *IEEE Transactions on knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.

[93] C. X. Ling, J. Huang, and H. Zhang, "Auc: A better measure than accuracy in comparing learning algorithms," in *Conference of the canadian society for computational studies of intelligence*, Springer, 2003, pp. 329–341.

[94] R. Meyes, M. Lu, C. W. De Puiseau, and T. Meisen, "Ablation studies in artificial neural networks," *arXiv preprint arXiv:1901.08644*, 2019.

[95] D. J. Benjamin et al., "Redefine statistical significance," *Nature human behaviour*, vol. 2, no. 1, pp. 6–10, 2018.

[96] V. Amrhein, F. Korner-Nievergelt, and T. Roth, "The earth is flat (p¿ 0.05): Significance thresholds and the crisis of unreplicable research," *PeerJ*, vol. 5, e3544, 2017.

[97] A. Homiar et al., "Development and evaluation of prompts for a large language model to screen titles and abstracts in a living systematic review," *BMJ Mental Health*, vol. 28, no. 1, e301762, Jul. 2025, ISSN: 2755-9734. DOI: 10.1136/bmjment-2025-301762 [Online]. Available: http://dx.doi.org/10.1136/bmjment-2025-301762

[98] H. Bichri, A. Chergui, and M. Hain, "Investigating the impact of train / test split ratio on the performance of pre-trained models with custom datasets," *International Journal of Advanced Computer Science and Applications*, vol. 15, no. 2, 2024. DOI: 10.14569/IJACSA.2024.0150235 [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2024.0150235

[99] Y. Xu and R. Goodacre, "On splitting training and validation set: A comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning," *Journal of analysis and testing*, vol. 2, no. 3, pp. 249–262, 2018.

[100] J.-H. Kim, "Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap," *Computational Statistics and Data Analysis*, vol. 53, no. 11, pp. 3735–3745, 2009, ISSN: 0167-9473. DOI: https://doi.org/10.1016/j.csda.2009.04.009 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167947309001601

[101] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, "An introduction to statistical learning," en, in (Springer texts in statistics), Springer texts in statistics. Cham, Switzerland: Springer International Publishing, Jul. 2023, ch. 5.

[102] G. C. Cawley and N. L. Talbot, "Efficient leave-one-out cross-validation of kernel fisher discriminant classifiers," *Pattern Recognition*, vol. 36, no. 11, pp. 2585–2592, 2003, ISSN: 0031-3203. DOI: https://doi.org/10.1016/S0031-3203(03)00136-5 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320303001365

[103] S. Wang, W. Zhou, H. Lu, A. Maleki, and V. Mirrokni, *Approximate leave-one-out for fast parameter tuning in high dimensions*, 2018. arXiv: 1807.02694 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1807.02694

[104] T.-T. Wong, "Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation," *Pattern Recognition*, vol. 48, no. 9, pp. 2839–2846, 2015, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2015.03.009 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320315000989

[105] S. Herbold, A. Trautsch, and F. Trautsch, "On the feasibility of automated prediction of bug and non-bug issues," *Empirical Software Engineering*, vol. 25, no. 6, pp. 5333–5369, Sep. 2020, ISSN: 1573-7616. DOI: 10.1007/s10664-020-09885-w [Online]. Available: http://dx.doi.org/10.1007/s10664-020-09885-w

[106] F. Dalpiaz, D. Dell'Anna, F. B. Aydemir, and S. Çevikol, "Requirements classification with interpretable machine learning and dependency parsing," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*, 2019, pp. 142–152. DOI: 10.1109/RE.2019.00025

[107] scikit-learn developers, *Cross-validation: Evaluating estimator performance*, Software documentation, 2025.

[108] N. Carlini et al., *Extracting training data from large language models*, 2021. arXiv: 2012.07805 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2012.07805

[109] V. B. Parthasarathy, A. Zafar, A. Khan, and A. Shahid, *The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities*, 2024. arXiv: 2408.13296 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2408.13296

[110] A. Halfon et al., *Stay tuned: An empirical study of the impact of hyperparameters on llm tuning in real-world applications*, 2024. arXiv: 2407.18990 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2407.18990

[111] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *The journal of machine learning research*, vol. 13, no. 1, pp. 281–305, 2012.

[112] P. Liashchynskyi and P. Liashchynskyi, *Grid search, random search, genetic algorithm: A big comparison for nas*, 2019. arXiv: 1912.06059 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1912.06059

[113] S. Loussaief and A. Abdelkrim, "Convolutional neural network hyper-parameters optimization based on genetic algorithms," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 10, 2018. DOI: 10.14569/IJACSA.2018.091031 [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2018.091031

[114] A. Agrawal, X. Yang, R. Agrawal, R. Yedida, X. Shen, and T. Menzies, "Simpler hyperparameter optimization for software analytics: Why, how, when?" *IEEE Transactions on Software Engineering*, vol. 48, no. 8, pp. 2939–2954, Aug. 2022, ISSN: 2326-3881. DOI: 10.1109/tse.2021.3073242 [Online]. Available: http://dx.doi.org/10.1109/TSE.2021.3073242

[115] J. He, M. Rungta, D. Koleczek, A. Sekhon, F. X. Wang, and S. Hasan, *Does prompt formatting have any impact on llm performance?* 2024. arXiv: 2411.10541 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2411.10541

[116] L. Ngweta, K. Kate, J. Tsay, and Y. Rizk, *Towards llms robustness to changes in prompt format styles*, 2025. arXiv: 2504.06969 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2504.06969

[117] M. A. Lones, "Avoiding common machine learning pitfalls," *Patterns*, vol. 5, no. 10, p. 101 046, Oct. 2024, ISSN: 2666-3899. DOI: 10.1016/j.patter.2024.101046 [Online]. Available: http://dx.doi.org/10.1016/j.patter.2024.101046

[118] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, "Learning from class-imbalanced data: Review of methods and applications," *Expert Systems with Applications*, vol. 73, pp. 220–239, 2017, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2016.12.035 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417416307175

[119] D. M. Powers, "What the f-measure doesn't measure: Features, flaws, fallacies and fixes," *arXiv preprint arXiv:1503.06410*, 2015.

[120] S. Kapoor et al., "Reforms: Consensus-based recommendations for machine-learning-based science," *Science Advances*, vol. 10, no. 18, May 2024, ISSN: 2375-2548. DOI: 10.1126/sciadv.adk3452 [Online]. Available: http://dx.doi.org/10.1126/sciadv.adk3452

[121] K. Tian et al., *Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback*, 2023. arXiv: 2305.14975 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2305.14975

[122] M. Xiong et al., *Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms*, 2024. arXiv: 2306.13063 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2306.13063

[123] Z. Kenton, T. Everitt, L. Weidinger, I. Gabriel, V. Mikulik, and G. Irving, *Alignment of language agents*, 2021. arXiv: 2103.14659 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2103.14659

[124] Y. Bang et al., *A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity*, 2023. arXiv: 2302.04023 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2302.04023

[125] B. Wang et al., *Decodingtrust: A comprehensive assessment of trustworthiness in gpt models*, 2024. arXiv: 2306.11698 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2306.11698

[126] Y.-A. Liu, R. Zhang, J. Guo, M. de Rijke, Y. Fan, and X. Cheng, *Robust neural information retrieval: An adversarial and out-of-distribution perspective*, 2024. arXiv: 2407.06992 [cs.IR]. [Online]. Available: https://arxiv.org/abs/2407.06992

[127] L. Cui et al., " API2Vec++: Boosting API Sequence Representation for Malware Detection and Classification," *IEEE Transactions on Software Engineering*, vol. 50, no. 08, pp. 2142–2162, Aug. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3422990 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3422990

[128] Q. Zhang et al., " APPT: Boosting Automated Patch Correctness Prediction via Fine-Tuning Pre-Trained Models," *IEEE Transactions on Software Engineering*, vol. 50, no. 03, pp. 474–494, Mar. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3354969 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3354969

[129] C.-J. Wu et al., *Sustainable ai: Environmental implications, challenges and opportunities*, 2022. arXiv: 2111.00364 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2111.00364

[130] A. de Vries, "The growing energy footprint of artificial intelligence," *Joule*, vol. 7, no. 10, pp. 2191–2194, 2023, ISSN: 2542-4351. DOI: https://doi.org/10.1016/j.joule.2023.09.004 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542435123003653

[131] X. Ying, "An overview of overfitting and its solutions," *Journal of Physics: Conference Series*, vol. 1168, no. 2, p. 022 022, Feb. 2019. DOI: 10.1088/1742-6596/1168/2/022022 [Online]. Available: https://doi.org/10.1088/1742-6596/1168/2/022022

[132] C. E. Metz, "Basic principles of roc analysis," *Seminars in Nuclear Medicine*, vol. 8, no. 4, pp. 283–298, 1978, ISSN: 0001-2998. DOI: https://doi.org/10.1016/S0001-2998(78)80014-2 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0001299878800142

[133] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006, ROC Analysis in Pattern Recognition, ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2005.10.010 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016786550500303X

[134] R. L. Wasserstein and N. A. Lazar, "The asa statement on p-values: Context, process, and purpose," *The American Statistician*, vol. 70, no. 2, pp. 129–133, 2016. DOI: 10.1080/00031305.2016.1154108 eprint: https://doi.org/10.1080/00031305.2016.1154108. [Online]. Available: https://doi.org/10.1080/00031305.2016.1154108

[135] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.

[136] O. Rainio, J. Teuho, and R. Klén, "Evaluation metrics and statistical tests for machine learning," *Scientific Reports*, vol. 14, no. 1, Mar. 2024, ISSN: 2045-2322. DOI: 10.1038/s41598-024-56706-x [Online]. Available: http://dx.doi.org/10.1038/s41598-024-56706-x

[137] A. Hora, "Predicting test results without execution," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 542–546, ISBN: 9798400706585. DOI: 10.1145/3663529.3663794 [Online]. Available: https://doi.org/10.1145/3663529.3663794

[138] A. Kostina, M. D. Dikaiakos, D. Stefanidis, and G. Pallis, *Large language models for text classification: Case study and comprehensive review*, 2025. arXiv: 2501.08457 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2501.08457

[139] M. Tufano, S. Chandel, A. Agarwal, N. Sundaresan, and C. Clement, *Predicting code coverage without execution*, 2023. arXiv: 2307.13383 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2307.13383

[140] C. Zhu, B. Xu, Q. Wang, Y. Zhang, and Z. Mao, *On the calibration of large language models and alignment*, 2023. arXiv: 2311.13240 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2311.13240

[141] J. Morris, E. Lifland, J. Y. Yoo, J. Grigsby, D. Jin, and Y. Qi, "Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020, pp. 119–126.

[142] Hugging Face, *Ai energy score*, https://huggingface.github.io/AIEnergyScore/, 2026.

[143] J. You, *How much energy does chatgpt use?* Accessed: 2026-02-17, 2025. [Online]. Available: https://epoch.ai/gradient-updates/how-much-energy-does-chatgpt-use

[144] Z. Ma, A. R. Chen, D. J. Kim, T.-H. Chen, and S. Wang, "Llmparser: An exploratory study on using large language models for log parsing," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639150 [Online]. Available: https://doi.org/10.1145/3597503.3639150

[145] M. Rinard, "Software engineering research in a world with generative artificial intelligence," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3649399 [Online]. Available: https://doi.org/10.1145/3597503.3649399

[146] Y. Peng, S. Gao, C. Gao, Y. Huo, and M. Lyu, "Domain knowledge matters: Improving prompts with fix templates for repairing python type errors," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3608132 [Online]. Available: https://doi.org/10.1145/3597503.3608132

[147] J. Xu et al., "Unilog: Automatic logging via llm and in-context learning," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3623326 [Online]. Available: https://doi.org/10.1145/3597503.3623326

[148] Y. Huang et al., "Crashtranslator: Automatically reproducing mobile application crashes directly from stack trace," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3623298 [Online]. Available: https://doi.org/10.1145/3597503.3623298

[149] Q. Guo et al., "Exploring the potential of chatgpt in automated code refinement: An empirical study," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3623306 [Online]. Available: https://doi.org/10.1145/3597503.3623306

[150] H. Yu et al., "Codereval: A benchmark of pragmatic code generation with generative pre-trained models," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3623316 [Online]. Available: https://doi.org/10.1145/3597503.3623316

[151] M. Geng et al., "Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3608134 [Online]. Available: https://doi.org/10.1145/3597503.3608134

[152] S. Feng and C. Chen, "Prompting is all you need: Automated android bug replay with large language models," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3608137 [Online]. Available: https://doi.org/10.1145/3597503.3608137

[153] Y. Deng, C. S. Xia, C. Yang, S. D. Zhang, S. Yang, and L. Zhang, "Large language models are edge-case generators: Crafting unusual programs for fuzzing deep learning libraries," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3623343 [Online]. Available: https://doi.org/10.1145/3597503.3623343

[154] J. Chen, X. Hu, Z. Li, C. Gao, X. Xia, and D. Lo, "Code search is all you need? improving code suggestions with code search," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639085 [Online]. Available: https://doi.org/10.1145/3597503.3639085

[155] Z. Sun, X. Du, F. Song, S. Wang, and L. Li, "When neural code completion models size up the situation: Attaining cheaper and faster completion through dynamic model inference," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639120 [Online]. Available: https://doi.org/10.1145/3597503.3639120

[156] A. Al-Kaswan, M. Izadi, and A. van Deursen, "Traces of memorisation in large language models for code," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639133 [Online]. Available: https://doi.org/10.1145/3597503.3639133

[157] M. Izadi, J. Katzy, T. Van Dam, M. Otten, R. M. Popescu, and A. Van Deursen, "Language models for code completion: A practical evaluation," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639138 [Online]. Available: https://doi.org/10.1145/3597503.3639138

[158] X. Du et al., "Evaluating large language models in class-level code generation," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639219 [Online]. Available: https://doi.org/10.1145/3597503.3639219

[159] R. Pan et al., "Lost in translation: A study of bugs introduced by large language models while translating code," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639226 [Online]. Available: https://doi.org/10.1145/3597503.3639226

[160] Y. W. Chow, L. Di Grazia, and M. Pradel, "Pyty: Repairing static type errors in python," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.

1145/3597503.3639184 [Online]. Available: https://doi.org/10.1145/3597503.3639184

[161] Y. Jiang et al., "Xpert: Empowering incident management with query recommendations via large language models," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639081 [Online]. Available: https://doi.org/10.1145/3597503.3639081

[162] Y. Su et al., "Enhancing exploratory testing by large language model and knowledge graph," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639157 [Online]. Available: https://doi.org/10.1145/3597503.3639157

[163] Z. Liu et al., "Make llm a testing expert: Bringing human-like interaction to mobile gui testing via functionality-aware decisions," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639180 [Online]. Available: https://doi.org/10.1145/3597503.3639180

[164] C. S. Xia, M. Paltenghi, J. Le Tian, M. Pradel, and L. Zhang, "Fuzz4all: Universal fuzzing with large language models," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639121 [Online]. Available: https://doi.org/10.1145/3597503.3639121

[165] Z. Liu et al., "Testing the limits: Unusual text inputs generation for mobile app crash detection with large language model," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639118 [Online]. Available: https://doi.org/10.1145/3597503.3639118

[166] J. Fu, J. Liang, Z. Wu, and Y. Jiang, "Sedar: Obtaining high-quality seeds for dbms fuzzing via cross-dbms sql transfer," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639210 [Online]. Available: https://doi.org/10.1145/3597503.3639210

[167] Y. Nong et al., "Vgx: Large-scale sample generation for boosting learning-based software vulnerability analyses," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639116 [Online]. Available: https://doi.org/10.1145/3597503.3639116

[168] M. M. Imran, P. Chatterjee, and K. Damevski, "Uncovering the causes of emotions in software developer communication using zero-shot llms," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639223 [Online]. Available: https://doi.org/10.1145/3597503.3639223

[169] J. Xu, R. Yang, Y. Huo, C. Zhang, and P. He, "Divlog: Log parsing with prompt enhanced in-context learning," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639155 [Online]. Available: https://doi.org/10.1145/3597503.3639155

[170] T. Ahmed, K. S. Pai, P. Devanbu, and E. Barr, "Automatic semantic augmentation of language model prompts (for code summarization)," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639183 [Online]. Available: https://doi.org/10.1145/3597503.3639183

[171] Y. Yang, X. Hu, X. Xia, D. Lo, and X. Yang, "Streamlining java programming: Uncovering well-formed idioms with idiomine," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639135 [Online]. Available: https://doi.org/10.1145/3597503.3639135

[172] M. Toslali et al., "Agrabot: Accelerating third-party security risk management in enterprise setting through generative ai," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 74–79, ISBN: 9798400706585. DOI: 10.1145/3663529.3663829 [Online]. Available: https://doi.org/10.1145/3663529.3663829

[173] O. Dunay et al., "Multi-line ai-assisted code authoring," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 150–160, ISBN: 9798400706585. DOI: 10.1145/3663529.3663836 [Online]. Available: https://doi.org/10.1145/3663529.3663836

[174] N. Alshahwan et al., "Automated unit test improvement using large language models at meta," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 185–196, ISBN: 9798400706585. DOI: 10.1145/3663529.3663839 [Online]. Available: https://doi.org/10.1145/3663529.3663839

[175] D. Roy et al., "Exploring llm-based agents for root cause analysis," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 208–219, ISBN: 9798400706585. DOI: 10.1145/3663529.3663841 [Online]. Available: https://doi.org/10.1145/3663529.3663841

[176] S. Wu et al., "Combating missed recalls in e-commerce search: A cot-prompting testing approach," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 220–231, ISBN: 9798400706585. DOI: 10.1145/3663529.3663842 [Online]. Available: https://doi.org/10.1145/3663529.3663842

[177] X. Zhang et al., "Automated root causing of cloud incidents using in-context learning with gpt-4," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 266–277, ISBN: 9798400706585. DOI: 10.1145/3663529.3663846 [Online]. Available: https://doi.org/10.1145/3663529.3663846

[178] K. Sarda, Z. Namrud, M. Litoiu, L. Shwartz, and I. Watts, "Leveraging large language models for the auto-remediation of microservice applications: An experimental study," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 358–369, ISBN: 9798400706585. DOI: 10.1145/3663529.3663855 [Online]. Available: https://doi.org/10.1145/3663529.3663855

[179] D. Goel et al., "X-lifecycle learning for cloud incident management using llms," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*,

ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 417–428, ISBN: 9798400706585. DOI: 10.1145/3663529.3663861 [Online]. Available: https://doi.org/10.1145/3663529.3663861

[180] Y. Chen, Z. Hu, C. Zhi, J. Han, S. Deng, and J. Yin, "Chatunitest: A framework for llm-based test generation," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 572–576, ISBN: 9798400706585. DOI: 10.1145/3663529.3663801 [Online]. Available: https://doi.org/10.1145/3663529.3663801

[181] H. Liu, Z. Jia, H. Zhou, H. Zhou, and S. Li, "Go the extra mile: Fixing propagated error-handling bugs," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 661–662, ISBN: 9798400706585. DOI: 10.1145/3663529.3663868 [Online]. Available: https://doi.org/10.1145/3663529.3663868

[182] S. B. Chavan and S. Mondal, "Do large language models recognize python identifier swaps in their generated code?" In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 663–664, ISBN: 9798400706585. DOI: 10.1145/3663529.3663869 [Online]. Available: https://doi.org/10.1145/3663529.3663869

[183] H. Patel, K. A. Shah, and S. Mondal, "Do large language models generate similar codes from mutated prompts? a case study of gemini pro," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 671–672, ISBN: 9798400706585. DOI: 10.1145/3663529.3663873 [Online]. Available: https://doi.org/10.1145/3663529.3663873

[184] D. Souza, "Comparing gemini pro and gpt-3.5 in algorithmic problems," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 698–700, ISBN: 9798400706585. DOI: 10.1145/3663529.3664463 [Online]. Available: https://doi.org/10.1145/3663529.3664463

[185] M. Schafer, S. Nadi, A. Eghbali, and F. Tip, " An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation," *IEEE Transactions on Software Engineering*, vol. 50, no. 01, pp. 85–105, Jan. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2023.3334955 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2023.3334955

[186] R. Tufano, O. Dabic, A. Mastropaolo, M. Ciniselli, and G. Bavota, " Code Review Automation: Strengths and Weaknesses of the State of the Art," *IEEE Transactions on Software Engineering*, vol. 50, no. 02, pp. 338–353, Feb. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2023.3348172 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2023.3348172

[187] Y. Zhang et al., " Automatic Commit Message Generation: A Critical Review and Directions for Future Work," *IEEE Transactions on Software Engineering*, vol. 50, no. 04, pp. 816–835, Apr. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3364675 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3364675

[188] Y. Tang, Z. Liu, Z. Zhou, and X. Luo, " ChatGPT vs SBST: A Comparative Assessment of Unit Test Suite Generation," *IEEE Transactions on Software Engineering*, vol. 50, no. 06, pp. 1340–1359, Jun. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3382365 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3382365

[189] Z. Liu, Y. Tang, X. Luo, Y. Zhou, and L. F. Zhang, " No Need to Lift a Finger Anymore? Assessing the Quality of Code Generation by ChatGPT," *IEEE Transactions on Software Engineering*, vol. 50, no. 06, pp. 1548–1584, Jun. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3392499 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3392499

[190] C. Liu, P. Cetin, Y. Patodia, B. Ray, S. Chakraborty, and Y. Ding, " Automated Code Editing With Search-Generate-Modify," *IEEE Transactions on Software Engineering*, vol. 50, no. 07, pp. 1675–1686, Jul. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3376387 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3376387

[191] H. Tu, Z. Zhou, H. Jiang, I. N. B. Yusuf, Y. Li, and L. Jiang, " Isolating Compiler Bugs by Generating Effective Witness Programs With Large Language Models," *IEEE Transactions on Software Engineering*, vol. 50, no. 07, pp. 1768–1788, Jul. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3397822 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3397822

[192] C. Fang et al., " Esale: Enhancing Code-Summary Alignment Learning for Source Code Summarization," *IEEE Transactions on Software Engineering*, vol. 50, no. 08, pp. 2077–2095, Aug. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3422274 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3422274

[193] S. Fakhoury, A. Naik, G. Sakkas, S. Chakraborty, and S. K. Lahiri, " LLM-Based Test-Driven Interactive Code Generation: User Study and Empirical Evaluation," *IEEE Transactions on Software Engineering*, vol. 50, no. 09, pp. 2254–2268, Sep. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3428972 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3428972

[194] J. Shin, H. Hemmati, M. Wei, and S. Wang, " Assessing Evaluation Metrics for Neural Test Oracle Generation," *IEEE Transactions on Software Engineering*, vol. 50, no. 09, pp. 2337–2349, Sep. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3433463 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3433463

[195] G. Yang, Y. Zhou, X. Chen, X. Zhang, T. Y. Zhuo, and T. Chen, " Chain-of-Thought in Neural Code Generation: From and for Lightweight Language Models," *IEEE Transactions on Software Engineering*, vol. 50, no. 09, pp. 2437–2457, Sep. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3440503 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3440503

[196] Z. Zhou, M. Li, H. Yu, G. Fan, P. Yang, and Z. Huang, " Learning to Generate Structured Code Summaries From Hybrid Code Context," *IEEE Transactions on Software Engineering*, vol. 50, no. 10, pp. 2512–2528, Oct. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3439562 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3439562

[197] S. Kang, J. Yoon, N. Askarbekkyzy, and S. Yoo, " Evaluating Diverse Large Language Models for Automatic and General Bug Reproduction," *IEEE Transactions on Software Engineering*, vol. 50, no. 10, pp. 2677–2694, Oct. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3450837 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3450837

[198] H. Wang, Z. Gao, X. Hu, D. Lo, J. Grundy, and X. Wang, " Just-In-Time TODO-Missed Commits Detection," *IEEE Transactions on Software Engineering*, vol. 50, no. 11, pp. 2732–2752, Nov. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3405005 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3405005

[199] Y. Li et al., " Exploring the Effectiveness of LLMs in Automated Logging Statement Generation: An Empirical Study," *IEEE Transactions on Software Engineering*, vol. 50, no. 12, pp. 3188–3207,

Dec. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3475375 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3475375

[200] P. Xue et al., " Automated Commit Message Generation With Large Language Models: An Empirical Study and Beyond," *IEEE Transactions on Software Engineering*, vol. 50, no. 12, pp. 3208–3224, Dec. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3478317 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3478317

[201] D. Liao et al., " $\mathbf{A^3}$A3-CodGen: A Repository-Level Code Generation Framework for Code Reuse With Local-Aware, Global-Aware, and Third-Party-Library-Aware," *IEEE Transactions on Software Engineering*, vol. 50, no. 12, pp. 3369–3384, Dec. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3486195 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3486195

[202] B. Steenhoek, H. Gao, and W. Le, "Dataflow analysis-inspired deep learning for efficient vulnerability detection," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3623345 [Online]. Available: https://doi.org/10.1145/3597503.3623345

[203] L. Ma et al., "Knowlog: Knowledge enhanced pre-trained language model for log understanding," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3623304 [Online]. Available: https://doi.org/10.1145/3597503.3623304

[204] Z. Zhou, C. Sha, and X. Peng, "On calibration of pre-trained code models," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639126 [Online]. Available: https://doi.org/10.1145/3597503.3639126

[205] W. Wang, Y. Li, A. Li, J. Zhang, W. Ma, and Y. Liu, "An empirical study on noisy label learning for program understanding," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639217 [Online]. Available: https://doi.org/10.1145/3597503.3639217

[206] Y. Sun et al., "Gptscan: Detecting logic vulnerabilities in smart contracts by combining gpt with program analysis," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639117 [Online]. Available: https://doi.org/10.1145/3597503.3639117

[207] M. H. Tanzil, J. Y. Khan, and G. Uddin, "Chatgpt incorrectness detection in software reviews," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639194 [Online]. Available: https://doi.org/10.1145/3597503.3639194

[208] J. Sun, J. Chen, Z. Xing, Q. Lu, X. Xu, and L. Zhu, "Where is it? tracing the vulnerability-relevant files from vulnerability reports," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639202 [Online]. Available: https://doi.org/10.1145/3597503.3639202

[209] Y. Wang, J. A. H. López, U. Nilsson, and D. Varró, "Using run-time information to enhance static analysis of machine learning code in notebooks," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Gal-

inhas, Brazil: Association for Computing Machinery, 2024, pp. 497–501, ISBN: 9798400706585. DOI: 10.1145/3663529.3663785 [Online]. Available: https://doi.org/10.1145/3663529.3663785

[210] Y. Shen and T. Breaux, " Stakeholder Preference Extraction From Scenarios," *IEEE Transactions on Software Engineering*, vol. 50, no. 01, pp. 69–84, Jan. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2023.3333265 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2023.3333265

[211] W. Sun, Z. Guo, M. Yan, Z. Liu, Y. Lei, and H. Zhang, " Method-Level Test-to-Code Traceability Link Construction by Semantic Correlation Learning," *IEEE Transactions on Software Engineering*, vol. 50, no. 10, pp. 2656–2676, Oct. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3449917 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3449917

[212] X. Zhou et al., " Leveraging Large Language Model for Automatic Patch Correctness Assessment," *IEEE Transactions on Software Engineering*, vol. 50, no. 11, pp. 2865–2883, Nov. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3452252 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3452252

[213] Y. Jiang, Y. Zhang, X. Su, C. Treude, and T. Wang, " StagedVulBERT: Multigranular Vulnerability Detection With a Novel Pretrained Code Model," *IEEE Transactions on Software Engineering*, vol. 50, no. 12, pp. 3454–3471, Dec. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3493245 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3493245

[214] A. Z. H. Yang, C. Le Goues, R. Martins, and V. Hellendoorn, "Large language models for test-free fault localization," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3623342 [Online]. Available: https://doi.org/10.1145/3597503.3623342

[215] Y. Cai, A. Yadavally, A. Mishra, G. Montejo, and T. Nguyen, "Programming assistant for exception handling with codebert," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639188 [Online]. Available: https://doi.org/10.1145/3597503.3639188

[216] D. Nam, A. Macvean, V. Hellendoorn, B. Vasilescu, and B. Myers, "Using an llm to help with code understanding," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639187 [Online]. Available: https://doi.org/10.1145/3597503.3639187

[217] Z. Yu et al., "Monitorassistant: Simplifying cloud service monitoring via large language models," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 38–49, ISBN: 9798400706585. DOI: 10.1145/3663529.3663826 [Online]. Available: https://doi.org/10.1145/3663529.3663826

[218] D. Pomian et al., "Em-assist: Safe automated extractmethod refactoring with llms," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024, Porto de Galinhas, Brazil: Association for Computing Machinery, 2024, pp. 582–586, ISBN: 9798400706585. DOI: 10.1145/3663529.3663803 [Online]. Available: https://doi.org/10.1145/3663529.3663803

[219] R. Pan, T. A. Ghaleb, and L. C. Briand, " LTM: Scalable and Black-Box Similarity-Based Test Suite Minimization Based on Language Models," *IEEE Transactions on Software Engineering*, vol. 50,

no. 11, pp. 3053–3070, Nov. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3469582 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3469582

[220] S. Gao et al., "Learning in the wild: Towards leveraging unlabeled data for effectively tuning pre-trained code models," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639216 [Online]. Available: https://doi.org/10.1145/3597503.3639216

[221] M. Nashaat and J. Miller, " Towards Efficient Fine-Tuning of Language Models With Organizational Data for Automated Software Review," *IEEE Transactions on Software Engineering*, vol. 50, no. 09, pp. 2240–2253, Sep. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3428324 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3428324

[222] S. Fatima, H. Hemmati, and L. C. Briand, " FlakyFix: Using Large Language Models for Predicting Flaky Test Fix Categories and Test Code Repair," *IEEE Transactions on Software Engineering*, vol. 50, no. 12, pp. 3146–3171, Dec. 2024, ISSN: 1939-3520. DOI: 10.1109/TSE.2024.3472476 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3472476

[223] Y. Zhang et al., "Learning-based widget matching for migrating gui test cases," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3623322 [Online]. Available: https://doi.org/10.1145/3597503.3623322

Appendices

A Supplemental Mapping Study Results

For brevity, sources were excluded for some figures and tables and are included here for further reference. Table 11 shows the distribution of the application types of LLMs of 83 total papers (see Figure 2).

Table 11 References for the distribution of LLM application types.

Category	Number of studies	References
Generation	58	[144–201]
Classification	15	[127, 128, 137, 202–213]
Recommendation	6	[214–219]
Generation & Classification	3	[220–222]
Classification & Recommendation	1	[223]

The following tables apply to the 18 papers included in the mapping study. Table 12 shows the number of studies belonging to each venue. Table 13 shows the number of papers using prompts for classification. See also Figure 3 and 4.

Table 12 References for distribution of venue in the mapping study.

Venue	Number of studies	References
TSE	8	[127, 128, 210–213, 221, 222]
FSE	2	[137, 209]
ICSE	8	[202, 203, 205–208, 220, 223]

Table 13 References for the distribution of prompt usage in the mapping study.

Prompt-based	Number of studies	References
No	10	[127, 128, 202, 203, 205, 208, 210, 220, 222, 223]
Yes	8	[137, 206, 207, 209, 211–213, 221]

For the main results of the mapping study, Table 14 shows the full references. Further, Table 15 shows the references for the papers that use a prompt-based approach to classification.

Table 14 References for the mapping study results.

Category	Total	References
Total mapping study papers	18	[127, 128, 137, 202, 203, 205–213, 220–223]
Evaluation metrics		
Precision	17 (94.4%)	[127, 128, 137, 202, 203, 205–211, 213, 220–223]
Recall	17 (94.4%)	[127, 128, 137, 202, 203, 205–211, 213, 220–223]
Accuracy	11 (61.1%)	[127, 128, 137, 203, 205, 207, 208, 210, 212, 213, 221]
F-Score	17 (94.4%)	[127, 128, 202, 203, 205–213, 220–223]
AUC	3 (16.7%)	[128, 208, 212]
ROC plots	0 (0.0%)	
MCC	0 (0.0%)	
Calibration	0 (0.0%)	
Fairness	0 (0.0%)	
Robustness		
Data distribution & Adversarial Attacks	1 (5.5%)	[127]
Noisy data	1 (5.5%)	[205]
Metrics justification		
No	9 (50.0%)	[127, 137, 205–210, 221]
Implicit	0 (0.0%)	
Previous work	5 (27.8%)	[128, 212, 213, 220, 223]
Yes	4 (22.2%)	[202, 203, 211, 222]
Confusion matrix	4 (22.2%)	[137, 206, 211, 212]
Evaluation over multiple data sets	5 (27.8%)	[128, 206, 211, 220, 223]
Type of baseline		
None	2 (11.1%)	[137, 206]
Own	3 (16.7%)	[207, 209, 222]
External	4 (22.2%)	[128, 203, 221, 223]
External and Own	9 (50.0%)	[127, 202, 205, 208, 210–213, 220]
Analysis of statistical significance	5 (27.8%)	[202, 211–213, 222]
Statistical significance justification		
No	4 (80.0%)	[211–213, 222]
Implicit	0 (0.0%)	
Previous work	1 (20.0%)	[202]
Yes	0 (0.0%)	

Table 15 References for the mapping study prompting results.

Category	Total	References
Total prompt-based classification papers	8	[137, 206, 207, 209, 211–213, 221]
Prompt approach		
Chain-of-verification	1 (12.5%)	[207]
Few-shot	2 (25.0%)	[213, 221]
Mimic-in-the-background	1 (12.5%)	[206]
Not reported	1 (12.5%)	[209]
Zero-shot	3 (37.5%)	[137, 211, 212]
Final prompt		
Not reported	2 (25.0%)	[209, 213]
Own	6 (75.0%)	[137, 206, 207, 211, 212, 221]
Prompt approach justification		
No	5 (62.5%)	[137, 209, 211–213]
Yes	3 (37.5%)	[206, 207, 221]
Final prompt justification		
No	4 (50.0%)	[207, 209, 211, 213]
Previous Work	1 (12.5%)	[137]
Yes	3 (37.5%)	[206, 212, 221]
Evaluation over multiple prompts		
No	8 (100.0%)	[137, 206, 207, 209, 211–213, 221]
Temperature		
No	7 (87.5%)	[137, 207, 209, 211–213, 221]
Yes	1 (12.5%)	[206]