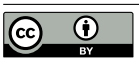
Automated Classification, Root Cause Analysis, and Repair Recommendations for Failed Mobile Testing by Specialized LLM

CHUN LI, Nanjing University, China
FEI WANG, Nanjing University, China
MINXUE PAN, Nanjing University, China
ZHONG LI, Nanjing University, China
MENGLIANG ZENG, OPPO, China
BIN ZHANG, OPPO, China
XUEJIAO YU, OPPO, China
BOYUN WANG, OPPO, China
KAIJIAN HUA, OPPO, China
XUANDONG LI, Nanjing University, China

Engineers utilize test scripts to conduct testing on mobile applications to ensure software reliability. In industrial settings, failures in mobile application testing stem not only from faults in the program or scripts but also from other aspects, such as the testing environment, making it challenging to apply existing automated analysis methods in such settings. Furthermore, mobile testing in industrial contexts generates a large volume of artifacts, such as screenshots and logs, and involves diverse device models, varied test environments, and complex functionalities. All these factors make the manual analysis of failed mobile tests a process that is typically time-consuming, costly, and labor-intensive. Large language models (LLMs) with strong logical reasoning capabilities offer a novel perspective for automated analysis of failed mobile tests. However, due to their limited knowledge of mobile testing, general LLMs can only achieve sub-optimal effectiveness in these tasks. To address these challenges, we propose AnaDroid, a novel LLM training framework specifically tailored for analyzing failed mobile tests. AnaDroid performs feature preprocessing for failed mobile tests and trains the large language model through two novel, tailored training procedures. Specifically, AnaDroid first leverages bidirectional pre-training to instill the domain-specific knowledge and reasoning capabilities required for failed test analysis into the model. AnaDroid further employs preference optimization to enhance the model's capacity to capture critical information within the input context, thereby further improving its analytical effectiveness. Through extensive evaluations across six open-source LLMs of diverse architectures and scales on a large-scale industrial dataset of 284k failed mobile testing scripts, AnaDroid outperformed all 4 baselines across three analysis tasks. Furthermore, during a month-long deployment within a global company's testing system, AnaDroid demonstrated its practical utility by successfully providing root cause and repair recommendations for 78% and 81% of the 11,304 failed tests, respectively.

Authors' Contact Information: Chun Li, Nanjing University, State Key Laboratory for Novel Software Technology, Nanjing, China, chunli@smail.nju.edu.cn; Fei Wang, Nanjing University, State Key Laboratory for Novel Software Technology, Nanjing, China, 522024320149@smail.nju.edu.cn; Minxue Pan, Nanjing University, State Key Laboratory for Novel Software Technology, Nanjing, China, mxp@nju.edu.cn; Zhong Li, Nanjing University, State Key Laboratory for Novel Software Technology, Nanjing, China, lizhong@nju.edu.cn; Mengliang Zeng, OPPO, Dongguan, China, zengmengliang@oppo.com; Bin Zhang, OPPO, Dongguan, China, zhangbin3@oppo.com; Xuejiao Yu, OPPO, Dongguan, China, yuxuejiao@oppo.com; Boyun Wang, OPPO, Dongguan, China, wangboyun@oppo.com; Kaijian Hua, OPPO, Dongguan, China, huakaijian@oppo.com; Xuandong Li, Nanjing University, State Key Laboratory for Novel Software Technology, Nanjing, China, lxd@nju.edu.cn.



ACM 2994-970X/2026/10-ARTISSTA006
https://doi.org/10.1145/3832097

CCS Concepts: • **Software and its engineering → Software testing and debugging**.

Additional Key Words and Phrases: Mobile Testing, Specialized LLM

ACM Reference Format:
Chun Li, Fei Wang, Minxue Pan, Zhong Li, Mengliang Zeng, Bin Zhang, Xuejiao Yu, Boyun Wang, Kaijian Hua, and Xuandong Li. 2026. Automated Classification, Root Cause Analysis, and Repair Recommendations for Failed Mobile Testing by Specialized LLM. *Proc. ACM Softw. Eng.* 3, ISSTA, Article ISSTA006 (October 2026), 25 pages. https://doi.org/10.1145/3832097

1 Introduction

In recent years, the widespread adoption of mobile devices has rendered mobile apps increasingly integral to our daily lives. Consequently, it has become critical for developers to ensure that their apps maintain high quality and perform reliably under user expectations. To this end, engineers develop test scripts that simulate interaction with the device to test the functionalities, such as file transfer and firmware update [4, 43, 60]. When a test fails, engineers rely on manual analysis to determine the cause, using artifacts generated during the testing process, such as logs, screenshots, stack traces, and other information. This process is often time-consuming, costly, and labor-intensive [5, 27]. In industrial settings, mobile testing involves large-scale mobile applications, diverse device models, various operating system versions, multiple test environments, and extensive functionalities. These complex factors lead to a broader range of failure causes of mobile testing, including network issues in the test environment, hardware malfunctions on test devices, operating system issues, and faults in either the application code or the test scripts. This complexity necessitates that engineers analyze each failure to assign the rectification task to different personnel, for instance, within development or testing departments, rather than the issues being resolvable by the engineer alone. Therefore, engineers primarily focus on classifying failed mobile tests and providing root cause analysis and corresponding repair recommendations in natural language [66], rather than directly performing the fixes themselves. Furthermore, mobile testing in industry can generate tens of thousands of lines of logs and numerous screenshots, also posing significant challenges for engineers in diagnosing and resolving failures. Consequently, there is a tremendous demand in the industry for automated analysis of failed mobile tests.

While extensive research exists on root cause localization [22, 28, 31, 38], program repair [1, 55, 56, 66], and GUI test script repair [4, 43, 60], these approaches cannot be directly applicable to our scenario. This is because such work typically attributes failures to a single source: either faulty program code or an incorrect or outdated test script. Under this assumption, most failures can be resolved automatically. However, in an industrial mobile testing scenario, the applicability of these approaches is limited due to the broader range of failure causes. For instance, when testing a file transfer over a proprietary protocol, a test failure may stem not only from implementation errors in the function or script but also from environmental settings, such as incorrectly configured device network state. Moreover, analyzing failed mobile tests involves multiple diverse features, such as logs and screenshots, which current approaches do not adequately consider. This multifaceted requirement further limits the adaptation of existing work to our context [32, 38, 55].

Large language models (LLMs) [2, 42, 51] trained on massive datasets have demonstrated remarkable effectiveness in natural language understanding and logical reasoning [1, 14, 35, 59], prompting a surge of recent research efforts to apply LLMs to various software engineering tasks. The reasoning capabilities of LLMs make them highly promising for analyzing failed mobile tests. However, directly leveraging general LLMs to analyze failed mobile tests yields sub-optimal performance, as demonstrated in our experiments. This is because analyzing the failed mobile tests requires the model to possess substantial domain knowledge in mobile testing. Prior studies in

other domains, such as medicine [50, 70] and law [10, 30], have observed similar findings. Moreover, LLMs with no prior exposure to mobile testing knowledge may also struggle to perform effective reasoning and analysis based on testing information such as documentation, logs, and screenshots. Therefore, enabling LLMs to effectively learn the mobile testing knowledge is crucial for leveraging them to automatically analyze failed mobile tests.

In this paper, we identify the critical problem of classification (CLS), root cause analysis (RCA), and repair recommendation (RR) for failed mobile tests. To address this, we propose AnaDroid, a novel LLM training framework specifically designed for analyzing failed mobile tests. The insight underlying AnaDroid is that the datasets composed of analyzed failed mobile tests inherently contain knowledge and reasoning logic required for analyzing failed mobile tests. By training LLMs on such data, we can enable them to acquire the mobile testing knowledge required to effectively analyze failed tests.

To realize this idea, AnaDroid incorporates a preprocessing pipeline tailored for multimodel data and a three-stage training process: pre-training, fine-tuning, and preference optimization. We perform data cleaning and compression on screenshots, logs, documents, and stack traces to minimize the context length, thereby enhancing model inference performance. The pre-training stage is the primary phase where LLMs learn knowledge [16, 69]. Therefore, we first leverage pre-training to enable the model to learn the requisite knowledge from our dataset for analyzing failed tests. Furthermore, due to the decoder-only architecture of LLMs, where the representation of each token is influenced solely by its left-side context, LLMs are unable to directly utilize global context for analyzing failed tests, potentially leading to sub-optimal effectiveness [61]. Going beyond mere pre-training to address this limitation, we constructed a bidirectional dataset during this phase, thereby enhancing the model's reasoning capabilities when analyzing failed tests. Following pre-training, we conduct instruction-based fine-tuning for various tasks, enabling the model to analyze failed mobile tests based on different instructions. Finally, we apply preference optimization to the LLMs, which enables more precise control over the model's behavior, ensuring it can identify and emphasize critical aspects of the input that may influence the results. The intuition behind this step is that subtle differences—such as inconsistencies in the error lines within stack traces—can lead to entirely different analytical outcomes when analyzing failed tests. To realize this, we designed a new semantic-based negative sampling strategy specifically for direct preference optimization.

Our evaluation of AnaDroid's performance in a large-scale dataset contains 284k failed mobile testing scripts. This dataset comes from the testing of hundreds of mobile applications, encompassing areas such as security, power consumption, operating systems, contacts, and a wide range of popular third-party applications, supplied by our industrial partner, a global smartphone manufacturer. Each failed test in the dataset includes corresponding test documentation, stack traces, logs, and screenshots. We benchmarked AnaDroid against four baselines on six different LLMs, demonstrating its exceptional efficacy by significantly outperforming all compared methods in within-project and cross-project scenarios. Additionally, a deployment within the production environment testing system of our industrial partner highlighted AnaDroid's practical value.

The main contributions of this paper are as follows:

- We propose AnaDroid, a novel large language model training framework, which contains three training steps and is specifically designed to train the model to effectively and efficiently analyze the failed mobile tests.
- We propose to utilize the bidirectional pre-training and direct preference optimization, which are designed to inject mobile testing knowledge and reasoning capabilities into the model and allow the model to better capture critical aspects of the input context that may significantly influence the output.

Fig. 1. An example of the debugging process for file transfer failures in industrial scenarios.

- We conduct extensive evaluations of AnaDroid on a large-scale dataset to validate it under industrial settings, and the experimental results demonstrate that AnaDroid is capable of effectively classifying failed mobile tests and providing corresponding root cause analysis and repair recommendations.

2 Methodology

2.1 Motivating Example

Figure 1 presents an example from our industrial partner. In this case, the analysis expert receives a test failure notification and reviews the related information. From the documentation, the expert learns that this is a test for a sharing application that transfers a file between two devices over a proprietary protocol. Examining the stack trace, the expert finds that the target device failed to display the receiving window. Further inspection of the screenshot reveals that the window was not being obstructed by other components, such as random pop-up advertisements. Next, the expert checks the logs against the preconditions specified in the documentation, and finds that both devices were connected to a network and had initiated the sharing application's thread; however, the target destination folder shows no file was transferred. This indicates that the stack trace error was caused by the file not being transferred correctly. Considering that the thread had initiated properly and that no transfer-related logs were observed, the most likely reason for the test failure is that the two devices were connected to different networks. Therefore, the expert classifies the failure as a test environment issue, with the root cause being a connection failure between the two devices due to different network environments. The recommended fix is to check the IP addresses and network segments of both devices. In this instance, existing approaches for root cause localization, fault localization, and GUI test script repair would fail to identify or resolve the fault. This is because the root cause of the test failure lies not within the test script or the application code, but in the configuration of the IP addresses and network segments of both devices. Furthermore, the analysis of the failed test requires synthesizing information from multiple sources—including logs, documentation, screenshots, and stack traces—which makes it difficult to adapt previous work to the context of industrial mobile application testing.

In real-world scenarios, smartphone manufacturers typically need to test a large number of applications, both in-house developed and third-party, to verify the functionality and compatibility of different apps on different versions of devices. For example, our industrial partner, a leading smartphone manufacturer, tests not only its apps, such as contacts, messaging, email, and to-do lists, as well as system apps for network and security functions, but also the compatibility of widely

popular third-party apps with the phones they produce. In this context, the continuous expansion of numerous features in commercial applications leads to an increasing number of failed mobile tests that need analysis, along with a growing volume and complexity of the associated information. For instance, our industrial partner, a global smartphone manufacturer, generates approximately 7,000 failed mobile tests in a week, with each test involving tens of thousands of lines of logs and multiple screenshots. The continuous evolution of multimodal information resulting from changes in both applications and tests renders heuristic algorithms based on templates or keywords highly brittle and prone to failure, whereas manual analysis is limited to approximately 1,500 tests per week. The large volume of failed mobile tests requiring analysis, combined with the complex multimodal information involved, poses significant challenges for engineers and creates an urgent need for an automated method of analysis.

2.2 Problem Statement

A failed mobile test, as depicted in Figure 1, includes a test documentation, stack trace, log, and screenshots. The **test documentation** contains the script's name, ID, preconditions, operational steps, and expected results, as well as the application under test and the features being tested. The **stack trace** includes error messages and the call chain. The **log** records running information from the instrumented application during the script's execution, while the **screenshots** capture the interface after certain operations within the script. Formally, let $t = (d, e, L, S)$ be a failed mobile test where d denotes the documentation, e denotes the stack trace, L denotes the log, and S denotes the screenshots. Following the analysis of a failed mobile test t, an expert classifies it into a category c, provides a root cause analysis r, and gives a repair recommendation g. In this context, c is selected from a predefined set of labels, such as requirement change issue or environment issue, while r and g are natural language descriptions. Classification helps engineers efficiently narrow the debugging scope, root cause analysis elucidates the reasons for the test failure, and repair recommendations provide actionable guidance on how to resolve the bug. An analyzed failed mobile test can be denoted as $x = (t, A)$, where $A = (c, r, g)$ denotes the analysis result. The problem we want to address in this paper is *how to effectively and efficiently analyze failed mobile tests to classify the failed tests, give the root cause, and provide repair recommendations in industries.*

Since analyzing failed mobile tests involves complex multimodal information and various failure scenarios, requiring complex reasoning processes, traditional deep learning models may not be sufficient to meet these demands. LLMs [2, 42, 51], which exhibit outstanding performance in logical reasoning [1, 14, 35, 59], provide a promising solution to this problem. However, despite the significant success of general LLMs, their effectiveness in analyzing failed mobile tests still has certain limitations.

First, general LLMs often have little or no domain-specific knowledge of mobile testing, which significantly limits their ability to effectively and efficiently analyze failed mobile tests [17, 39, 40, 62]. Specifically, when analyzing failed mobile tests, the model not only needs to possess general knowledge but also needs to understand the knowledge specific to the mobile application testing domain. More importantly, the model must have the capability to reason from multimodal feature to analyze the failed mobile tests. Unfortunately, the data containing this knowledge can only be collected from the mobile testing domain data or specific mobile applications and is not included in general-purpose training corpora, severely limiting their effectiveness in analyzing failed mobile tests. For example, GPT-5, a SOTA general LLM without exposure to mobile testing knowledge, provides an incorrect analytical conclusion for the case in Figure 1, which classifies the failure as a requirement change issue, determines the root cause to be that the receiving window closed automatically, and recommends revising the script as the solution. **Second**, general LLMs are capable of capturing semantic differences across various natural language inputs, enabling them

to perform general tasks more effectively [42, 51]. However, when analyzing failed mobile tests, subtle contextual differences that lead to vastly different analytical conclusions may stem not only from the semantics of inputs but also from other aspects. For example, in two separate failures of the same test case, a minor difference in the line numbers within the stack trace can point to a completely different analysis result. General LLMs that have not been trained on mobile testing knowledge may fail to capture or understand the specific subtle differences that can cause such variations, particularly when dealing with long-sequence, multimodal inputs [17, 65]. This can result in the model providing consistent outcomes for very similar cases that should lead to different conclusions, thereby reducing the model's effectiveness in analyzing failed mobile tests.

In summary, to effectively and efficiently analyze failed mobile tests to classify their failures, identify the root cause, and provide repair recommendations in the industrial context, we need to address the following challenges:

- **Challenge I: How to inject the mobile testing knowledge and reasoning ability required for analyzing failed mobile tests into the LLMs.**
- **Challenge II: How to enable LLMs to better capture the key part of the context that may significantly impact analysis results.**

2.3 Our Approach

To address the aforementioned challenges, we propose a novel LLM training framework named AnaDroid, designed for effectively analyzing failed mobile tests in industrial settings. Specifically, the design idea of AnaDroid is twofold:

Utilizing bidirectional pre-training to enable the model to learn domain knowledge of mobile testing and how to perform inference based on multimodal features. To enable LLMs to efficiently analyze failed mobile tests, we propose a bidirectional pre-training technique that pre-trains the large model on both forward and backward datasets. This approach allows the model to learn the domain knowledge of mobile testing and enhance its reasoning capabilities based on features from various modalities. The key insight here is twofold. First, the acquisition of knowledge in LLMs primarily occurs during the pre-training phase [16, 69], and the failed test data and analytical results inherently contain a substantial amount of mobile testing knowledge. Pre-training on such data enables LLMs to effectively learn how to analyze failed tests. Second, most current LLMs employ a left-to-right, decoder-only architecture for better natural language representation and understanding. However, this architecture is not well-suited for analyzing failed mobile tests. The reason is that the model's representation of each token only incorporates information from its left side, whereas an effective analysis of failed mobile tests requires consideration of the global context [61]. To mitigate this limitation, we construct a bidirectional dataset by reversing input sequences and conducting pre-training. The intuition is that by pre-training on a bidirectional dataset, the LLM can capture information from both sides of the tokens and develop a more global understanding of each sample, thereby enabling the model to analyze failed mobile tests more efficiently. We will discuss how we pre-train the LLMs in more detail in Section 3.3.1.

Conducting preference optimization to enable the LLMs to better capture the key part of the context that impacts analysis. To enable LLMs to provide more accurate analytical results, it is crucial to ensure that they can capture subtle variations in the input that may have a significant impact on the results. To achieve this objective, we employ preference optimization to enable the model to capture these critical parts in the input. Specifically, preference optimization [13, 42, 45] trains the model to generate outputs that are labeled as more appropriate for a given input while rejecting those labeled as less suitable. This approach allows for more precise control over the model's behavior, aligning it more closely with specific requirements. Through preference optimization, for any two samples with highly similar inputs but different ground truths, we can

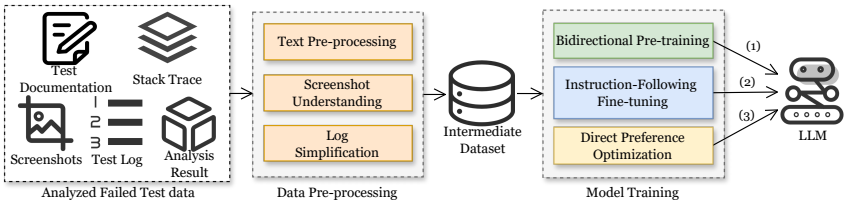


Fig. 2. Overview of AnaDroid.

label the ground truth of one sample as more suitable and mark the ground truth of the other sample as less suitable to train the model. The intuition here is that by using preference optimization to guide the model to output the correct answer while rejecting the answer corresponding to a similar input, we can compel the model to capture the subtle differences in the input that cause variations in the results, thereby enabling it to derive more effective analytical results. The details of the preference optimization will be presented in Section 3.3.3.

3 Design

In this section, we first provide an overview of AnaDroid workflow. Then, we elaborate on the technical details of each component in AnaDroid.

3.1 Overview

Figure 2 presents an overview of AnaDroid, which consists of two main stages: *Data Pre-processing* and *Model Training*. In the *Data Pre-processing* stage, we pre-process the analyzed failed testing data collected from industrial partners to facilitate the subsequent construction of the model's training dataset. Specifically, we perform text pre-processing on stack traces and analysis results to shorten text length, reduce noise in the data, and improve data quality. Then, we conduct screenshot understanding to extract semantic information from screenshots to use as input for the language model to combine visual information. Finally, we perform log simplification to extract key information from logs and reduce their size, mitigating truncation issues due to LLMs' context length limitations. In the *Model Training* stage, AnaDroid further transforms the pre-processed intermediate dataset into different training datasets and trains the model accordingly. **First**, we inject mobile testing knowledge and reasoning capabilities into the LLMs through bidirectional pre-training (BPT), enabling them to acquire the most fundamental analytical abilities. **Second**, we employ instruction-following fine-tuning (FT) to teach the model to adhere to engineers' instructions and expose the knowledge and capabilities learned during the pre-training phase. **Finally**, we utilize direct preference optimization (DPO) to enhance the model's ability to identify critical aspects of the input that may significantly impact the results, thereby enabling it to more effectively classify failed tests, identify the root causes of failures, and provide repair recommendations. Next, we describe each stage and the corresponding training process in detail.

3.2 Data Pre-Processing

In the data pre-processing stage, AnaDroid processes each analyzed failed mobile test $x = (t, A)$ to facilitate the subsequent dataset construction and training stage. Specifically, the data pre-processing stage consists of three steps: *text pre-processing*, *screenshot understanding*, and *log simplification*.

Text Pre-Processing. During text pre-processing, we specifically address stack traces and analysis results. This is because stack traces often contain redundant file paths and framework code, which can increase the length of the input and computational overhead. Additionally, analysis results may

include substantial noise, such as hardcoded IP addresses or meaningless statements like "already fixed." Thus, we first retain only the call chains in the stack trace that are relevant to the mobile test, removing information related to the library functions of the testing framework. Second, we use regular expressions to replace hardcoded data in analysis conclusions with placeholders and create a stopword list to filter out results containing only meaningless text. Note that although test documentation is also in plain text format, we do not pre-process it because it has generally been manually verified by engineers, and we consider it clean data.

Screenshot Understanding. Given the GUI-driven nature of mobile applications and the fact that the success of most tests is judged by observing the GUI [43, 60], incorporating information-rich screenshots into the analysis process can effectively assist in diagnosing failed mobile tests. However, images cannot be directly input into language models. Therefore, we extract the content from the screenshots, converting information from the image modality to the text modality. Specifically, we use the visual model MiniCPM-O [63] to understand the screenshots. MiniCPM-O offers robust visual understanding with fewer parameters, enabling faster inference [9, 63]. More specifically, we first arrange all screenshots S corresponding to the failed test t in chronological order. Then, we input all the screenshots into the model, allowing it to understand the content of each screenshot and output the corresponding descriptions. The primary reason for excluding UI structural information and direct screenshots as inputs is that such data consumes substantial context space within LLMs (or VLMs), potentially leading to attention dispersion and a subsequent decline in reasoning performance. Our experimental results further validate this observation. Consequently, converting screenshots into textual descriptions conserves the context window and enhances the model's comprehension of critical content, thereby boosting its overall analytical effectiveness. Although our method of converting screenshots into textual descriptions may result in some information loss, it conserves context length and ultimately yields superior performance. Selectively incorporating UI structural information is a valuable direction for mitigating context length issues. However, determining how to selectively filter UI elements based on information within the screenshots and other modalities (such as logs) remains a challenging problem. Especially in our problem setting, the localization of the root cause remains unclear, making it difficult to determine how UI structural information should be filtered. Furthermore, discarding certain screenshots might degrade the model's ability to understand testing continuity. Therefore, we adopted the approach of converting all screenshots into textual descriptions to alleviate context length constraints, and we plan to actively explore this research area in future work.

Log Simplification. Logs provide detailed records of information that engineers consider useful for debugging, such as whether an action was executed successfully, making them highly valuable for analyzing failed tests. However, logs are typically large in scale, and directly feeding them into the LLMs would exceed its context length constraints, causing the model to overlook other information [57]. Therefore, it is necessary to pre-process the logs to reduce their size while retaining key information. To achieve this, we compare the log against the document and the error message in the stack traces using semantic similarity. This allows us to extract the log segments that are semantically most relevant to the test case, thereby reducing the log volume. Specifically, we first apply the SOTA log parsing tool Drain [18] to extract developer-written log messages while filtering out metadata such as timestamps. We then utilize SentenceBERT [46] to compute the semantic similarity between each log content and (1) the error messages, as well as (2) each field in the documentation (e.g., preconditions and operation steps). For each log entry, we take the maximum of these similarity scores as its final relevance score, rank all candidates based on this metric, and retain the top-K segments as the simplified log input. By doing so, we can extract semantically relevant information regarding configurations and action executions associated with the test case.

Table 1. The data template of each dataset. *<A>* indicates that the value of *A* is used to fill in template. Due to space constraints, we only present the prompt used for root cause analysis. Replace 'root cause' with 'repair suggestion' or 'classification' to obtain the prompt corresponding to the other tasks.

Pre-training Dataset	
Forward Pre-Training Template	**Text**: The information for the failed test is shown as follows: *<Documentation>*. *<Screenshots>*. *<Log>*. *<Stack Trace>*. The corresponding *root cause* is *<Root Cause>*.
Backward Pre-Training Template	**Text**: The corresponding *root cause* of a failed test is *<Root Cause>*. The information includes: *<Stack Trace>*. *<Log>*. *<Screenshots>*. *<Test Documentation>*.
Instruction-Following Fine-tuning Dataset	
Input: Provide the *root cause* of the failed test based on the following information: *<Test Documentation>*. *<Screenshots>*. *<Log>*. *<Stack Trace>*. The *root cause* is: **Output**: *<Root Cause>*	
Direct Preference Optimization Dataset	
Input: Provide the *root cause* of the failed test based on the following information: *<Test Documentation>*. *<Screenshots>*. *<Log>*. *<Stack Trace>*. The *root cause* is: **Chosen**: *<Ground-truth Root Cause>* **Rejected**: *<Incorrect Root Cause>*	

3.3 Model Training

As discussed in Section 2.2, existing general LLMs face significant challenges in analyzing failed mobile tests due to their lack of mobile testing knowledge. To address this challenge, we designed a model training stage consisting of three steps to inject the required knowledge and reasoning capabilities into the model. Specifically, the model training stage consists of three steps: *bidirectional pre-training*, *instruction-following fine-tuning*, and *direct preference optimization*. Below, we explain more details of each training step.

3.3.1 Bidirectional Pre-Training. We first construct a pre-training dataset and pre-train the LLMs. The key insight here is that mobile testing knowledge and reasoning abilities are crucial for analyzing failed mobile tests, and this knowledge is embedded in the historical data. Thus, we can inject the knowledge required to analyze failed tests and the ability to reason based on input into the LLMs through pre-training, as the learning of knowledge by LLMs primarily occurs during the pre-training phase [16, 17, 69]. By this, we can effectively address the challenge of general LLMs lacking the knowledge and reasoning capabilities, and enable the LLMs to classify failed mobile tests and provide accurate root cause analysis and repair recommendations for failed mobile tests.

However, directly pre-training the LLMs is sub-optimal because current LLMs use a decoder-only architecture, which means the model's representation of each token only reflects knowledge from its preceding context [2, 12, 51]. While this architecture provides stronger language modeling and understanding abilities, it is not ideal for analyzing failed tests because the analysis of failed tests requires considering not just the left-side context but the entire global context [61]. One potential solution is to use other model architectures, such as bidirectional encoder-decoder [11, 34]. However, prior work has shown that decoder-only architectures currently exhibit the best performance in language understanding and reasoning [2, 51], which implies that switching to an alternative architecture could lead to sub-optimal effectiveness in understanding the failed tests and reasoning.

To address this issue, we further introduced a bidirectional pre-training dataset during the pre-training process. As shown in Table 1, we concatenate the data from analyzed failed test cases as context and the analysis results as the answers to form the prompt of the forward pre-training dataset. We then reverse all the information to obtain the prompt of the backward pre-training dataset, as shown in Table 1. The intuition here is that through bidirectional datasets, large models can learn information on both sides of tokens during pre-training and better understand samples

holistically, without sacrificing the architecture's advantage in understanding natural language and reasoning [6]. By doing so, we can improve the effectiveness of LLMs in analyzing failed tests.

Specifically, let $x_p = (t_p, A_p)$ denote the pre-processed analyzed failed test, where t_p and A_p are processed test information and analysis result. Next, we fill the corresponding positions in the bidirectional pre-training template with information from t_p and A_p, resulting in the pre-training data y. Then, we use y to pre-train the model, allowing the LLMs to predict the next token and inject knowledge into the model. More specifically, let $y = [s_1, s_2, \ldots, s_n]$, where s_i denotes the individual tokens, the LLM is then pre-trained via minimizing the following loss function.

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n} \log P(y_i \mid y_{<i}) \tag{1}$$

where $P(y_i \mid y_{<i})$ is the conditional probability of the model generating the i-th token based on the preceding $i-1$ tokens.

3.3.2 Instruction-Following Fine-Tuning. After pre-training, we conduct instruction-following fine-tuning (FT) to enable the LLMs to learn the style of human interaction and expose the knowledge and reasoning capabilities that were already acquired during pre-training [42, 69]. Meanwhile, since we are dealing with three distinct tasks—classification, root cause analysis, and repair recommendation—we need to employ different prompts to inform the model which task it should currently address. Specifically, the template for the FT data is shown in Table 1. The *input* includes the instruction that indicates the task to be completed along with test-related information, while the *output* is the ground truth corresponding to the input. During FT, we feed the *input* into the model and train it to predict the content of the *output*, eventually enabling the model to follow the instructions and expose learned mobile testing knowledge. More specifically, we similarly fill the processed test information and analysis results into the corresponding positions in the FT template to obtain the FT data. Let u denote the *input* sequence and $y = [s_1, s_2, \ldots, s_n]$ denote the *output* sequence. We then fine-tune the LLMs via the following loss function.

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n} \log P(y_i|u, y_{<i}) \tag{2}$$

where $P(y_i|u, y_{<i})$ is the conditional probability of the LLMs generating the i-th token in y based on the u and preceding $i-1$ tokens in y. Note that for the classification task, we follow the practices of prior LLM benchmarks, in which the model is provided with a complete list of predefined categories and is instructed to output the corresponding index number as its classification result [19, 53]. This approach can minimize the generation of extraneous content unrelated to the classification.

3.3.3 Direct Preference Optimization. After FT, we perform direct preference optimization (DPO) on the LLMs to enable them to better capture subtle variations in the input that may lead to different analysis outcomes, thereby enhancing the effectiveness of the LLMs in analyzing failed mobile tests. The key insight here is that by employing preference optimization to make the model output the answer corresponding to the given context and reject answers corresponding to extremely similar contexts, we can compel the model to better capture the result variations caused by contextual differences, thereby enhancing its performance. We utilize DPO instead of reinforcement learning with human feedback (RLHF) [42]. The reason is that RLHF is a complex and unstable process that requires additional training of a reward model, which incurs extra costs [45]. In contrast, DPO consumes fewer computational resources and offers a more stable training process.

Specifically, we first need to generate the corresponding preference optimization dataset. The format of each data in the dataset is shown in Table 1. The *input* is consistent with the FT dataset,

where *chosen* refers to the correct analysis result corresponding to the *input*, and *rejected* refers to the result that needs to be rejected. To generate the *rejected*, we iterate through the entire dataset and calculate the semantic similarity between the current and other data. The data with the highest semantic similarity to the *input*, but where the result has the lowest semantic similarity to the *chosen*, will be selected, and the corresponding result will be treated as the *rejected*. More specifically, let $\mathcal{D}$ denote the pre-processed dataset, and x_p represent an analyzed failed test within it. Then, $\mathcal{D}'_p = \mathcal{D} \setminus \{x_p\}$ denotes the remaining data without x_p. For each sample in $\mathcal{D}'_p$, we calculate a score defined as the ratio S_a/S_i. Here, S_a and S_i denote the semantic similarities between the sample and x_p in terms of their results and inputs, respectively. We employ SentenceBERT, an efficient tool for semantic similarity computation, to obtain these scores. We then use this score to rank the analysis results corresponding to each data x_i in $\mathcal{D}'_p$ and obtain the ascending sequence. Finally, the first element in the sequence that differs from the *chosen* is selected as the *rejected* for x_p. We then conduct DPO on the LLMs via the following loss function [45].

$$\mathcal{L} = -\log \sigma(\beta \log \frac{P_{\theta*}(y_w|u)}{P_{\theta_0}(y_w|u)} - \beta \log \frac{P_{\theta*}(y_l|u)}{P_{\theta_0}(y_l|u)}), \tag{3}$$

where σ is sigmoid function, β is temperature parameter, u denotes the *input* sequence, y_w denotes the *chosen* sequence, and y_l denotes the *rejected* sequence. Let θ^* represent the parameters of the LLM currently being trained, and θ_0 represent the parameters of the reference model (i.e., the LLM before DPO). The probabilities P_{θ^*} and P_{θ_0} are computed using the two respective models. Eq. 3 performs preference optimization by ensuring the LLM has a higher probability of generating y_w and a lower probability of generating y_l compared to the reference model, eventually encouraging the model to favor generating the *chosen* rather than *reject*. By doing so, we enable the model to better capture the subtle differences that can lead to different analytical conclusions, thereby further enhancing its effectiveness in analyzing failed mobile tests.

3.3.4 Inference. After the training process, we can use the LLM for inference. Specifically, for a newly generated failed mobile test, we pre-process it and fill the preprocessed test data into the fine-tuning template to obtain the corresponding *input* prompt. The model will output the corresponding answer by feeding the prompt into the model. More specifically, let t represent a failed test and $\mathcal{F}$ denote the trained LLM. We then populate the fine-tuning template with the t to generate the corresponding prompts u_c, u_r, and u_g for the classification, root cause analysis, and repair recommendation, respectively. Finally, $\mathcal{F}(u_c)$, $\mathcal{F}(u_p)$, and $\mathcal{F}(u_g)$ are taken as the model's outputs for the classification, root cause analysis, and repair recommendation, respectively.

4 Evaluation

We evaluate AnaDroid on the following research questions:

- RQ1: How does AnaDroid's effectiveness compare to baselines on failed mobile test classification, root cause analysis, and repair recommendations?
- RQ2: How does AnaDroid perform in the cross-project scenario?
- RQ3: How do different variants of AnaDroid perform (i.e., ablation studies on different modules and features)?
- RQ4: How does AnaDroid perform when deployed in an actual production environment?
- RQ5: What are the time costs of AnaDroid?

4.1 Evaluation Setup

Benchmark. To answer the RQs, we collaborated with our industrial partner, which operates multiple smartphone product lines worldwide. They provided us with failed mobile tests generated

during the unit test of different mobile applications. The mobile tests cover functionalities such as operating systems, power consumption, security, networking, various applications, and third-party application compatibility. Each failed test includes the corresponding documentation, stack traces, screenshots, logs, and analysis results. Following data pre-processing and deduplication, we have 284,552 failed tests across 278 applications, averaging 1,023 failed tests per application. The engineers classified these test failures into five categories: Test Framework Issues (4.78%), Environment Issues (22.55%), Script/Test Case Issues (44.12%), Requirement Change Issues (13.15%), and Implementation Issues (15.37%). We provide a detailed explanation of each category on our project website due to space limitations. Since the test data involves defects in commercial mobile devices and software, we have anonymized all the data. Specifically, we anonymized the names of all tested applications and test scripts, retaining only the categories of the applications and the functionalities and identifiers of the test scripts. Additionally, we have released some sanitized test cases as reference examples. All data are included in our project website. Note that the effectiveness of AnaDroid is not dependent on a specific dataset, as mobile testing processes universally generate corresponding screenshots, logs, stack traces, and documentation related to test scripts. The large-scale dataset we used originates from a worldwide smartphone manufacturer, encompassing a broad range of tested applications and testing functionalities, ensuring sufficient generalizability. Therefore, we believe that AnaDroid can also be adapted to other mobile testing datasets.

Baselines. To the best of our knowledge, we are the first to attempt training LLMs to perform failure classification, root cause analysis, and provide repair recommendations for failed mobile tests. Consequently, we conduct comparative experiments using vanilla LLMs, RAG-enhanced LLMs, and various variants of AnaDroid. In addition to LLM-based approaches, we compare our method against traditional neural models, namely Seq2Seq [49] and Transformer [52]. The vanilla baseline refers to the direct application of LLMs for inference without prior exposure to the dataset. For RAG, we selected three retrieval mechanisms, namely SentenceBERT [46], DPR [23], and ColBERT [24]. Note that due to space constraints, the table only presents the best-performing retrieval mechanism (i.e., DPR) for RAG. The complete results have been made available in the `data` directory of the anonymous repository.

Models. To ensure a comprehensive evaluation, we incorporate models of diverse architectures and parameter scales, namely ChatGLM3-6B-Chat (ChatGLM-6B), Qwen2.5-7B-Instruction (Qwen-7B), Qwen2.5-32B-Instruction (Qwen-32B), and DeepSeek-R1-32B (DeepSeek-32B). We also evaluate Vision-Language Models (VLMs), specifically Qwen2.5-VL-7B-Instruction (Qwen-VL-7B), Qwen2.5-VL-32B-Instruction (Qwen-VL-32B), and Qwen3-VL-30B-Instruction (Qwen3-VL-30B). Due to computational resource constraints, Qwen-VL-32B is exclusively utilized for the vanilla and RAG configurations. Unless specifically stated, the experiments were conducted using ChatGLM-6B as a base LLM. Due to data security concerns—which preclude the use of third-party LLMs like GPT-5—and inherent computational constraints in industrial settings, our research focuses on smaller, locally deployable models and single-turn dialogue. Even within large corporations, available resources are still limited. Therefore, enterprises aim to address this problem by utilizing the most efficient resources and models.

Evaluation Metric. For the failure classification (CLS) task, we directly use classification accuracy and Macro-F1 as our evaluation metrics. Macro-F1 [41] is designed to evaluate the average performance of a model across all classes under multi-class and class-imbalanced scenarios. As discussed in the introduction, our scenario involves more complex root causes, such as network issues and hardware malfunctions, which require an expert to analyze the failed test and assign rectification tasks to different personnel. Therefore, our primary objective is to generate the root cause analysis (RCA) and repair recommendations (RR) in natural language. Therefore, we adopted widely used metrics from natural language generation tasks [49, 52, 58]. Specifically, we use BLEU-4 [44],

ROUGE-L [33], and SentenceBERT with Cosine Similarity [48] as evaluation metrics. BLEU-4 and ROUGE-L are text similarity metrics that compare the count of n-grams in the generated result against the ground truth. SentenceBERT with Cosine Similarity (SBCS) is a semantic similarity metric that uses SentenceBERT to compute the embeddings of both the generated result and ground truth, then calculates their cosine similarity. The higher these three metrics, the closer the model's output is to the correct result, indicating better model performance. To further evaluate the performance of RCA and RR, we conducted a human evaluation (HE) involving engineers from our industrial partner who manually assessed the model outputs. Following established sampling guidelines [7, 25], we selected 384 samples to achieve a 95% confidence level with a 5% margin of error. To ensure objectivity, each sample was independently evaluated by four professional engineers, with inter-rater reliability assessed via Fleiss' Kappa [15], as it generalizes Cohen's Kappa [8] for scenarios involving more than two raters. The resulting score of $\kappa = 0.73$ indicates *substantial agreement* according to standard criteria [26]. Any remaining disagreements were subsequently resolved through consensus-building discussions.

Evaluation Scenario. RQ1 investigates the performance of the AnaDroid in a within-project setting, where data from different applications populate both the training and testing sets. In this scenario, we employ ten-fold cross-validation, where each iteration uses nine folds for training and the remaining one fold for evaluation. In practice, new applications not included in the dataset will be developed. Therefore, we need to evaluate the performance of AnaDroid in a cross-project scenario in RQ2. In this RQ, we utilize the ten-fold cross-validation, which uses the model to analyze all failed tests of applications from one fold while training the model on data of applications from the remaining nine folds. In addition to RQ1 and RQ2, in RQ4 we evaluated the performance of AnaDroid in a real-world production environment. Specifically, we deployed the model into the testing system of our industrial partner and manually assessed its effectiveness during the first month of deployment.

Implementation. We use Llama-Factory [68], an open-source framework that enables efficient training of LLMs, to implement AnaDroid. We employ LoRA [20] in all training processes to improve training efficiency. For the parameters in AnaDroid, We employed grid search to determine the optimal parameter combination. Due to space limitations, all hyperparameters and detailed training configurations have been included on our project website. Since our models were trained on specific defect data from the company, we are unable to open-source them to prevent potential reverse engineering attacks. All experiments are conducted on a workstation with Intel(R) Xeon(R) Gold 6248R CPU, 144 GB memory, and two Nvidia A100 GPUs, running Ubuntu 22.04.

4.2 RQ1: Effectiveness

The objective of this RQ is to evaluate the effectiveness of AnaDroid. We conduct a comparative analysis of AnaDroid against four baselines using different models outlined in Section 4.1 to assess their performance on the tasks of classification, root cause analysis, and repair recommendations. Results are detailed in Table 2.

Analysis of Table 2 yields notable insights: First, existing general LLMs are almost incapable of failure classification and providing effective root cause analysis and repair recommendations for failed mobile tests. In particular, the base LLMs, including the 32B model, achieved the lowest performance on different metrics for both tasks. Despite leveraging RAG for context augmentation, small-scale general LLMs underperform relative to traditional neural architectures. For models below 10B parameters, RAG exhibits a significant performance gap in RCA and RR tasks, trailing Transformers by 12.84% and 12.9% in average HE scores. This performance deficit persists even for some LLMs beyond the 10B threshold, where RAG still fails to match Transformer models in human

Table 2. Comparison with Base and RAG at three tasks. CLS, RCA, and RR represent classification, root cause analysis, and repair recommendation, respectively; SBCS represents the cosine-based semantic similarity calculated by SentenceBERT.

Techniques	Models	CLS		RCA				RR			
		ACC	Macro-F1	BLEU	ROUGE	SBCS	HE	BLEU	ROUGE	SBCS	HE
Traditional Neural Models											
Seq2Seq	/	40.13	30.07	30.39	39.74	50.53	47.26	31.78	42.17	51.27	44.29
Transformer	/	42.77	31.74	33.74	44.15	52.17	45.38	34.46	45.53	52.88	46.65
Open Source Model (<10B)											
Vanilla	ChatGLM-6B	16.42	12.35	1.43	5.94	23.47	10.74	0.67	2.53	26.21	11.16
	Qwen-7B	15.38	11.67	1.56	5.38	24.11	10.92	0.35	1.51	24.63	10.47
	Qwen-VL-7B	12.74	9.57	1.04	4.11	19.52	9.92	0.33	1.27	18.38	9.3
RAG	ChatGLM-6B	39.96	32.17	23.29	33.26	49.88	39.29	23.37	32.87	47.88	37.78
	Qwen-7B	41.27	33.61	22.41	35.42	47.82	39.55	23.59	34.61	49.35	40.63
	Qwen-VL-7B	36.41	28.74	19.87	27.33	40.13	32.47	17.67	28.72	42.09	32.91
AnaDroid	**ChatGLM-6B**	**81.32**	**74.69**	**62.77**	**73.27**	**72.38**	**77.48**	**73.92**	**78.44**	**80.42**	**80.82**
	Qwen-7B	**82.16**	**75.37**	**65.43**	**75.15**	**74.26**	**78.52**	**71.33**	**75.16**	**79.83**	**81.34**
	Qwen-VL-7B	**77.45**	**69.88**	**60.52**	**71.38**	**70.17**	**72.43**	**68.49**	**72.85**	**77.34**	**75.26**
Open Source Model (>10B)											
Vanilla	Qwen-32B	23.36	17.92	2.06	7.59	29.21	13.55	1.32	3.84	28.53	18.12
	DeepSeek-32B	22.47	17.34	1.94	7.13	28.84	11.39	1.07	3.32	28.47	18.44
	Qwen-VL-32B	22.83	17.51	1.88	6.84	27.63	13.42	0.91	3.13	27.69	16.39
	Qwen3-VL-30B	24.32	19.4	4.23	11.53	29.68	21.83	2.26	7.44	30.04	23.55
RAG	Qwen-32B	50.82	41.75	29.59	40.62	51.1	43.04	28.78	35.29	51.88	42.62
	DeepSeek-32B	47.73	39.02	29.41	40.52	50.73	45.59	31.83	37.66	53.04	47.42
	Qwen-VL-32B	46.53	37.86	24.85	33.87	45.13	39.07	24.41	34.92	47.51	39.64
	Qwen3-VL-30B	46.98	37.88	25.11	34.42	45.79	39.28	24.78	35.38	48.41	40.05
AnaDroid	**Qwen-32B**	**86.39**	**83.14**	**73.47**	**81.86**	**82.45**	**82.62**	**80.32**	**86.48**	**85.77**	**84.16**
	DeepSeek-32B	**87.34**	**83.86**	**74.52**	**82.3**	**82.87**	**82.43**	**80.74**	**87.52**	**86.45**	**84.37**

evaluations. These results indicate that general-purpose LLMs only achieve inferior effectiveness in analyzing the failed mobile application tests.

Second, AnaDroid consistently outperforms all compared baselines across various tasks and metrics, regardless of the model scale. Specifically, for LLMs with fewer than 10B parameters, AnaDroid achieves accuracy improvements ranging from 87.7% to 104.8% and Macro-F1 improvements from 130.9% to 143.8% over Seq2Seq, Transformer, and RAG in classification. Furthermore, in the human evaluation of RCA and RR, AnaDroid surpasses the top-performing baseline by an average margin of 64.3% and 74.0%, respectively. In RAG, DPR proved to be the most effective. In the HE evaluation for the RCA task, using Qwen-7B as the base model, AnaDroid achieved relative improvements of 98.5%, 111.2%, and 176.1% over DPR, ColBERT, and SentenceBERT, respectively. The experimental results underscore the effectiveness of AnaDroid. Through a customized training strategy that incorporates mobile testing knowledge and enables the LLMs to capture critical contextual information, AnaDroid substantially boosts the performance of LLMs in the analysis of failed mobile tests.

Third, we found that scaling up the model size does not significantly enhance the effectiveness of both Vanilla and RAG. This observation suggests that when a model lacks the requisite knowledge, simply increasing its scale is insufficient to achieve satisfactory performance, further underscoring the necessity of developing tailored training strategies. Moreover, our experimental results indicate that utilizing all raw screenshots for end-to-end reasoning with VLMs actually leads to a decline in overall effectiveness. Specifically, in the RCA task, AnaDroid integrated with Qwen-7B

Table 3. Cross-project effectiveness on different techniques. CLS, RCA, and RR represent classification, root cause analysis, and repair recommendation, respectively; SBCS represents the cosine-based semantic similarity calculated by SentenceBERT.

Techniques	Models	CLS		RCA				RR			
		ACC	Macro-F1	BLEU	ROUGE	SBCS	HE	BLEU	ROUGE	SBCS	HE
Traditional Neural Models											
Seq2Seq	/	27.59	20.77	12.58	21.64	35.18	24.3	15.33	24.91	37.44	22.78
Transformer	/	30.4	23.49	14.06	22.83	36.33	24.82	18.93	30.32	40.25	23.36
Open Source Model (<10B)											
RAG	ChatGLM-6B	24.52	17.94	6.47	12.46	29.62	19.53	4.97	11.34	30.59	16.74
	Qwen-7B	26.77	19.78	8.29	14.28	28.44	19.14	5.57	11.96	29.86	17.2
	Qwen-VL-7B	20.16	14.31	7.72	13.92	28.73	19.22	5.23	11.82	27.33	16.38
AnaDroid	**ChatGLM-6B**	**53.74**	**40.68**	**40.83**	**55.18**	**62.47**	**58.74**	**46.22**	**60.67**	**66.82**	**62.27**
	Qwen-7B	**54.33**	**41.24**	**41.24**	**56.2**	**64.05**	**60.28**	**48.51**	**62.74**	**68.49**	**64.54**
	Qwen-VL-7B	**50.14**	**36.11**	**38.66**	**52.87**	**60.32**	**55.35**	**42.63**	**57.11**	**63.25**	**58.6**
Open Source Model (>10B)											
RAG	Qwen-32B	32.77	25.87	14.09	22.8	38.68	28.74	16.58	25.24	37.32	31.52
	DeepSeek-32B	33.45	26.23	17.52	26.69	39.07	30.81	18.14	28.39	38.97	33.62
	Qwen-VL-32B	26.59	20.65	13.24	21.55	35.48	24.62	12.08	20.49	33.47	26.54
	Qwen3-VL-30B	26.63	20.72	13.95	22.12	35.93	24.77	12.13	20.64	34.03	26.41
AnaDroid	**Qwen-32B**	**61.5**	**51.33**	**48.36**	**60.47**	**66.59**	**64.72**	**51.58**	**67.24**	**70.36**	**68.83**
	DeepSeek-32B	**62.61**	**52.49**	**48.82**	**61.12**	**66.72**	**64.78**	**51.74**	**67.13**	**70.21**	**68.79**

outperformed Qwen-VL-7B by 8.4% in terms of HE. Notably, even Qwen2.5-VL-32B yields worse performance than the text-only Qwen2.5-7B optimized by AnaDroid. Furthermore, the performance of the state-of-the-art Qwen3-VL-30B still fails to surpass that of our AnaDroid-optimized Qwen2.5-7B baseline. A plausible explanation is that processing multiple raw screenshots rapidly inflates the context length, which induces attention dilution and consequently impairs the model's reasoning capabilities. Thus, we convert screenshots into descriptions in AnaDroid rather than directly input into the VLMs.

4.3 RQ2: Cross-Project Effectiveness

In RQ2, we further evaluate the effectiveness of AnaDroid in a cross-project scenario. Table 3 presents the comparison results between AnaDroid and different baselines. Note that for the Vanilla method, the results for cross-project and within-project scenarios are identical, as both involve directly inputting the data requiring inference into the model without utilizing the training set. From Table 2 and Table 3, we observe that the performance of traditional neural models, RAG, and AnaDroid experience a certain degree of decline in the cross-project scenario. For example, there is a 24.18% decrease in the human evaluation of AnaDroid in RCA. This is reasonable because, in a cross-project scenario, the screenshots, logs, and other features involved in failed tests differ significantly from the distribution in the training set. Furthermore, we observe that traditional neural models exhibit the most significant performance degradation. Specifically, the performance of Seq2Seq and Transformer on the HE of RCA tasks decreased by 48.5% and 45.3%, respectively. This decline may be attributed to the insufficient capability and capacity inherent in traditional models. Despite a performance decline, AnaDroid still outperforms other baselines. For instance, AnaDroid with Qwen-7B shows improvements of 62.42% and 57.22% in accuracy and Macro-F1 compared with the strongest baseline, RAG with DeepSeek-32B. Similar trends are observable across other tasks and metrics. This result demonstrates that the pre-training and direct preference optimization stages enable the model to more effectively learn how to reason from

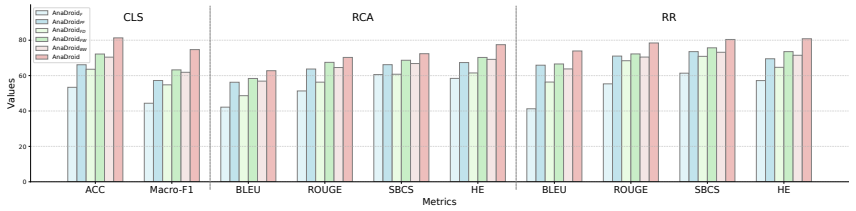


Fig. 3. Ablation study of different components.

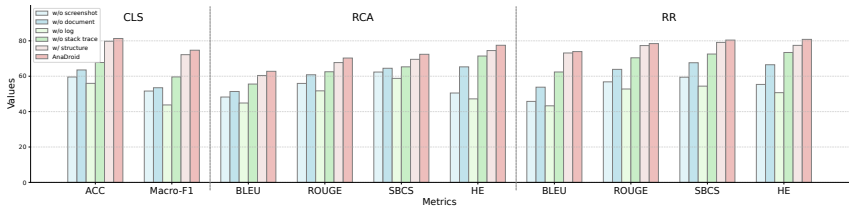


Fig. 4. Ablation study of different features.

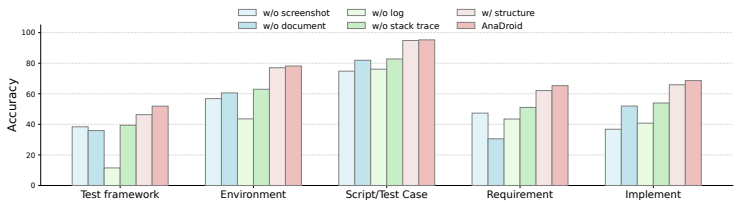


Fig. 5. Per-modality contribution across failure types.

Table 4. Comparison of cross-project RCA performance between AnaDroid and AnaDroid w/ RAG.

Method	BLEU	ROUGE	SBCS	HE
AnaDroid	41.24	56.20	64.05	60.28
AnaDroid w/ RAG	47.65	59.44	67.49	66.52

Table 5. Comparison between our reversal strategy with sliding window attention.

Strategy	BLEU	ROUGE	SBCS	HE
SWA	57.36	64.73	67.61	70.79
ReversalStrategy	65.43	73.15	74.26	78.52

the available data, thereby enhancing its generalization ability in cross-project scenarios. We also introduced few-shot examples from a new project into the Qwen-7B model trained via AnaDroid to evaluate to what extent these examples can improve cross-project RCA performance during inference. Specifically, with the integration of RAG, the Qwen-7B model trained by AnaDroid achieved BLEU, ROUGE, SBCS, and HE scores of 47.65, 59.44, 67.49, and 66.52, respectively, on the RCA task. From the experimental results, we observed that this approach yielded an additional relative improvement of approximately 10.3% in the human evaluation metrics for RCA.

4.4 RQ3: Ablation Study

Effect of Different Components. In particular, we consider the following five variants of AnaDroid: AnaDroid$_F$ only conducts fine-tuning. AnaDroid$_{PF}$ removes the DPO. AnaDroid$_{FD}$ removes the bidirectional pre-training. AnaDroid$_{FW}$ removes the backward dataset in pre-training. AnaDroid$_{BW}$ removes the forward dataset in pre-training. Figure 3 shows the experimental results,

Table 6. Ablation study of DPO negative example selection strategy.

RCA	BLEU	ROUGE	SBCS	HE
SentenceBERT	65.43	73.15	74.26	78.52
Random	37.42	50.52	58.23	61.82
EditDistance	47.25	60.37	62.82	67.87

Table 7. Sensitivity analysis of log lengths.

Length Range	CLS	RCA	RR
	Accuracy	HE	HE
< 5000	82.37	78.66	81.52
5000 – 15000	82.06	78.31	81.44
> 15000	82.25	78.43	81.26

and we have the following observations. First, removing any training stage in AnaDroid leads to a decline in effectiveness. This indicates that each training stage of AnaDroid contributes to its effectiveness. Second, when comparing AnaDroid$_F$, AnaDroid$_{PF}$, AnaDroid$_{FD}$, we found that pre-training is the most important stage among the three training processes, as the variants without pre-training showed the largest performance loss. Specifically, AnaDroid$_{PF}$ achieves an improvement of 15.3% and 9.6% in HE on the RCA task compared to AnaDroid$_F$ and AnaDroid$_{FD}$, respectively. This is because pre-training is the most crucial process for injecting knowledge into LLMs, effectively improving the model's effectiveness in analyzing failed tests. Furthermore, AnaDroid$_{PF}$ achieves a 13.98% reduction in HE for the RR task compared to AnaDroid. This indicates that DPO enables the model to more effectively capture critical information within the input, thereby enhancing its overall effectiveness. Third, by comparing AnaDroid$_{FW}$ and AnaDroid$_{BW}$, we observed that both forward and backward data contribute to performance improvements. This indicates that our proposed bidirectional pre-training can, to a certain extent, enhance the model's performance in analyzing failed tests.

Effect of Different Features. We also conducted ablation studies on different features. Specifically, we considered the following five variants: excluding screenshots, excluding documentation, excluding logs, excluding stack traces, and incorporating UI structural information. Figure 4 shows the experimental results, and we have the following observations. First, each feature contributes to the overall performance. Specifically, the accuracy of classification follows an ascending order across the variants, excluding logs, screenshots, documentation, and stack traces, respectively. Similar trends are observed across other metrics. These results indicate that the relative importance of these features, from highest to lowest, is as follows: logs, screenshots, documentation, and stack traces. Second, we observed that incorporating UI structural information leads to a marginal decline in the model's effectiveness. For instance, on the RCA task, the performance in terms of HE decreased by 3.84% after adding structural information. This may be attributed to the inherent complexity of UI structures; moreover, when multiple screenshots are involved, the simultaneous insertion of structural information for each interface significantly increases the context length. Recent studies [17, 65] indicate that such expansion can lead to attention dispersion, thereby degrading the model's performance during inference.

We also evaluate the per-modality contribution across failure types. We evaluate this by measuring the model's classification performance across different failure types after removing each modality. From the experimental results in Figure 5, we observe the following: First, removing screenshot information causes the most significant drop in the model's classification accuracy for script/test case issues and implementation issues. This indicates that the textual information derived from screenshots effectively assists the model in evaluating actions or behaviors within the scripts, as well as identifying potential flaws in the application's implementation. Second, the removal of log information results in the most pronounced degradation in classification performance for testing framework and test environment issues. This is likely because logs contain extensive records regarding the framework and environment; consequently, their removal leads to a sharp decline in performance across these two categories. Third, documentation is crucial for the model

Table 8. Practical evaluation results of AnaDroid-trained ChatGLM3 during a one-month deployment period.

CLS		RCA	RR
ACC	Macro-F1	HE	HE
84.38	78.28	78.44	81.83

Table 9. Training time cost of different training stages.

Training Stage	PT	FT	DPO	TOTAL
Training Time	8.85 h	7.31 h	14.28 h	30.44 h

to determine whether an issue stems from requirement changes, as it records detailed testing specifications. Finally, introducing structural information does not yield performance improvements in any specific issue category, which aligns with our previous analysis that incorporating extra context can distract the model's attention.

Comparison of the Sequence Reversal Strategy with Sliding Window Attention. We conducted an ablation study on the RCA task using Qwen2.5-7B to compare the sequence reversal strategy with sliding window attention (SWA). The experimental results are presented in Table 5. From the table, we observe that the reversal strategy is more effective than sliding window attention. A plausible explanation is that the reversal strategy operates at the data level to encourage the model to learn associations in both directions, thereby better enhancing its reasoning capabilities on domain-specific problems. In contrast, sliding window attention requires restricting the attention window for each token; while this extends the supported context length, it may inadvertently compromise the model's reasoning ability.

Effect of the DPO Negative-Example Selection Strategy. We compared three negative-example selection strategies for DPO on Qwen2.5-7B and root cause analysis (RCA): SentenceBERT-based semantic retrieval, random sampling, and edit-distance-based selection (Levenshtein distance). Table 6 shows the experimental results. From the table, we observe that under the same DPO training framework, the SentenceBERT-based strategy improved human evaluation scores for RCA by 27.0% over random sampling, and by 15.6% over edit-distance-based selection. We further observed that random sampling can introduce noisy pairs and even degrade overall performance. Conversely, edit-distance-based selection is highly susceptible to superficial lexical differences rather than capturing true semantic relevance. These empirical observations directly motivated our decision to utilize SentenceBERT for constructing negative examples.

Sensitivity Analysis of Log Lengths. We categorized the applications into three groups based on their log lengths: fewer than 5,000 lines, between 5,000 and 15,000 lines, and exceeding 15,000 lines. Subsequently, we evaluated the same top-k setting on each group independently using Qwen2.5-7B. Table 7 shows the experiment results, and we find that the maximum variation gap across the three log-length groups is merely 0.35 percentage points in RCA, 0.26 percentage points in repair recommendation, and 0.31 percentage points in failure classification. These minimal variances indicate that our log simplification strategy remains stable across different log-length distributions.

4.5 RQ4: Practical Evaluation

To further evaluate the effectiveness of AnaDroid, we deployed an LLM trained with our framework into the testing system of our industrial partner to assess its performance in a real-world production environment. Specifically, we collected the outputs from the LLM during its first month of deployment and had the partner's engineers manually evaluate their accuracy. In total, we collected 11,304 failed mobile tests for this evaluation. Noted that during the deployment period, the executed mobile tests involved new application versions and features. Consequently, the data distribution encountered during deployment deviated from that of our original training set. Table 8

presents the results of the manual evaluation. From the table, we can observe that the model trained with AnaDroid achieved promising results in this practical deployment. In particular, AnaDroid achieved an accuracy of 84.38% on the failure classification task. For root cause analysis and repair recommendations, the model achieved accuracies of 78% and 81%, respectively, based on the manual evaluation by our partner's engineers. These results confirm the effectiveness of AnaDroid in a real-world setting and demonstrate its generalization capabilities. Complex mobile testing in industrial settings often involves analyzing a vast number of failures, each potentially containing tens of thousands of log lines and several screenshots, posing a significant challenge for engineers. We also measured the impact on the engineer analysis efficiency before and after the model's deployment. Before deployment, an engineer analyzed an average of 8.5 failed tests per day; this number increased to 16.3 per day after the model was deployed. This result demonstrates that AnaDroid can effectively enhance the analytical efficiency of engineers. Through an informal interview, developers confirmed the practical usefulness of AnaDroid.

4.6 RQ5: Efficiency

This RQ empirically analyzes the efficiency of AnaDroid. Table 9 presents the time cost of each training stage of AnaDroid. As shown in this table, the complete training on the full dataset (284k) takes approximately 30 hours, with the direct preference optimization process taking the most time, nearly half of the total duration. This is because the optimization process is more complex, requiring the model to output *chosen* and not output *reject*. Nevertheless, it is important to emphasize that AnaDroid achieves significantly better performance in analyzing failed tests than the baselines, as demonstrated in Section 4.2. Therefore, we believe the time cost of AnaDroid is worthwhile for achieving better models. Furthermore, considering that the training process is offline, the training time of AnaDroid is also acceptable in practice.

5 Discussion

Threats to Validity. The main threat to *internal validity* lies in the technical implementation of AnaDroid, the compared approaches, and experimental scripts. To mitigate this threat, we developed AnaDroid based on widely used libraries and employed the original implementations of the compared approaches and base models. We have reviewed the source code of AnaDroid and experimental scripts thoroughly. The main threat to *external validity* may be tied to the benchmark used in our study. To reduce this threat, we perform experiments on an industrial dataset provided by our industrial partner, which contains 284k failed tests from 278 software and covers diverse functionalities. Furthermore, we compare AnaDroid with four baselines across six open-source LLMs in our experiments. The main threat to *construct validity* lies in the parameters in AnaDroid and metrics used in experiments. To mitigate this threat, we present the detailed parameter settings on our project site. To reduce the threat from metrics employed, we employed various metrics that are widely used in prior text generation studies [49, 58] and multi-class classification [41].

Generalizability of AnaDroid. AnaDroid has demonstrated remarkable effectiveness in analyzing failed mobile tests. Nevertheless, we believe its key insights can also be further generalized to other domains in the future. The generalizability of AnaDroid comes from two aspects. First, our experiments demonstrate that models trained with AnaDroid can effectively and automatically analyze failed mobile tests in industrial scenarios, achieving promising results. The large-scale dataset we used contains 284K samples, covering over 200 applications and a diverse range of error types, thereby spanning a wide spectrum of mobile application testing scenarios. Therefore, we believe the framework provided by AnaDroid can also be applied to other mobile application testing datasets from industrial contexts. Second, the key insight of AnaDroid concerns how to preprocess data and use it to train LLM for solving analysis problems, in a manner that is decoupled

from specific platforms or UI frameworks. Thus, even if new UI standards, testing frameworks, log formats, or other platforms similar to mobile devices (such as smart TVs) emerge in the future, the principles of AnaDroid can still be applied. By using relevant, problem-specific datasets, LLMs can be trained to address the unique challenges of these new environments.

6 Related Work

LLM for Mobile Testing. As LLMs exhibit outstanding performance in natural language understanding and logical reasoning, SE researchers have widely applied them to various SE tasks, achieving significant progress. Despite the extensive application of LLMs in mobile testing, these efforts have predominantly focused on GUI testing rather than the automated analysis of mobile test failures—specifically, classifying failures and providing root cause analysis and repair suggestions in natural language. Consequently, existing research on LLMs for GUI testing is difficult to adapt to our task. To date, existing LLM-based research has primarily concentrated on automatic GUI testing [35], test script migration [3, 67], test script generation [29, 64], bug report reproduction [14, 21], and text input generation [36, 37]. To the best of our knowledge, no prior work has utilized LLMs to classify failures in mobile testing and to generate root cause analyses and repair recommendations in natural language. AnaDroid is the first to utilize three different training tasks in LLM for mobile testing, injecting the knowledge and reasoning abilities required for analyzing failed tests into the LLMs, enabling it to effectively classify failure, provide root cause analysis, and give repair recommendations.

Specialized LLM. Training specialized LLMs has been widely applied in various fields, such as finance [54, 62], law [10, 40], code [39, 47], and IT operations [17]. The most common approach to training LLMs in specific domains is to fine-tune a pre-trained general LLM using a small domain-specific dataset. LawGPT [40] fine-tunes GPT-3 using legal domain data, enabling the model to answer legal questions, generate legal documents, and provide legal advice. Additionally, another common strategy is to continue pre-training the LLM on domain-specific data. CodeLlama [47] pre-trains the model through code infilling tasks, such as predicting missing parts of programs, thereby training a large code model capable of code completion, type inference, and more. BloombergGPT [54] collects English financial documents from the past 20 years, including news, filings, and press releases, and uses this data as a training corpus to pre-train a LLM for tasks such as sentiment analysis and news classification. To the best of our knowledge, we are the first to employ three distinct training processes on domain-specific tasks to enhance the model's reasoning capabilities on domain data, and also the first to leverage a specialized LLM to address the problem of failed mobile test analysis.

7 Conclusion

In this paper, we propose AnaDroid, a novel LLM training framework, designed to effectively analyze the failed mobile tests to classify failure, provide root cause analysis and repair recommendations. The training process of AnaDroid consists of three steps. First, we inject domain-specific knowledge into the LLM through pre-training and a bidirectional dataset, enhancing the model's reasoning ability for domain-specific tasks. Second, we fine-tune the model through instruction-following to expose the knowledge and capabilities acquired during pre-training and teach the model the desired interaction style. Finally, we apply direct preference optimization to help the model better capture key contents in the input that could significantly influence the analysis results, further improving the LLM's effectiveness in analyzing failed tests. Our experimental results show that after training with AnaDroid, the LLM's performance in providing root cause analysis and repair recommendations is significantly improved, demonstrating the effectiveness of AnaDroid.

8 Data Availability

AnaDroid is publicly available at https://github.com/pppppkun/AnaDroid/. The complete RAG experimental results are located in the `data` directory.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This work is supported in part by the National Natural Science Foundation of China under Grant Nos. 62372227 and 62402214, the China Postdoctoral Science Foundation under Grant Nos. 2025T180420 and 2025M781473, the Postgraduate Research & Practice Innovation Program of Jiangsu Province under Grant No. KYCX25_0368, the "111 Center" under Grant No. B26023, and the Nanjing University PhD Student Zhujian Program. Zhong Li is additionally supported by the Postdoctoral Fellowship Program of CPSF under Grant No. GZB20250386 and the Jiangsu Funding Program for Excellent Postdoctoral Talent under Grant No. 2025ZB317.

References

[1] Islem Bouzenia, Premkumar T. Devanbu, and Michael Pradel. 2025. RepairAgent: An Autonomous, LLM-Based Agent for Program Repair. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 2188–2200. https://doi.org/10.1109/ICSE55347.2025.00157

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[3] Shaoheng Cao, Minxue Pan, Yuanhong Lan, and Xuandong Li. 2025. Intention-Based GUI Test Migration for Mobile Apps using Large Language Models. *Proc. ACM Softw. Eng.* 2, ISSTA (2025), 2296–2318. https://doi.org/10.1145/3728978

[4] Shaoheng Cao, Minxue Pan, Yu Pei, Wenhua Yang, Tian Zhang, Linzhang Wang, and Xuandong Li. 2024. Comprehensive Semantic Repair of Obsolete GUI Test Scripts for Mobile Applications. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 90:1–90:13. https://doi.org/10.1145/3597503.3639108

[5] An Ran Chen, Tse-Hsun (Peter) Chen, and Junjie Chen. 2022. How Useful is Code Change Information for Fault Localization in Continuous Integration?. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 52:1–52:12. https://doi.org/10.1145/3551349.3556931

[6] Justin Chih-Yao Chen, Zifeng Wang, Hamid Palangi, Rujun Han, Sayna Ebrahimi, Long T. Le, Vincent Perot, Swaroop Mishra, Mohit Bansal, Chen-Yu Lee, and Tomas Pfister. 2024. Reverse Thinking Makes LLMs Stronger Reasoners. *CoRR* abs/2411.19865 (2024). https://doi.org/10.48550/ARXIV.2411.19865 arXiv:2411.19865

[7] William G Cochran. 1977. *Sampling techniques* (3rd ed.). John Wiley & Sons.

[8] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46. https://doi.org/10.1177/001316446002000104

[9] OpenCompass Contributors. 2023. OpenCompass: A Universal Evaluation Platform for Foundation Models. https://github.com/open-compass/opencompass.

[10] Jiaxi Cui, Munan Ning, Zongjian Li, Hao Li, Yang Ya, Bohua Chen, Bin Ling, Yonghong Tian, and Li Yuan. 2026. Chatlaw: A Multi-Agent Legal Assistant based on a Role-Aligned Mixture-of-Experts Architecture. *Fundamental Research* (2026). https://doi.org/10.1016/j.fmre.2026.03.026

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/V1/N19-1423

[12] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. GLM: General Language Model Pretraining with Autoregressive Blank Infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*. Association for Computational Linguistics, 320–335. https://doi.org/10.18653/V1/2022.ACL-LONG.26

[13] Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. 2024. KTO: Model Alignment as Prospect Theoretic Optimization. *CoRR* abs/2402.01306 (2024). https://doi.org/10.48550/ARXIV.2402.01306 arXiv:2402.01306

[14] Sidong Feng and Chunyang Chen. 2024. Prompting Is All You Need: Automated Android Bug Replay with Large Language Models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 67:1–67:13. https://doi.org/10.1145/3597503.3608137

[15] Joseph L Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological bulletin* 76, 5 (1971), 378. https://doi.org/10.1037/h0031619

[16] Jiawei Gu, Zacc Yang, Chuanghao Ding, Rui Zhao, and Fei Tan. 2024. CMR Scaling Law: Predicting Critical Mixture Ratios for Continual Pre-training of Language Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*. Association for Computational Linguistics, 16143–16162. https://doi.org/10.18653/V1/2024.EMNLP-MAIN.903

[17] Hongcheng Guo, Jian Yang, Jiaheng Liu, Liqun Yang, Linzheng Chai, Jiaqi Bai, Junran Peng, Xiaorong Hu, Chao Chen, Dongfeng Zhang, Xu Shi, Tieqiao Zheng, Liangfan Zheng, Bo Zhang, Ke Xu, and Zhoujun Li. 2024. OWL: A Large Language Model for IT Operations. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=SZOQ9RKYJu

[18] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. 2017. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In *2017 IEEE International Conference on Web Services, ICWS 2017, Honolulu, HI, USA, June 25-30, 2017*. IEEE, 33–40. https://doi.org/10.1109/ICWS.2017.13

[19] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=d7KBjmI3GmQ

[20] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net. https://openreview.net/forum?id=nZeVKeeFYf9

[21] Yuchao Huang, Junjie Wang, Zhe Liu, Yawen Wang, Song Wang, Chunyang Chen, Yuanzhe Hu, and Qing Wang. 2024. CrashTranslator: Automatically Reproducing Mobile Application Crashes Directly from Stack Trace. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 18:1–18:13. https://doi.org/10.1145/3597503.3623298

[22] Julen Kahles, Juha Torronen, Timo Huuhtanen, and Alexander Jung. 2019. Automating Root Cause Analysis via Machine Learning in Agile Software Testing Environments. In *12th IEEE Conference on Software Testing, Validation and Verification, ICST 2019, Xi'an, China, April 22-27, 2019*. IEEE, 379–390. https://doi.org/10.1109/ICST.2019.00047

[23] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*. Association for Computational Linguistics, 6769–6781. https://doi.org/10.18653/V1/2020.EMNLP-MAIN.550

[24] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*. ACM, 39–48. https://doi.org/10.1145/3397271.3401075

[25] Barbara A. Kitchenham and Shari Lawrence Pfleeger. 2002. Principles of survey research: part 5: populations and samples. *ACM SIGSOFT Softw. Eng. Notes* 27, 5 (2002), 17–20. https://doi.org/10.1145/571681.571686

[26] J Richard Landis and Gary G Koch. 1977. The measurement of observer agreement for categorical data. *biometrics* (1977), 159–174. https://doi.org/10.2307/2529310

[27] Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. 2019. Automated program repair. *Commun. ACM* 62, 12 (2019), 56–65. https://doi.org/10.1145/3318162

[28] Chun Li, Hui Li, Zhong Li, Minxue Pan, and Xuandong Li. 2025. Enhancing Fault Localization in Industrial Software Systems via Contrastive Learning. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 691–703. https://doi.org/10.1109/ICSE55347.2025.00009

[29] Chun Li, Yifan Xiong, Zhong Li, Wenhua Yang, and Minxue Pan. 2023. Mobile Test Script Generation from Natural Language Descriptions. In *23rd IEEE International Conference on Software Quality, Reliability, and Security, QRS 2023, Chiang Mai, Thailand, October 22-26, 2023*. IEEE, 348–359. https://doi.org/10.1109/QRS60937.2023.00042

[30] Haitao Li, You Chen, Qingyao Ai, Yueyue Wu, Ruizhe Zhang, and Yiqun Liu. 2024. LexEval: A Comprehensive Chinese Legal Benchmark for Evaluating Large Language Models. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*. http://papers.nips.cc/paper_files/paper/2024/hash/2cb40fc022ca7bdc1a9a78b793661284-Abstract-Datasets_

and_Benchmarks_Track.html

[31] Xia Li, Wei Li, Yuqun Zhang, and Lingming Zhang. 2019. DeepFL: integrating multiple fault diagnosis dimensions for deep fault localization. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019*. ACM, 169–180. https://doi.org/10.1145/3293882.3330574

[32] Zhong Li, Minxue Pan, Yu Pei, Tian Zhang, Linzhang Wang, and Xuandong Li. 2022. DeepLabel: Automated Issue Classification for Issue Tracking Systems. In *Internetware 2022: 13th Asia-Pacific Symposium on Internetware, Hohhot, China, June 11 - 12, 2022*. ACM, 231–241. https://doi.org/10.1145/3545258.3545276

[33] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, 74–81.

[34] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). arXiv:1907.11692 http://arxiv.org/abs/1907.11692

[35] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. 2024. Make LLM a Testing Expert: Bringing Human-like Interaction to Mobile GUI Testing via Functionality-aware Decisions. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 100:1–100:13. https://doi.org/10.1145/3597503.3639180

[36] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Yuekai Huang, Jun Hu, and Qing Wang. 2024. Unblind Text Inputs: Predicting Hint-text of Text Input in Mobile Apps via LLM. In *Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024, Honolulu, HI, USA, May 11-16, 2024*. ACM, 51:1–51:20. https://doi.org/10.1145/3613904.3642939

[37] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Zhilin Tian, Yuekai Huang, Jun Hu, and Qing Wang. 2024. Testing the Limits: Unusual Text Inputs Generation for Mobile App Crash Detection with Large Language Model. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 137:1–137:12. https://doi.org/10.1145/3597503.3639118

[38] Yiling Lou, Qihao Zhu, Jinhao Dong, Xia Li, Zeyu Sun, Dan Hao, Lu Zhang, and Lingming Zhang. 2021. Boosting coverage-based fault localization via graph-based representation learning. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*. ACM, 664–676. https://doi.org/10.1145/3468264.3468580

[39] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian J. McAuley, Han Hu, Torsten Scholak, Sébastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, and et al. 2024. StarCoder 2 and The Stack v2: The Next Generation. *CoRR* abs/2402.19173 (2024). https://doi.org/10.48550/ARXIV.2402.19173 arXiv:2402.19173

[40] Ha-Thanh Nguyen. 2023. A Brief Report on LawGPT 1.0: A Virtual Legal Assistant Based on GPT-3. *CoRR* abs/2302.05729 (2023). https://doi.org/10.48550/ARXIV.2302.05729 arXiv:2302.05729

[41] Juri Opitz and Sebastian Burst. 2019. Macro F1 and Macro F1. *CoRR* abs/1911.03347 (2019). arXiv:1911.03347 http://arxiv.org/abs/1911.03347

[42] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. http://papers.nips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html

[43] Minxue Pan, Tongtong Xu, Yu Pei, Zhong Li, Tian Zhang, and Xuandong Li. 2022. GUI-Guided Test Script Repair for Mobile Apps. *IEEE Trans. Software Eng.* 48, 3 (2022), 910–929. https://doi.org/10.1109/TSE.2020.3007664

[44] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. ACL, 311–318. https://doi.org/10.3115/1073083.1073135

[45] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. http://papers.nips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html

[46] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Association for Computational Linguistics, 3980–3990. https://doi.org/10.18653/V1/D19-1410

[47] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL] https://arxiv.org/abs/2308.12950

[48] Weisong Sun, Yun Miao, Yuekang Li, Hongyu Zhang, Chunrong Fang, Yi Liu, Gelei Deng, Yang Liu, and Zhenyu Chen. 2025. Source Code Summarization in the Era of Large Language Models. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 1882–1894. https://doi.org/10.1109/ICSE55347.2025.00034

[49] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. 3104–3112. https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html

[50] Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. 2023. Large language models in medicine. *Nature medicine* 29, 8 (2023), 1930–1940. https://doi.org/10.1038/s41591-023-02448-8

[51] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *CoRR* abs/2302.13971 (2023). https://doi.org/10.48550/ARXIV.2302.13971 arXiv:2302.13971

[52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[53] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. 2024. MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*. http://papers.nips.cc/paper_files/paper/2024/hash/ad236edc564f3e3156e1b2feafb99a24-Abstract-Datasets_and_Benchmarks_Track.html

[54] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David S. Rosenberg, and Gideon Mann. 2023. BloombergGPT: A Large Language Model for Finance. *CoRR* abs/2303.17564 (2023). https://doi.org/10.48550/ARXIV.2303.17564 arXiv:2303.17564

[55] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated Program Repair in the Era of Large Pre-trained Language Models. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 1482–1494. https://doi.org/10.1109/ICSE48619.2023.00129

[56] Chunqiu Steven Xia and Lingming Zhang. 2022. Less training, more repairing please: revisiting automated program repair via zero-shot learning. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*. ACM, 959–971. https://doi.org/10.1145/3540250.3549101

[57] Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2024. InfLLM: Training-Free Long-Context Extrapolation for LLMs with an Efficient Context Memory. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*. http://papers.nips.cc/paper_files/paper/2024/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html

[58] Dongling Xiao, Han Zhang, Yu-Kun Li, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. 2020. ERNIE-GEN: An Enhanced Multi-Flow Pre-training and Fine-tuning Framework for Natural Language Generation. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*. ijcai.org, 3997–4003. https://doi.org/10.24963/IJCAI.2020/553

[59] Junjielong Xu, Ziang Cui, Yuan Zhao, Xu Zhang, Shilin He, Pinjia He, Liqun Li, Yu Kang, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. 2024. UniLog: Automatic Logging via LLM and In-Context Learning. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 14:1–14:12. https://doi.org/10.1145/3597503.3623326

[60] Tongtong Xu, Minxue Pan, Yu Pei, Guiyin Li, Xia Zeng, Tian Zhang, Yuetang Deng, and Xuandong Li. 2021. GUIDER: GUI structure and vision co-guided test script repair for Android apps. In *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021*. ACM, 191–203. https://doi.org/10.1145/3460319.3464830

[61] Aidan Z. H. Yang, Claire Le Goues, Ruben Martins, and Vincent J. Hellendoorn. 2024. Large Language Models for Test-Free Fault Localization. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 17:1–17:12. https://doi.org/10.1145/3597503.3623342

[62] Hongyang Yang, Xiao-Yang Liu, and Christina Dan Wang. 2023. FinGPT: Open-Source Financial Large Language Models. *CoRR* abs/2306.06031 (2023). https://doi.org/10.48550/ARXIV.2306.06031 arXiv:2306.06031

[63] Yuan Yao, Tianyu Yu, Ao Zhang, Chongyi Wang, Junbo Cui, Hongji Zhu, Tianchi Cai, Haoyu Li, Weilin Zhao, Zhihui He, Qianyu Chen, Huarong Zhou, Zhensheng Zou, Haoye Zhang, Shengding Hu, Zhi Zheng, Jie Zhou, Jie Cai, Xu Han, Guoyang Zeng, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. MiniCPM-V: A GPT-4V Level MLLM on Your Phone. *CoRR* abs/2408.01800 (2024). https://doi.org/10.48550/ARXIV.2408.01800 arXiv:2408.01800

[64] Shengcheng Yu, Chunrong Fang, Yuchen Ling, Chentian Wu, and Zhenyu Chen. 2023. LLM for Test Script Generation and Migration: Challenges, Capabilities, and Opportunities. In *23rd IEEE International Conference on Software Quality, Reliability, and Security, QRS 2023, Chiang Mai, Thailand, October 22-26, 2023*. IEEE, 206–217. https://doi.org/10.1109/QRS60937.2023.00029

[65] Peiwen Yuan, Shaoxiong Feng, Yiwei Li, Xinglin Wang, Yueqi Zhang, Chuyi Tan, Boyuan Pan, Heda Wang, Yao Hu, and Kan Li. 2024. Focused Large Language Models are Stable Many-Shot Learners. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*. Association for Computational Linguistics, 6247–6261. https://doi.org/10.18653/V1/2024.EMNLP-MAIN.359

[66] Jian Zhang, Chong Wang, Anran Li, Wenhan Wang, Tianlin Li, and Yang Liu. 2024. VulAdvisor: Natural Language Suggestion Generation for Software Vulnerability Repair. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE 2024, Sacramento, CA, USA, October 27 - November 1, 2024*. ACM, 1932–1944. https://doi.org/10.1145/3691620.3695555

[67] Yakun Zhang, Wenjie Zhang, Dezhi Ran, Qihao Zhu, Chengfeng Dou, Dan Hao, Tao Xie, and Lu Zhang. 2024. Learning-based Widget Matching for Migrating GUI Test Cases. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 69:1–69:13. https://doi.org/10.1145/3597503.3623322

[68] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. 2024. LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Association for Computational Linguistics, Bangkok, Thailand, 400–410. https://doi.org/10.18653/v1/2024.acl-demos.38

[69] Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. LIMA: Less Is More for Alignment. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. http://papers.nips.cc/paper_files/paper/2023/hash/ac662d74829e4407ce1d126477f4a03a-Abstract-Conference.html

[70] Hongjian Zhou, Fenglin Liu, Boyang Gu, Xinyu Zou, Jinfa Huang, Jinge Wu, Yiru Li, Sam S. Chen, Peilin Zhou, Junling Liu, Yining Hua, Chengfeng Mao, Chenyu You, Xian Wu, Yefeng Zheng, Lei Clifton, Zheng Li, Jiebo Luo, and David A. Clifton. 2024. A Survey of Large Language Models in Medicine: Progress, Application, and Challenge. arXiv:2311.05112 [cs.CL] https://arxiv.org/abs/2311.05112

Received 2026-01-30; accepted 2026-06-25