Master's thesis

Master's Programme in Computer Science

The Use of Large Language Models in Mobile Application Testing

Sara Salmi

December 19, 2024

Faculty of Science

University of Helsinki

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Master's Programme in Computer Science

Tekijä — Författare — Author

Sara Salmi

Työn nimi — Arbetets titel — Title

The Use of Large Language Models in Mobile Application Testing

Ohjaajat — Handledare — Supervisors

Prof. Giulio Jacucci

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	December 19, 2024	42 pages

Tiivistelmä — Referat — Abstract

Large Language Models (LLMs) have got major advancements in the last few years and they provide new possibilities in software testing due to their contextual understanding and human-like reasoning capabilities. As mobile applications grow both in popularity and complexity, ensuring their quality through testing is crucial. Since mobile application testing still relies heavily on manual testing and automated testing tools are lacking in certain areas, there is potential to enhance the current mobile application testing processes with the help of LLMs.

To research the use of LLMs in mobile application testing, this thesis reviewed 20 research papers from years 2023-2024. The results show that LLMs have been used in seven different testing activities, such as defect management, graphical user interface (GUI) testing and text input generation. In these testing tasks, the LLMs took different roles, such as simulating human professionals, generating test scripts or deciding the next actions to take when navigating the mobile app views. Most of the LLM applications were successful, but the need for further research was also identified, especially in vulnerability assessment.

The main challenges and limitations for using LLMs in mobile application testing lie in the unpredictable qualities of LLMs, such as randomness in LLM outputs and the variability in the performance of different models. There are also other limitations, such as the difficulty for LLMs to handle certain testing specific tasks, like scrolling. While it is important to address these limitations, LLMs have already shown great potential in enhancing the mobile application testing processes, and LLMs might be able to tackle some of the issues in current automation tools in the future. However, the research is still in its early stages, and further studies are required to fully understand the practical applicability of LLMs in mobile application testing.

ACM Computing Classification System (CCS)
Software and its engineering → Software creation and management → Software verification and validation
Computing methodologies → Artificial Intelligence → Natural language processing

Avainsanat — Nyckelord — Keywords

Software testing, Mobile application testing, Large Language Model, LLM

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Software study track

Contents

1 Introduction

Mobile applications are used by billions of people globally [17, 16] for a wide range of activities, such as social networking, banking, and leisure activities like fitness and gaming. In addition to the prevalence of mobile apps in our daily lives, the complexity of these apps is also increasing. The apps often need to provide multiple different features, take advantage of the mobile device's sensors (such as biometric and GPS sensors) and feature different kinds of third-party integrations. This complexity is further increased by the diversity of devices for which the correct functionality needs to be ensured: mobile devices have different manufacturers, hardware components, operating system versions, and device specifications which can have an effect on how the app is working for the user [4].

In addition to the previously mentioned issues, mobile applications still rely heavily on manual testing [11], mainly due to the limitations in existing automation tools and the complexity of the mobile ecosystem [1, 11]. For example, security testing and regression testing for mobile applications are not well supported [1]. The applications need to also keep up with the constant operating system updates [21]. All of this often makes the testing of mobile applications complicated, time-consuming and expensive.

Given these factors, robust and efficient software testing is more critical than ever to maintain mobile app quality. Bugs in the apps can cause more than just minor user frustrations - they can damage a company's reputation, compromise user privacy, and lead to significant financial losses, especially if the bugs lead to violating regulations like the General Data Protection Regulation (GDPR). Thus, effective mobile application testing is essential, and new methods to enhance the testing process are continually being explored.

Recent advancements in Large Language Models (LLMs) have opened up new possibilities in the field of software testing. A large survey on software testing with LLMs revealed that LLMs are already being used in various stages of software testing, such as preparing test cases, generating test reports, reporting bugs, fixing bugs and performing regression testing [18]. Mobile application testing also stood out as a popular category in that same survey, which suggests there is a growing interest to enhance the mobile application testing processes with LLMs.

Therefore, we want to take a deeper look into the possibilities of utilizing LLMs in mobile application testing. We also want to find out the possible challenges and limitations of it.

Our research will address the following research questions (RQs):

RQ1: How are LLMs used in mobile application testing?

RQ2: What kind of limitations or challenges are there for using LLMs in mobile application testing?

This thesis presents the findings of a Systematic Literature Review (SLR) examining the use of LLMs in mobile application testing. The review includes 20 research papers published between 2023-2024, 11 of which are peer-reviewed. To our understanding, this thesis is the first of its kind to gather a comprehensive review of the possibilities and challenges of using LLMs in mobile application testing and is therefore filling a gap in existing research.

The paper is structured as follows: chapter 2 explains the basic concepts that are used in this paper, chapter 3 presents the research method in detail, chapter 4 presents the literature review findings, chapter 5 discusses the impact of the findings and the limitations of the study and, finally, chapter 6 concludes the whole research and highlights the key takeaways.

For disclosure, it is important to mention that large language model ChatGPT [14] was used during this thesis work to provide ideas and guidance. For example, ChatGPT was asked to tell what kind of information to collect from the reviewed papers in order to properly answer the question of *"How are LLMs used in mobile application testing?"*. ChatGPT was also asked to check the correctness of some claims, such as the relationship of natural language processing and large language models, and to explain some sentences from the reviewed papers that were somehow unclear. While everything written in this thesis is our own work, ChatGPT was a valuable help during this thesis process, which also highlights the real-life practicality of LLMs and the relevance of our research topic.

2 Background

2.1 Generative AI

Generative AI (Artificial Intelligence) has transformed the world of AI, enhancing the AI capabilities from merely analyzing existing data to being able to also produce new data [3]. Generative AI refers to machine learning models that are designed to create new data, such as images or text, by learning patterns and observing relationships from the model's training data [5]. The data that is generated by generative AI models is similar (but not identical) to the training data, which allows the generative models to create seemingly new, yet plausible data [5, 19] that often cannot be even distinguished anymore from human craftsmanship [5]. Thus, even with the same input prompt, the generative AI models are capable of producing different outputs [19], for example different kinds of artworks with the same topic prompt.

Generative AI has many different practical applications [6] and the possibilities are almost limitless. Not only has generative AI been used to generate new artistic data, it also supports businesses and aids humans with scientific research. Generative AI can, for example, be used to create art, write text that resembles the work of writers, edit images or create videos [6, 5]. Generative AI models can also be used for different conversational tasks, where the user can, for example, ask the model to create itineraries, summarize text or provide medical advice (although it does not not replace human professionals) [6].

Generative AI has also practical applications in many other areas, such as video game generation (e.g., generating game assets and game characters), music creation and different biotech tasks, such as modelling proteins and creating new drugs [6]. On top of these, generative AI has also business and marketing applications, as the models can write product descriptions, create advertisements, automate customer conversations or write website contents - just to name a few [6]. Among these generative AI applications are also large language models (LLMs), which will be discussed in Section 2.3.

2.2 Natural Language Processing (NLP)

A lot of research efforts have been directed to Natural Language Processing (NLP) in the recent years. NLP allows people to communicate with computers in a seamless way, since NLP enables computers to perform different natural language related tasks, like understand and generate text, answer questions and translate text from one language to another [2]. The advancements in NLP have been crucial at enabling the seamless use of many systems that are nowadays used on a daily basis, such as search engines, chatbots and generative AI tools [8].

For example, instead of relying solely on matching specific keywords, NLP has allowed search engines to understand the user's intent behind the search words in order to provide overall more accurate and relevant results [8]. This kind of understanding of intent is possible due to advancements in NLP, which paves the way also for more accessible use of generative AI models. For example, asking a generative AI model to create a song with a specific topic is possible due to the generative AI models' language understanding capabilities. Advancements in NLP and Generative AI have also been instrumental in the development of large language models (LLMs), although it is possible to also argue in the opposite direction that the development of LLMs has expanded the NLP capabilities.

2.3 Large Language Models (LLMs)

In this thesis, we focus on Large Language Models (LLMs), which are a form of generative AI. Large language models are machine learning models that are designed to perform different NLP tasks [15], such as understand, generate and summarize text in a human-like manner. LLMs are also able to generate code, perform creative writing tasks and translate languages.

LLMs utilize neural network architectures, such as Transformer architecture, to capture complex language patterns and dependencies [20]. LLMs are trained on vast amounts of text data from a variety of different sources, such as books, websites and articles [20] and the models typically have billions of machine learning parameters that enable them to understand language and predict which word to use next based on the previous words in a given context [20, 7]. For example, GPT-3 from OpenAI has 175 billion parameters [2].

LLMs can be uni-modal, meaning that the model's input and output are the same data

type [5]. Typically this means that the model takes only text as input and also gives the output as text. However, recently LLM capabilities have been even further extended by multi-modality. Multi-modality means that the LLMs are able to process more than just one type of data, and the model's input can be different data type than the model's output [5]. Thus, these multi-modal LLMs are also able to process for example images, making them even more powerful at handling different tasks. An example of a multi-modal model is GPT-4 from OpenAI, which is able to take both text and images as input and which gives the outputs as plain text [13]. We will consider both uni-modal and multi-modal models as LLMs in the context of this thesis.

To further improve the LLM capabilities in different tasks, LLMs can also be fine-tuned. Fine-tuning means that a pre-trained large language model is re-trained on new, more specialized data, which can make general-purpose models more suited for specific tasks, such as answering questions or classifying documents [5]. Often fine-tuning is used if the LLM needs domain-specific knowledge to efficiently perform the tasks it is set to do.

2.4 Mobile Application Testing

Mobile application (app) testing consists of the testing processes that are meant for ensuring quality specifically in mobile applications. It has some characteristics that make it different from other type of testing, such as desktop application testing. Mobile application testing needs to cover not only the expected behavior, but the correct behavior also needs to be verified on multiple devices and operating system versions [4, 11].

There might be differences on both hardware and software level between different devices, and the differences in the device properties also add some complexity to the testing [4, 11]. For example, the screen size and resolution can vary from one device to another, which can affect how different elements of the app are shown on the screen [4]. Thus, compatibility testing is often of high importance when testing mobile applications [1]. Different sensors, such as biometric sensor, need to also be taken into account when testing mobile applications [4].

Mobile applications are also tested in multiple other ways [11]. For example, functional testing is performed for the apps to ensure the app's features work according to the requirements, and it is often of high importance when testing mobile applications [1]. It is also important to ensure that the mobile apps run well and are stable to use, which is why performance testing is often done for the apps [1]. Usability testing is also seen as a

key method for ensuring the pleasant use of mobile applications [11].

Mobile devices have still resource limitations compared to computers and laptops [21], for example due to mobile devices' portability and small size. In addition to physical limitations, the mobile ecosystem can be characterized by its fast-paced nature. The mobile operating systems are often getting upgrades in short time periods and on a regular basis [21, 1], and the mobile apps themselves need to react to these fast changes. The apps need to also compete in a fast-evolving market space, which requires both fast development speed and efficient testing. This also makes regression testing very important, as testers need to make sure the new changes do not break existing functionalities [1].

Although the mobile application space is challenging and efficient testing solutions are needed, current automated mobile application testing tools are lacking in certain areas [1]. For example, security testing and regression testing are not well supported for mobile applications [1]. Many apps are also lacking effective testing practices especially in terms of code coverage and API testing [9] . Mobile application testing also still relies heavily on manual testing [11], which is likely due to the lack of suitable automation tools and due to having to manage the variety of test devices and operating systems. All this adds complexity to the testing and often makes it time-consuming.

LLMs might provide help to tackle some of the current limitations or obstacles in mobile application testing. Due to the major advances of LLMs in the last few years [2, 18], the use of LLMs has gained increased curiosity in the field of software testing and also in mobile application testing [18]. Thus, this thesis will take a deeper look not only at these capabilities, but also at the limitations of LLMs in mobile application testing.

3 Research Method

We conducted the study as a Mini Systematic Literature Review (SLR). The process follows closely the traditional SLR process, but the term mini is used to highlight that due to the scope of the thesis, a smaller, more concise version of SLR was used since an extensive SLR would have been too laborious. The research process is described in detail in this section.

3.1 Research Questions

As mentioned in the Introduction, in this thesis we want to explore the possibilities and challenges of using LLMs in mobile application testing. The following research questions were formulated:

RQ1: How are LLMs used in mobile application testing?

RQ2: What kind of limitations or challenges are there for using LLMs in mobile application testing?

3.2 The Research Portal

We performed the paper search in Google Scholar. Other portals, namely Scopus, ACM Digital Library and IEEE Xplore were also considered, but the results from those were very limited and every relevant result that we got from those three portals were included in the results we got from Google Scholar. Therefore using more portals would have only made the process more complicated while not providing any real benefit.

Moreover, given that LLMs are still a fairly recent innovation and new research is continuously conducted, we wanted to gain access to as many research papers as possible which also supports the use of Google Scholar, as it finds also those papers that have not yet been published in peer-reviewed portals.

3.3 Search Query

After trying multiple search queries, we finally proceeded with the following search query: *(llm* OR "large language model" OR "GPT") AND ("mobile app" OR "mobile GUI" OR "Android app") AND (test* OR "bug" OR "issue" OR "defect" OR "fault" OR "error" OR "failure" OR "crash" OR debug* OR "repair" OR "fix" OR assert* OR fuzz*)*

We can break down the search query into 3 parts. The main idea is that the query has at least one LLM related keyword, one mobile application related keyword and one testing related keyword to ensure the results match our research questions as well as possible. Wildcards (marked with asterisk *) were used in the search phrase in order to get matching results for different variations of the same word (for example in the case of fuzz*, the search would find results where *fuzzing*, *fuzz testing*, *fuzzer* or some other variation starting with *fuzz* is used).

For LLMs, the query (llm* OR "large language model" OR "GPT") includes different versions of the roof term "LLM" as well as the term "GPT" ("Generative pre-trained transformer") because some papers mentioned only a specific LLM and GPT was a commonly used term in the names for different LLMs. Adding other name variations for different LLMs gave a lot of unsuitable results and only made the query longer, thus challenging the 256 character limit for Google Scholar search queries. As a result of that, we decided to not name other LLMs.

To ensure the search provides results in the correct application area, i.e. mobile applications, we included different mobile related keywords to focus the search on this application area. As we can see from the query ("mobile app" OR "mobile GUI" OR "Android app"), it includes three mobile related keywords. It includes the keyword "Android app" because most of the mobile related research is done for the Android ecosystem. Mobile GUI was selected because it's a popular research area for LLMs. The keyword "mobile app" is a self-evident choice because it matches well with the overall focus area of our research. A more general term "mobile" was also considered but it gave a lot of results outside the area of computer science which is why it was excluded.

The papers had to also have at least one software testing related keyword. We wanted to include as many variations for software testing as possible, since simply adding "software testing" or "test" did not give enough results. The testing related keywords we used were largely inspired by the keywords J. Wang et al. [18] described in their paper, since they had gathered a comprehensive list of different software testing related keywords. Thus,

our keywords were (test* OR "bug" OR "issue" OR "defect" OR "fault" OR "error" OR "failure" OR "crash" OR debug* OR "repair" OR "fix" OR assert* OR fuzz*).

3.4 Inclusion and Exclusion Criteria

The inclusion and exclusion criteria were defined to include those papers that provide answers to the research questions and exclude the papers that do not match the research questions or are otherwise unsuitable to use as references.

We wanted to exclude papers that were published before the year 2022, as the development of LLMs has been extremely fast in the past few years and utilizing LLMs in software testing is a relatively new area of interest. In addition to that, based on the papers we found, majority of the research papers were published in 2024, so including papers from over couple of years back would have mainly just slowed down the paper selection process.

The paper needed to be a research paper. However, we did not limit the search to only peer-reviewed papers as there would not have been enough material to perform the literature review without the non-peer-reviewed ones. We acknowledge the risk that comes with it, but we have evaluated the papers carefully within the paper selection process and we have taken measures that help to ensure the validity of this research. For more information about the measures we have taken, see the Section 5.6.

In addition to limiting the publication date, the paper had to be written in English so that it is understandable by us. It also needed to be related to testing mobile applications with the help of LLMs; either by presenting testing practices or by addressing the limitations of why LLMs cannot be used in mobile application testing context. Papers that were not research papers or that were theses, were not accessible for free or with University of Helsinki credentials or did not mention LLMs in the context of mobile application testing were excluded.

The summary of the inclusion and exclusion criteria can be seen in Table 3.1.

3.5 The Paper Selection Process

The search was done during August 2024 so results that have been published after that are not included. However, for clarity purposes, it is worth mentioning that two search results, specifically [S16] and [22], were found through Google Scholar search from arXiv

	Inclusion criteria
1	The paper is a research paper
2	The paper is related to mobile application testing with LLMs
3	The paper either presents ways to utilize LLMs for mobile application testing and/or discusses the limitations or challenges of it
	Exclusion criteria
1	The paper is not a research paper
2	The paper is a thesis
3	The paper is published before the year 2022
4	The paper is not written in English
5	The paper is not accessible for free or with University of Helsinki credentials
6	The paper is not related to mobile application testing
7	The paper does not present or mention ways to test mobile applications with LLMs or describe the limitations or challenges of it

Table 3.1: Inclusion and exclusion criteria

in August but as we were going through the papers, we noticed they were published on ACM during September. We used the ACM publication for those papers as the reference, which is why the reference section shows two papers published after August.

Google Scholar showed only an estimated number of results for the search query, therefore the real number of results is not available. The estimated number of results with our search query was 1440. We limited the results to the years 2022-2024, which resulted in 1120 papers overall.

Due to the scope of the thesis, we selected only the first 200 results from the 1120 papers for detailed evaluation. There were 2 reasons for this: 1) Google scholar sorts the papers by relevance and the number of papers that matched the inclusion criteria started to drop significantly after the first 100 results, which suggests that it is unlikely that relevant papers were excluded due to not being in the top 200 results. 2) Going through more than 200 results would have made the research process a lot more time-consuming, thus making it too laborious for the scope of this thesis.

We systematically analyzed the 200 top results by examining the paper title and abstract and, if necessary, we quickly scanned the full paper to decide whether the paper matched the inclusion criteria. Duplicate papers were excluded, and otherwise unsuitable papers

were excluded based on the exclusion criteria. After applying the inclusion and exclusion criteria, the number of papers that made it to the final review is 20. These papers are referenced in a separate bibliography.

The whole paper selection process is described in the graph below.

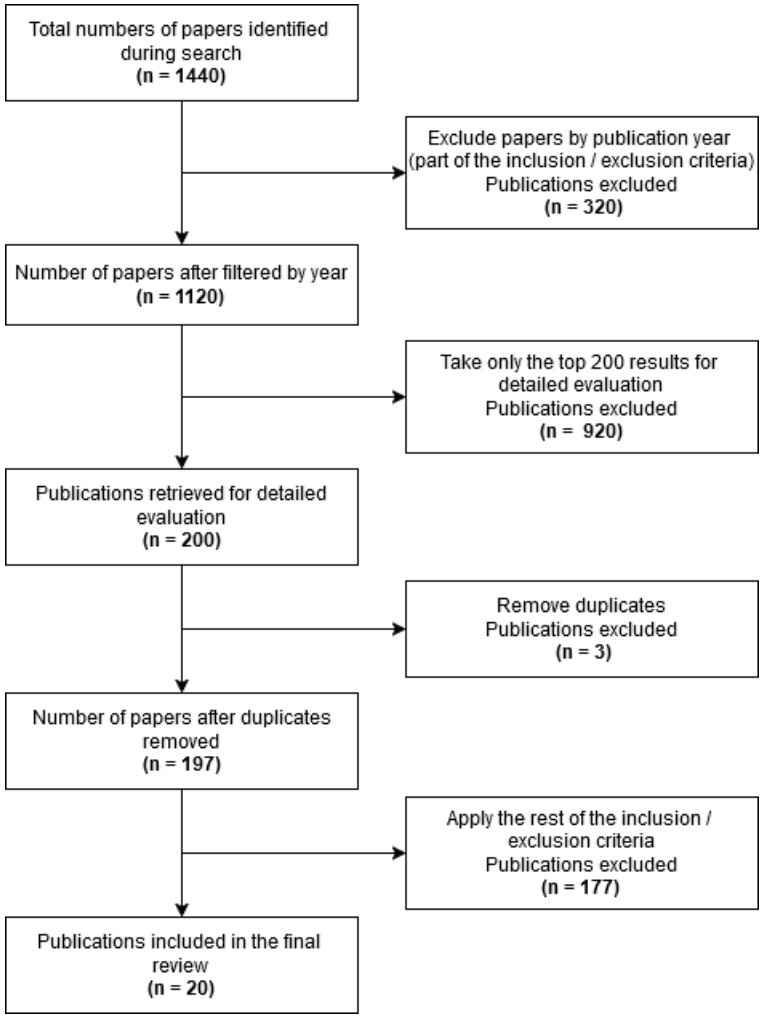

Figure 3.1: The detailed paper selection process

4 Results

4.1 Overall Result Analysis

As stated in the Research Method section, 20 papers made it to the final review. All 20 papers mentioned ways to utilize LLMs in mobile application testing, while 19 of them were also addressing the limitations or challenges of using LLMs in mobile application testing. Table 4.1 shows basic information about the papers included in the final review. It shows the reference number, the publication year, the general topic and the original paper title for each included paper. It also highlights which papers have been peer-reviewed and which have not.

From the 20 papers, 18 of them (90%) were from the year 2024, while only two of them (10%) were from the year 2023. No studies from the year 2022 made it to the final review, which highlights the novelty of using LLMs in mobile application testing and software testing in general. This also aligns well with the fast development of LLMs in the last couple of years. The bar plot in Figure 4.1 shows the number of included papers based on publication year. The number of papers in this field is expected to grow even faster in the upcoming years, which is well highlighted also by the growing trend of the included papers.

Overall 11 of the reviewed papers (55%) were published in a peer-reviewed journal or in conference proceedings. The rest of the papers (45%) were not peer-reviewed and were published either in arXiv or SSRN. The high number of non-peer-reviewed papers might be explained by the fact that all of the non-peer-reviewed papers were published this year, so it is possible some of them have not yet gone through the (often time-consuming) peer-review process. Using LLMs in mobile application testing is also a new research area, so it is plausible that the number of peer-reviewed papers will increase in the near future.

The high number of non-peer-reviewed papers is a risk for the validity of this research, but we have mitigated the risk by different measures: we are transparent about which papers are peer-reviewed and which are not, and we also analyze the results for the peer-reviewed and non-peer-reviewed papers both together and separately, so that it is possible to tell if the non-peer-reviewed papers alter the overall results.

Ref	Year	Topic	Paper title	Peer-reviewed
[S11]	2024	GUI testing	Make LLM a Testing Expert: Bringing Human-like Interaction to Mobile GUI Testing via Functionality-aware Decisions	Yes
[S18]	2023	GUI testing	Intent-Driven Mobile GUI Testing with Autonomous Large Language Model Agents	Yes
[S3]	2024	Defect management	Prompting Is All You Need: Automated Android Bug Replay with Large Language Models	Yes
[S16]	2024	Defect management	Feedback-Driven Automated Whole Bug Report Reproduction for Android Apps	Yes
[S4]	2024	Defect management	AutoConsis: Automatic GUI-driven Data Inconsistency Detection of Mobile Apps	Yes
[S6]	2024	Defect management	CrashTranslator: Automatically Reproducing Mobile Application Crashes Directly from Stack Trace	Yes
[S2]	2024	Text input generation	Testing the Limits: Unusual Text Inputs Generation for Mobile App Crash Detection with Large Language Model	Yes
[S10]	2023	Text input generation	Fill in the Blank: Context-aware Automated Text Input Generation for Mobile GUI Testing	Yes
[S19]	2024	Test generation and maintenance	LLM for Test Script Generation and Migration: Challenges, Capabilities, and Opportunities	Yes
[S20]	2024	Test generation and maintenance	Learning-based Widget Matching for Migrating GUI Test Cases	Yes
[S15]	2024	Test execution and replay	AXNav: Replaying Accessibility Tests from Natural Language	Yes
[S12]	2024	GUI testing	Vision-driven Automated Mobile GUI Testing via Multimodal Large Language Model	No
[S5]	2024	GUI testing	AUITestAgent: Automatic Requirements Oriented GUI Function Testing	No
[S13]	2024	Defect management	LLM-CompDroid: Repairing Configuration Compatibility Bugs in Android Apps with Pre-trained Large Language Models	No
[S7]	2024	Defect management	A Study of Using Multimodal LLMs for Non-Crash Functional Bug Detection in Android Apps	No
[S1]	2024	Text input generation	Large Language Models for Mobile GUI Text Input Generation: An Empirical Study	No
[S8]	2024	Vulnerability assessment	Assessing the Effectiveness of LLMs in Android Application Vulnerability Analysis	No
[S14]	2024	Vulnerability assessment	LLbezpeky: Leveraging Large Language Models for Vulnerability Detection	No
[S17]	2024	Test generation and maintenance	XUAT-Copilot: Multi-Agent Collaborative System for Automated User Acceptance Testing with Large Language Model	No
[S9]	2024	Test reporting	Redefining Crowdsourced Test Report Prioritization: An Innovative Approach with Large Language Model	No

Table 4.1: Basic information about the reviewed papers

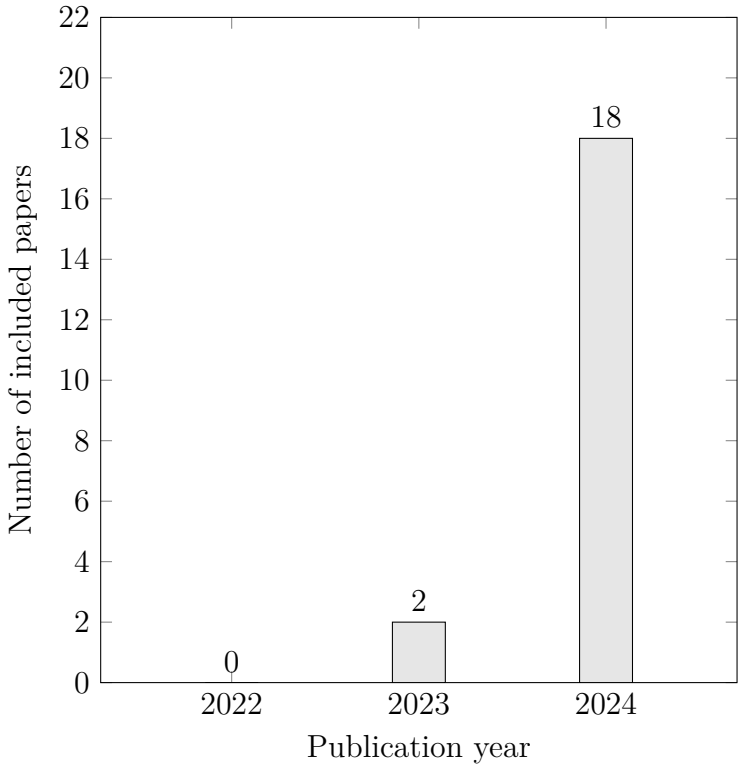

Figure 4.1: Number of included papers by publication year

4.2 Use of LLMs in Mobile Application Testing

We will approach the use of LLMs in mobile application testing from three different perspectives. Firstly, we want to find out the general areas of testing where LLMs have been used. Secondly, we want to find out which LLMs have been used to see which models are popular and perhaps more suitable for mobile application testing than others. Thirdly, we want to find out the practical role of LLMs in these scenarios.

4.2.1 Mobile Application Testing Topics

To get an understanding of how LLMs are used in mobile application testing, we will first take a look into the testing topics of each paper. The testing topic describes the area of mobile application testing where LLMs are used, and the topics are divided into 7 different categories. In this section, we will describe these different categories and highlight their prevalence in order to provide a good overview of how LLMs are used in mobile application testing.

The testing topics are first introduced in Table 4.1 which shows the basic information about the reviewed papers. The prevalence for each topic is highlighted in Table 4.6, which also shows the proportions of each category, both separately for peer-reviewed and non-peer-reviewed papers, as well as altogether. The topics and their proportions are also visually described in Figure 4.2 (not separated between peer-reviewed and non-peer-reviewed papers).

LLMs were most commonly used in **defect management**, covering almost one third of all included papers (30%). Defect management encompasses the papers that are related to detecting, reproducing and repairing defects, namely bugs and crashes. The papers in this category cover multiple different kinds of activities: automatic bug reproduction [S3, S16], crash reproduction [S6], detection of data inconsistency bugs [S4], detection and fixing of compatibility bugs [S13] as well as detection of non-crash functional bugs [S7]. The purpose of these is to streamline and automate the defect management process, especially in areas where current automation tools are lacking.

Second most popular testing topic for using LLMs was **GUI testing**: four papers (20%) focused purely on GUI testing [S11, S18, S12, S5]. GUI testing covers the papers that are about finding bugs from the mobile application via the graphical user interface (GUI). Mobile applications rely heavily on the GUI to perform different tasks and the testing of GUI often requires navigating through different views by clicking different buttons and giving suitable inputs. The whole GUI flow is not only very relevant for the app users, but the successful navigation between mobile GUI views is also crucial for good testing coverage.

It is also worth noting that some papers mentioned GUI testing but the main focus was on some other testing activity, namely generating text input for GUI testing purposes [S10] or migrating GUI test cases from one app to another [S20]. Therefore, those two papers are categorized under different topics as they are not mainly focused on GUI testing but rather on the enablers for it.

The place for the third most popular topic was shared among two categories. One of them is **text input generation**, since three papers overall (15%) introduced ways in which LLMs can assist to generate text inputs for different mobile application testing purposes. Two of those papers focused on creating text input for mobile GUI testing [S1, S2], while one of them focused on creating unusual text inputs for detecting application crashes (fuzz testing) [S2]. Proper text inputs are crucial since many mobile app views often require appropriate text input to proceed to the next UI view and the inputs can also cause

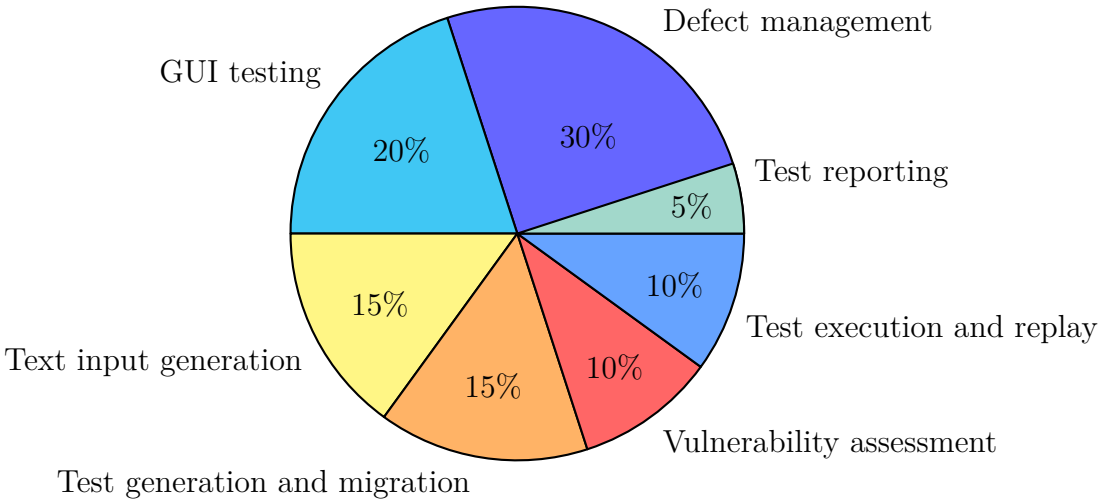

Figure 4.2: Testing topics by prevalence

crashes. For example, if you are booking a flight, you usually need to give an origin, a destination, traveling dates, etc. in order to get to the proceeding views. Without proper text inputs many relevant views might be missed, thus lowering the testing coverage. It is also very important to handle unwanted inputs properly in order to improve the security of mobile applications.

The other topic in the shared third place is **test generation and maintenance**, which also covered three papers overall (15%). Test generation and maintenance topic covers the papers that are for example about generating test scripts or cases, maintaining test cases or migrating tests from one platform or application to another. One of the papers under this category was about test generation and migration [S19], one about GUI test case migration from source app to target app by widget matching [S20] and one about test script generation for automated user acceptance testing (UAT) [S17]. The purpose of this category was to make the test generation and maintenance process more efficient and less reliant on manual work from humans.

Vulnerability assessment was a smaller category, covering altogether 2 papers (10%). The vulnerability assessment topic covers the papers that focus on analysis, detection and fixing of vulnerability issues. This includes for example analyzing code snippets for vulnerabilities or fixing vulnerability causing issues from code snippets with the help of LLMs in order to improve the security of mobile applications. More specifically, the papers under this topic were about vulnerability analysis for Android applications [S8] and vulnerability detection and fixing for Android applications [S14].

There were also two smaller categories: **test execution and replay** and **test reporting**. Test execution and replay covers the papers that are related to running or replaying tests. Only one paper was categorized under this topic (5%), and it was about automatically running accessibility tests for iOS applications [S15] in order to reduce the need for manual work in UAT and to make the process faster. Test reporting, on the other hand, encompasses the papers that are about analyzing, gathering and presenting test results. Only one paper (5%) was categorized under this topic, and it was about crowdsourced test report prioritization [S9], which purpose is to address the inefficiencies in managing the vast number of reports generated during crowdsourced mobile application testing.

4.2.2 Breakdown of the Used LLMs

We will briefly describe which LLMs were used in the reviewed papers. Table 4.2 shows which LLM or LLMs were used in each reviewed paper. It also highlights which of the papers were peer-reviewed and which were not. The used LLMs and their proportions are also showed in Figure 4.3, but it is not separated between peer-reviewed and non-peer-reviewed papers.

As we can see from the Figure 4.3, the most used LLMs were different LLMs from OpenAI: 25% of the papers were using ChatGPT, 20% GPT-4 and 10% GPT-3. Different versions of the GPT-4 model were also used, namely GPT-4o and GPT-4 Turbo. Four papers (20%) were comparing how different LLMs performed for some specific tasks. Other models, such as BERT and GPT-3.5, had a smaller 5% share. For one paper (5%), the chosen LLM was not clearly stated [S17], thus it is described as "not specified".

4.2.3 The Role of LLMs in Mobile Application Testing

To gain a better understanding of how LLMs are used in mobile application testing, we decided to take a more detailed look at the role of LLM in each research paper. The findings from peer-reviewed papers are collected to Table 4.3, while the findings from the non-peer-reviewed papers are collected to Table 4.4. The tables highlight the general use case, as well as the role of the LLMs. They also show the outcome of the research to see how well the LLMs performed in the given tasks. In this section, we will highlight the main findings based on the two tables.

As we can see from Tables 4.3 and 4.4, the role of LLM varied depending on the use case.

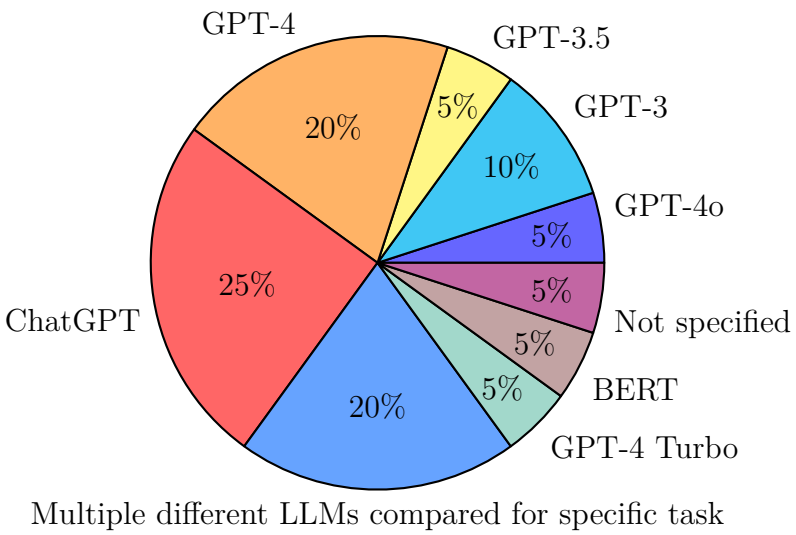

Figure 4.3: Used LLMs (for more details, see Table 4.2)

A few solutions relied on the use of multiple LLM-based agents to perform a specific task, and it was especially popular in GUI testing [S5, S18, S17]. In some papers, the LLM also took the role of a human professional. For example, the LLM worked as a human-like GUI tester describing which action to take next [S11] or simulated a professional developer who interprets bug reports and tries to reproduce them [S3].

Due to the generative nature of LLMs, many of the papers used the LLM for generating different testing items, such as creating text inputs [S2, S10, S1], generating test scripts [S19, S17], creating code fixes [S13] or generating event sequences that reproduce reported bugs [S16]. In some cases, the LLM rather analyzed whether a bug was found and gave a justification for the decision [S4, S7].

Overall, the result of using LLM was a success in most cases [S11, S15, S1, S14, S17], and many of the successful LLM applications also outperformed the state-of-the-art tools [S18, S3, S16, S4, S6, S2, S10, S20, S12, S5, S7, S9]. This means that especially in GUI testing, defect management, text input generation, test generation and test reporting the use of LLM was successful and the LLM-based solution performed better than existing tools.

However, some solutions also acknowledged the limitations of using LLMs in mobile application testing and highlighted the need for further research [S19, S13, S8]. This means that at least vulnerability assessment, test script migration and detection and fixing of compatibility bugs are still in early stages and more research is needed to understand if LLMs are well applicable for those areas of testing.

Ref	Used LLM(s)	Peer-reviewed
[S11]	ChatGPT	Yes
[S18]	GPT-3.5	Yes
[S3]	ChatGPT	Yes
[S16]	GPT-4	Yes
[S4]	ChatGPT	Yes
[S6]	GPT-3	Yes
[S2]	ChatGPT	Yes
[S10]	GPT-3	Yes
[S19]	ChatGPT	Yes
[S20]	BERT	Yes
[S15]	GPT-4	Yes
[S12]	GPT-4	No
[S5]	GPT-4o	No
[S13]	3 different LLMs compared: GPT-3.5, GPT-4, Google Bard	No
[S7]	5 different LLMs compared: Gemini, ChatGLM-4, GPT-3.5, GPT-4, GPT-4o	No
[S1]	9 different LLMs compared: GPT-3, GPT-4, Baichuan, Spark, LLaMa2-7B, LLaMa2-13B, GLM-3, GLM-4, GLM-4V	No
[S8]	9 different LLMs compared: GPT-3.5, GPT-4, GPT-4 Turbo, Llama 2, Zephyr Alpha, Zephyr Beta, Nous Hermes Mixtral, MistralOrca, Code Llama	No
[S14]	GPT-4	No
[S17]	Not specified	No
[S9]	GPT-4 Turbo	No

Table 4.2: Used LLMs

As we can see from the two tables, LLMs are not always the sole solution to a testing problem but rather they are often used in combination with other traditional tools (e.g., in [S11, S4, S13]). For example, the collection of contextual information which is then given to LLMs for further analyzing, was often collected with the help of traditional tools. In addition to this, the LLM's response might be further directed to other tool's needs. Thus, LLMs typically take care of some specific parts in the testing of mobile applications.

4.3 Limitations and Challenges

Using LLMs in mobile application testing comes with its own problems. While some papers pointed out problems specific to their research case, many papers mentioned also more general issues of using LLMs in testing context. This section highlights the issues that were raised in the reviewed papers. Overall, 19 of the 20 reviewed papers mentioned

limitations or challenges, leaving the paper [S10] out of the review scope for this section.

Table 4.5 provides textual descriptions of the challenges or limitations that were collected from the reviewed papers. Figure 4.4 summarizes these challenges and limitations into a more precise visual format so that it is easier to get an idea about the prevalence of the mentioned issues. The findings that are described below are written based on this Table 4.5 and Figure 4.4.

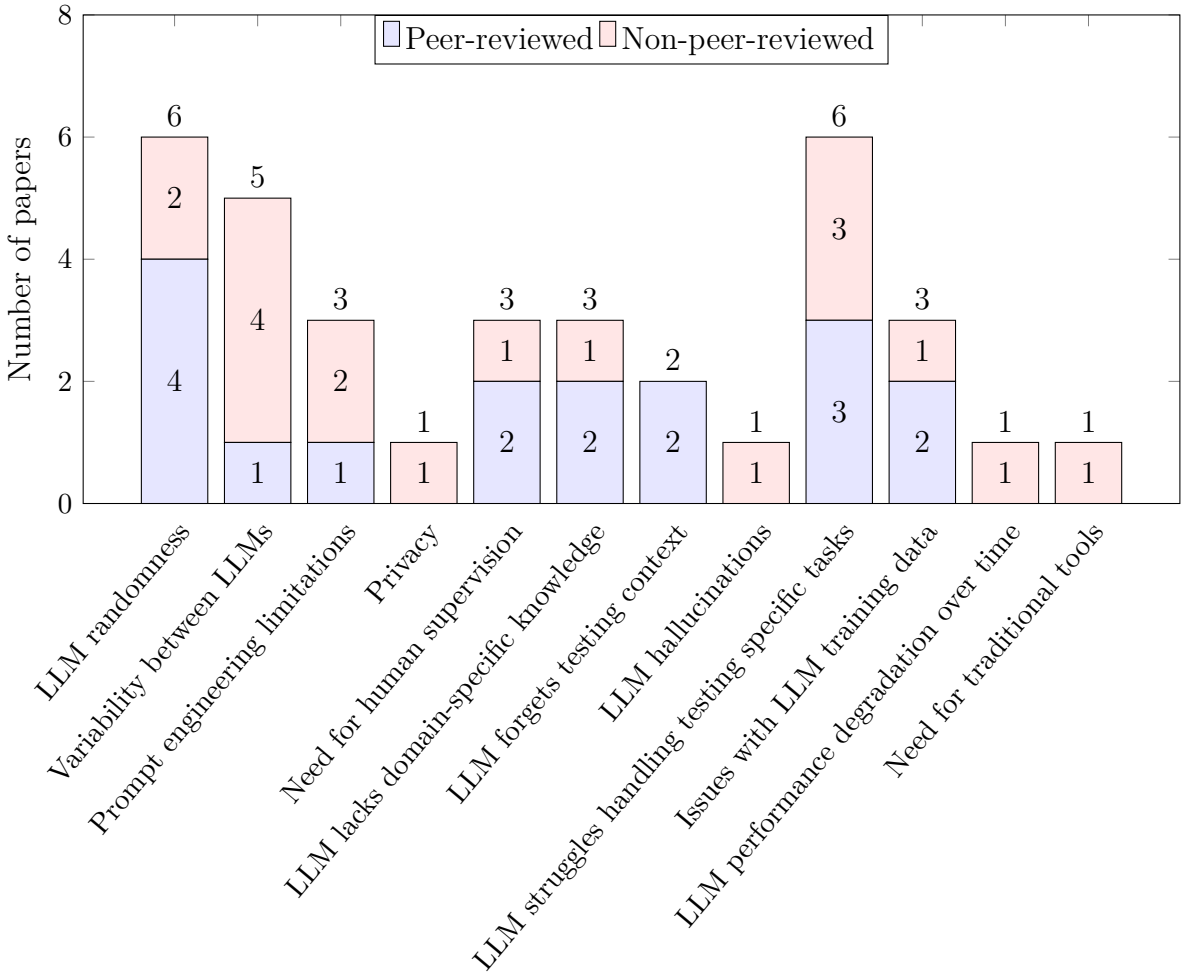

Figure 4.4: Limitations and challenges by number of papers that mentioned the issue

4.3.1 Issues With LLMs and Their Training Data

Some limitations and challenges are related to LLMs themselves. The most prevalent limitation or challenge for using LLMs in mobile aplications testing is the inherent randomness of LLMs [S18, S3, S16, S13, S7, S2]. The randomness causes many issues, such

as outputs in unexpected or wrong formats [S2] and different results even with the same question prompt. The LLM output might also contain incorrect information [S16].

There is also a lot of variability between different large language models which affects the testing results and performance [S7, S1, S8, S14, S16]. Different LLMs have different strengths and weaknesses, which makes some models more suited for certain tasks than others. The outputs of different models might also vary, which can lead to different testing results depending on which model is used and that can make the testing results unreliable. Due to this, a strategic selection of the model might be needed [S8] to ensure the LLM works well in the wanted use case.

Another commonly raised challenge is that while general-purpose models are often easily accessible and very powerful, they lack domain-specific knowledge [S4, S15, S5]. The lack of domain-specific knowledge limits the LLM capabilities when testing mobile applications, as they might lack the understanding of the tested applications or the testing activities themselves. The LLMs might require fine-tuning to make them better suited for their use purpose and to improve their capabilities in different testing areas [S5]. For example, if LLMs are used for GUI testing, the testing accuracy and efficiency could be improved by creating a LLM that is dedicated for that purpose [S5].

There were also more sparsely reported issues related to the LLMs themselves. These issues were mentioned only by one paper, but they still have an affect on how the LLMs are performing. These seldom reported issues are for example LLM hallucinations [S9] which cause the LLMs to imagine things or make faulty claims, and performance degradation of online business LLMs over time [S7], which can have a direct effect on the quality of testing. Privacy concerns were also raised since readily available and powerful GPT-type LLMs not suitable for privacy sensitive data [S1]. The privacy issues can be mitigated by using offline models, but they are often not as powerful as the GPT-type models [S1].

On top of the previously mentioned issues that are related to the nature of LLMs, LLM training data can also cause challenges. The training data might be out-dated or it might accidentally leak semantic data to the LLM which influences the testing results [S14]. Data contamination might also happen if the LLM knows about the subject application already through training data [S18]. These issues could be tackled by retraining the LLM, but it is often resource-intensive [S14]. Another limitation is that open-source LLMs are easier to fine-tune and use directly than proprietary models [S20], which can limit the use of proprietary models in mobile application testing.

4.3.2 Other Issues

In addition to the LLM related challenges, there were also more specialized challenges related to prompt engineering, performing testing tasks as well as the need for human supervision. We will describe these issues under this section.

Many of the reviewed papers raised issues related to performing specific testing tasks. For example, in GUI testing LLMs sometimes focus on low-level activities rather than high-level functionality, which is important for users [S11]. LLMs also struggle handling complex mobile testing activities, like scrolling or managing pop-ups [S19]. In addition to these, LLMs might not be able to identify bugs or crashes that require enabling dark theme in the mobile application [S6]. Testing might also be blocked if LLM needs valid inputs to proceed but they are not provided in the prompt [S6]. This could be a problem if the LLM needs for example a valid username and password to reach some other part of the app that is hidden behind the login functionality [S6].

LLMs sometimes also forget the testing context, which hinders their testing performance [S19, S11] as they do not always remember accurately what has already been tested. An example of this is that LLMs struggle remembering historical GUI explorations accurately [S11], which hinders the LLM decision making and thus affects the overall test results.

Limitations in prompt engineering might also cause problems. For example, too long prompts can cause the LLM to hallucinate, forget and plan tests ineffectively [S17]. Poor prompt design might also lead to biased or overall poor LLM outputs [S14]. In addition to the previously mentioned issues, it was also reported that due to the context limit of LLM prompts, preceding conversation had to be sometimes reset [S18], which limits how long conversations can be had with the LLM. However, the LLM answer quality can be improved by having a user who knows what to ask for and how [S14].

Lastly, a commonly reported challenge is that human intervention and supervision is still needed because LLMs cannot be fully trusted [S15]. This is due to the fact that LLMs might report high number of false positives [S7] or make errors [S15] in their reasoning and that causes an issue for fully automating the testing with LLMs. Thus, human supervision and intervention is often needed to ensure the LLM makes valid decisions and produces accurate results [S19, S15, S7].

4.4 Peer-reviewed vs Non-peer-reviewed Papers

Given the high number of non-peer-reviewed papers that were included in this review, we wanted to compare the results from these two different kinds of papers separately to see whether including the non-peer-reviewed ones will bias the overall results. Therefore, in this section, we will briefly highlight the main differences between the peer-reviewed and non-peer-reviewed papers.

For the most part, the testing topics were shared among the peer-reviewed and non-peer-reviewed papers, as we can see from Table 4.6. Both peer-reviewed and non-peer-reviewed papers presented ways to use LLMs in defect management, GUI testing, text input generation and test generation and maintenance. However, the possibility to use LLMs in vulnerability assessment and test reporting were only presented in the non-peer-reviewed papers. On the other hand, the peer-reviewed papers presented a way to use LLMs in test execution and replay, which was not covered in the non-peer-reviewed papers. This shows that the most common topics were the same for both peer-reviewed and non-peer-reviewed papers and the main differences lie in the smaller categories.

For the used LLMs, there are some clear differences between the peer-reviewed and non-peer-reviewed papers, as we can see from Table 4.2. As mentioned before, the main difference is that only non-peer-reviewed papers were comparing multiple models against each other for specific tasks while peer-reviewed papers were only using one model type in each paper. One non-peer-reviewed paper also didn't disclose the used LLM clearly, therefore it is described as "Not specified". Peer-reviewed papers had overall more variety in the selection of models if we leave the model comparisons for the same task out of consideration. GPT-4 model was used both by peer-reviewed and non-peer-reviewed models and it was a popular model overall. On the other hand, ChatGPT, which was the most commonly used model based on Figure 4.3, was only used in peer-reviewed papers.

When it comes to the role of LLMs, there was overall a lot of variation depending on the overall use case. Thus, highlighting differences between the peer-reviewed and non-peer-reviewed papers is challenging. The main difference is that the non-peer-reviewed recognized overall the need for further research more often than the peer-reviewed ones.

For the limitations and challenges, the main difference is that four non-peer-reviewed papers mentioned the variability of different LLMs as a challenge while only one of the peer-reviewed papers mentioned that same challenge (see Figure 4.4). This is probably due to the fact that only the non-peer-reviewed papers were comparing how different models

performed in the same task, while all the other papers had selected to use just one LLM in their solution, as we can see from Table 4.2. There were also some minor differences in the less commonly mentioned challenges, such as LLMs forgetting testing context, which was mentioned only by peer-reviewed papers, or privacy, which was mentioned only by non-peer-reviewed papers. Other mentioned challenges and limitations were quite evenly shared between the peer-reviewed and non-peer-reviewed papers.

Topic	Peer-reviewed	Count	Percentage	Overall count	Overall percentage
Defect management	Yes	4	20%	6	30%
	No	2	10%		
GUI testing	Yes	2	10%	4	20%
	No	2	10%		
Text input generation	Yes	2	10%	3	15%
	No	1	5%		
Test generation and maintenance	Yes	2	10%	3	15%
	No	1	5%		
Vulnerability assessment	Yes	0	0%	2	10%
	No	2	10%		
Test execution and replay	Yes	1	5%	1	5%
	No	0	0%		
Test reporting	Yes	0	0%	1	5%
	No	1	5%		
Total				20	100%

Table 4.6: Testing topics by paper type

Ref	General use case	LLM's role in practice	Result	Peer-reviewed
[S11]	Mobile GUI testing as Q&A task	LLM works as a human-like GUI tester, describing the steps a real tester might take when navigating between different views in the app	LLM provided valid operations to navigate through the app, thus helping to increase the testing coverage and identify bugs when combined with other tools	Yes
[S18]	Mobile GUI Testing	Four LLM-based agents work together to create GUI test case scripts that can achieve specific tasks	LLM-based solution achieved higher activity coverage than state-of-the-art GUI testing techniques, creating for the most part realistic tasks	Yes
[S3]	Automatic bug reproduction from bug reports	LLM simulates the reasoning process of an expert developer by interpreting bug reports, extracting actionable information, and guiding the reproduction of bugs	The LLM-based solution outperformed the state-of-the-art solutions at bug reproduction, making the replay process more efficient	Yes
[S16]	Bug report reproduction for Android apps	LLM works as a bug reproduction tool, generating event sequences that reproduce the reported bugs	LLM successfully handled both crash reports and non-crash functional bug reports and reproduced a high rate of bug reports in a fast manner, outperforming three existing tools both by speed and success rate	Yes
[S4]	Data inconsistency bug detection for mobile apps	LLM extracts target data (the values that need to be analyzed to meet the testing requirements) and identifies if there are inconsistency bugs by verifying relationships between these target data	LLM was able to reduce the manual effort that would usually go into identifying data inconsistency bugs; When combining the LLM together with other tools, the solution outperformed other GUI-based data inconsistency detection tools	Yes
[S6]	Crash reproduction by crash reports	LLM predicts the exploration steps for triggering a specific crash	LLM-based solution outperformed existing automated GUI tools and successfully reproduced majority of the crashes	Yes
[S2]	Unusual text input generation for mobile app crash detection	LLM works as a text input generator creating both valid and invalid inputs, with a focus on generating diverse invalid inputs that potentially cause mobile apps to crash	LLM-based solution had higher bug detection rate than best baseline solutions, and it helped to find new crashes	Yes
[S10]	Text input generation for mobile GUI testing	LLM generates text inputs based on contextual GUI information and linguistic patterns where the LLM can fill the missing parts of the text; those inputs work as input to different widgets to enable navigation on GUI views	The passing rate of the generated text inputs by the LLM-based tool was much higher compared to the best baseline and it covered more app activities and pages than other compared GUI testing tools	Yes
[S19]	Mobile application test script generation and migration	LLM generates test scripts for scenario-based test generation, cross-platform test migration, and cross-app test migration	LLm managed to generate scenario-based test scripts, cross-app test migration and cross-app migration test scripts when provided with adequate information; Certain limitations exist e.g. due to required manual effort; Further research needed	Yes
[S20]	GUI test case migration from source app to target app by widget matching	LLM works as a tool to convert each GUI widget's textual description into a context-aware numerical representation that can be used to compare widget similarities	LLM-based solution was effective and improved the accuracy both for event matching and test case migration when compared with baseline approaches	Yes
[S15]	Accessibility testing on iOS apps	LLM-based UI navigation system translates test instructions into a set of actionable steps, executes the steps on a live device and evaluates the results of the taken action to improve the navigation plan	The LLM-based system was successful at replaying manual test instructions and professional accessibility testers found the system to be useful in their work	Yes

Table 4.3: Details and outcome of LLM use in the peer-reviewed papers

Ref	General use case	LLM's role in practice	Result	Peer-reviewed
[S12]	Non-crash functional bug detection via mobile GUI	LLM decides which operations (e.g., click or scroll) are required to navigate the mobile GUI and predicts the expected output based on its contextual understanding of the app's behavior to identify non-crash bugs	When compared with 12 baselines, the LLM-based tool achieved improved average recall and precision compared to the best baseline	No
[S5]	Mobile GUI testing	Multiple LLM-based agents work together to enable GUI interaction and function verification; the LLM-based agents interpret and execute GUI interaction instructions and analyze the interaction results in comparison to expected outputs to verify function behavior	LLM-based solution outperformed existing tools in the quality of generated GUI interactions and achieved high accuracy in verification; the LLM-based tool also revealed new functional bugs in a real-life scenario	No
[S13]	Configuration compatibility bug detection and fixing for Android apps	LLM detects compatibility bugs from XML elements and/or fixes the compatibility issue causing XML elements	LLMs were overall ineffective at detecting and fixing compatibility bugs; Some promising results in addressing complicated compatibility issues; When combining LLMs with traditional compatibility fixing tools, the LLM-based tools outperformed the traditional tools	No
[S7]	Non-crash functional bug detection	LLM works as a test oracle for detecting non-crash functional bugs in Android applications	The LLM-based solution outperformed existing tools state-of-the-art tools in detecting non-crash functional bugs	No
[S1]	Text input generation for Android app GUI testing	LLM generates valid text inputs that can be inserted into the app to proceed from one GUI view to the next (allowing for higher testing coverage)	LLMs were able to generate text inputs that helped to detect real bugs in open-source apps; Performance between different LLMs varied; GPT series LLMs showed the strongest performance	No
[S8]	Android application vulnerability analysis	LLM detects Android vulnerabilities from vulnerable code samples	Different LLMs were compared for the same task and out of the compared LLMs GPT-4 and Code Llama emerged as the top performers; Overall performance of LLMs in vulnerability analysis is still in early stages; More research needed	No
[S14]	Vulnerability detection for Android applications	LLM detects vulnerabilities in Android apps and aims to fix them	LLMs showed promising results in mobile app vulnerability detection and fixing; However, LLMs are not suitable for vulnerability assessment out of the box; More research needed	No
[S17]	Test script generation for automated user acceptance testing (UAT)	Three LLM-based agents work with each other to handle different tasks for test script generation	The LLM-based system was close to the efficiency of humans in UAT test script generation; The system has been applied in a real-life scenario to reduce manual work	No
[S9]	Crowdsourced test report prioritization	LLM analyzes test reports and clusters them by the bug type (and this clustering is then used as a basis for report prioritization)	LLM-based solution prioritized test reports effectively and efficiently, outperforming the state-of-the-art approach	No

Table 4.4: Details and outcome of LLM use in the non-peer-reviewed papers

Ref	Limitations / challenges	Peer-reviewed
[S11]	LLM struggles at remembering historical GUI explorations accurately (hence losing the big picture), LLM focuses on low-level activities instead of the higher-level functionality	Yes
[S18]	Randomness of LLMs, LLM context limit, potential data contamination if the LLM knows about the subject application already due to training data	Yes
[S3]	Randomness of LLMs	Yes
[S16]	Randomness of LLMs, variability in LLM performance between different models, outputs produced by LLMs might be in wrong format or contain incorrect information	Yes
[S4]	LLMs lack domain-specific knowledge	Yes
[S6]	LLMs cannot always capture special preconditions for crashes (such as need to switch to dark theme), LLM cannot handle cases that require valid input (like user credentials) unless they are provided beforehand	Yes
[S2]	Uncertainty of LLM output results (results not given in the expected format)	Yes
[S10]	-	Yes
[S19]	Difficulties remembering testing context, manual human intervention needed, LLMs do not handle well complex test events, like scrolling or managing pop-up windows	Yes
[S20]	Open-source LLMs are easier to fine-tune and use directly compared to proprietary LLMs	Yes
[S15]	LLMs cannot be fully trusted, need for human intervention/supervision to avoid errors, people might be hesitant to question AI decisions even if they are incorrect, LLMs lack domain-specific knowledge	Yes
[S12]	MLLMs might be reliant on textual inputs even if screenshots can be provided to it (makes GUI testing challenging since LLM can't mark the actionable widgets on the screenshot), LLM focuses on operational actions instead of functionality, LLMs struggle determining what the current GUI page should be like after each performed action (test oracle problem)	No
[S5]	LLMs lack domain-specific knowledge	No
[S13]	LLMs require support from traditional tools, randomness of LLMs	No
[S7]	Performance degradation of online business LLMs over time, variability in LLM performance between different models, randomness of LLMs, high false positive rate (requires time-consuming manual verification from humans)	No
[S1]	Readily available GPT type LLMs not suitable for privacy sensitive data, the privacy issues can be mitigated by using offline LLMs but they aren't as effective as publicly available GPT LLMs	No
[S8]	Variability in LLM performance between different models, commercial LLMs poorer in security testing compared to open LLMs, LLMs not optimal for detecting nuanced security issues in Android applications	No
[S14]	Poor prompt design can cause biased or poor LLM output (requires skilled user who knows what to ask for and how), LLM training data might be out-dated or not consider newly emerged/found vulnerability issues (need to update and retrain the LLM which is resource-intensive), variability in LLM performance between different models, even seemingly neutralized ground truth might accidentally leak semantic data that influences the LLM results	No
[S17]	Long prompts can cause LLMs to forget, hallucinate and plan (tests) ineffectively	No
[S9]	LLM hallucinations, LLM performance might vary depending on the used language	No

Table 4.5: Challenges and limitations of using LLMs in mobile application testing

5 Discussion

5.1 The Potential of LLMs

Based on the literature review findings, LLMs have showed promising results in many areas of mobile application testing, such as defect management, GUI testing and text input generation (e.g., [S11, S1, S3]) by making these testing activities more efficient. Many of the proposed solutions have also outperformed existing state-of-the-art tools, which shows the potential of LLMs in improving current mobile application testing practices.

The contextual understanding capabilities and human-like reasoning capabilities of LLMs allow tasks that typically rely heavily on manual work to be transformed into tasks that a machine can do, like shown in [S3]. Thus, LLMs will likely make the mobile application testing processes more efficient in the future and reduce the need for human efforts, especially as LLMs get even more capabilities. However, since LLMs can still make errors or provide different results depending on which model is used, it is important not to fully rely on LLM decisions and have humans monitor the testing processes and results [S19, S15, S7].

As LLMs gain more capabilities, for example due to multi-modality [13], their applicability in mobile application testing is likely to improve. For example, GUI testing tasks might get easier as screenshots or possibly even screen recordings of the tested app are possibly to give to the LLMs as input. Due to these increased capabilities, the use of multi-modal LLMS in mobile application testing will likely become more common in the future.

Although many of the LLM-based testing solutions for mobile applications were successful, some papers also acknowledged the need for further research. Especially vulnerability assessment, test script migration and detection and fixing of compatibility bugs are still in early stages and more research is needed to understand whether LLMs can provide benefits in those testing areas [S19, S13, S8]. This is, for example, due to the unstable performance of different LLMs in these tasks and the need to combine other tools with LLMs for better performance, which requires good planning.

5.2 The Limitations of LLMs

5.2.1 The Importance of Selecting The Right Model

It's important to take into account the limitations of LLMs when it comes to testing mobile applications. Although LLMs show promising results in many areas of mobile application testing, using LLMs in testing still relies at least partly on human intervention and supervision due to the inherent randomness of LLMs and the unpredictability of the LLM responses [S15, S16].

It is clear that LLMs are not a magic tool for solving mobile application testing problems and they models come with their own issues. To ensure good performance of LLMs in mobile application testing, it is good to carefully consider which model or models to use. Different LLMs have different strengths and weaknesses, so a strategic selection of the model might be needed to ensure the LLM is well suited for its use case [S8].

The LLMs might also require retraining and fine-tuning because the lack of domain-specific knowledge hinders their performance in highly specialized testing tasks. For example, in the context of vulnerability analysis, it would be encouraged to use a model that is trained with up-to-date data about recently emerged or found vulnerabilities [S14]. Fine-tuning will help the models to provide more accurate and reliable results in testing tasks.

5.2.2 Privacy Concerns

Only one paper mentioned privacy issues as a limitation for using LLMs in mobile application testing, which is somewhat surprising, as privacy is often a top concern both for individual people and companies. Although user-specific data might not be directly given to LLMs, the LLMs often gain access to app-specific data that might end up being used in training of LLMs or other machine learning models.

The lack of concern over privacy might be related to the availability of offline LLM [S1] which have higher focus on privacy. However, existing research also shows that LLMs need urgently privacy-preserving mechanisms [10]. This suggests testers should not blindly rely on LLMs, especially without considering the business and privacy impact if sensitive information is leaked to the LLMs by accident.

5.2.3 Traditional Tools Are Still Needed

LLMs still require support from other tools, because the use of LLMs is often combined with other tools and using LLMs as the sole solution might not be beneficial [S13]. Previous research has also shown that LLMs might not be the sole solution for testing [18]. However, there is also not not a clear consensus to how well LLMs can solve software testing related problems when combined with other tools [18].

While having to combine LLMs with traditional tools is not necessarily a limitation, it aims to highlight that LLMs are often only a supportive tool in a certain part of the testing process, not the sole solution for testing, or a replacement for traditional manual or automated testing. For example, the LLM often needs to be provided with sufficient context information about the problem, which is collected with traditional tools [S11]. LLMs also seemed to perform better in some cases, when they were combined with other tools (e.g., [S11, S4, S13]).

This suggests that there is potential in combining traditional tools with LLMs to bring the best out of both worlds. Researchers, individuals and companies should be willing to try combined solutions to see if LLMs can provide better benefits that way. It might also be easier to adopt the use of new AI tools, like LLMs, when they are combined with something the users are already familiar with or use regularly.

5.3 Comparison to Related Work

The use of LLMs in mobile application testing is a relatively new research area and not much is known about the topic yet. However, there is existing literature on the use of LLMs in software testing in general. For example, previous research has shown that LLMs are used in the later stages of testing, like in system testing, instead of the early stages of testing, like test planning [18]. This seems to be the case also with mobile application testing, as the presented solutions are focusing on testing the fully integrated application. Thus, our research aligns well with previous work. However, this also means that the existing gap of not using LLMs in early stages of software testing still remains.

Existing literature has also identified problems in current mobile test automation tools. For example, previous research shows that mobile applications are missing good automated solutions for security testing [1]. This was addressed in the papers we reviewed, as our research identified two papers that explored the possibility of utilizing LLMs in

vulnerability assessment [S8, S14]. Although it was acknowledged that more research is needed before LLMs can be successfully used for vulnerability assessment, LLMs still show potential in identifying and fixing vulnerabilities in mobile apps. This gives hope for designing better security testing solutions for mobile applications in the future with the help of LLMs. However, the need for better regression testing tools [1] was not addressed in the papers we reviewed.

In the future, mobile application testing will likely be a further integration of AI, machine learning and automated processes [12]. This aligns well with the results of this thesis, as LLMs have showed promising results in many mobile application testing areas and some of the solutions have already been tried successfully in real-life scenarios [S5, S17, S15]. LLMs are also providing solutions to issues that have previously required a lot of manual effort, such as generating a variety of invalid inputs to detect crashes or adding human-like thinking and reasoning into GUI testing. However, LLMs are not a straight out of the box solution for testing problems and come with their own issues that need to be addressed for successful LLM use in the future.

5.4 Answers to the Research Questions

The goal of this thesis was to find answers to two research questions.

RQ1: How are LLMs used in mobile application testing?

LLMs are used in seven different mobile application testing areas: defect management, GUI testing, text input generation, test generation and maintenance, test execution and replay, vulnerability assessment and test reporting. LLMs have showed promising results especially in the first three categories, and the LLM-based solutions have helped to, for example, improve testing coverage and make the testing processes more efficient. They have also addressed previously challenging issues, such as how to create a plethora of invalid inputs for crash detection automatically.

However, LLMs have also limitations in some areas. For example, compatibility bug detection and fixing is still in early stages due to poor LLM performance in this task and the reliance on traditional tools. LLMs have also limited applicability in vulnerability assessment due to the complexity of the vulnerability issues and problems in LLM training data. Test case migration was also challenging because of the need for human intervention.

The role of LLMs varies a lot depending on the use case, and giving a direct answer

is difficult. Some examples of the role of LLMs are that the LLM simulates a human professional in reproducing bugs, works as a text input generator creating a variety of invalid inputs based on instructions or detects vulnerabilities in code samples. Often the LLM works as a tool to generate something, such as text inputs, test scripts or actionable GUI navigation steps.

The used models were also compared. Most commonly used models were ChatGPT, GPT-4 and GPT-3, while other models had a smaller share. Performance comparisons between different models in specific tasks were also popular. This might suggest these GPT-type models are powerful and best suited for mobile application at least currently.

Due to the high number of non-peer-reviewed papers, we also compared the results between the peer-reviewed and non-peer-reviewed papers to identify possible differences. The most popular testing topics were shared among the non-peer-reviewed papers and main differences lie in the smaller topics. Thus, in this regard, the non-peer-reviewed papers did not bias the main findings.

The LLM roles had similar characteristics in both peer-reviewed and non-peer-reviewed papers. However, the LLM applicability seemed to be better in the peer-reviewed papers, as the non-peer-reviewed papers identified more limitations. The main difference in the use of models is that many peer-reviewed papers used ChatGPT, while many of the non-peer-reviewed ones compared multiple models in the same task.

RQ2: What kind of limitations or challenges are there for using LLMs in mobile application testing?

Many different limitations and challenges were identified. Most notable ones are the randomness of LLMs, which causes unpredictable outputs, and the variability between different LLMs, which also affects the testing performance depending on which model is used. One model might work better in a specific task than some other model, which means that the used model should be selected carefully and strategically. Many LLMs also lack domain-specific data, which hinders the model's ability to handle highly specialized tasks, such as GUI testing.

LLMs also have limited ability to handle certain testing specific tasks, such as scrolling. Some less reported, but still notable challenges are also LLMs forgetting the testing context, limitations in prompt engineering (such as context limit) and problems with the LLM training data.

As the number of non-peer-reviewed papers was high, we analyzed the peer-reviewed and

non-peer-reviewed papers separately also in this case. The main difference is that mainly non-peer-reviewed papers pointed out variability between different LLMs as a challenge. This is likely due to the nature of those papers, because they were comparing the use of different models in the same testing tasks. The non-peer-reviewed papers also identified overall more challenges and limitations. However, other commonly identified challenges and limitations were quite evenly shared between the papers, which means that the impact of including the non-peer-reviewed papers had a big impact only on one of the identified challenges, which was the variability between models.

5.5 Ideas for Future work

Based on our findings, the current use of LLMs in mobile application testing is heavily focused on similar kinds of tasks, since four topics covered already 80% of the LLM use cases: defect management, GUI testing, test generation and maintenance and text input generation. Thus, different areas of testing, such as API testing for mobile application backend, could be explored. Since better regression testing tools are also needed [1] and LLMs could help to provide better automated solutions for that purpose, regression testing with the help of LLMs is a possible research direction as well.

Previous research also shows that LLMs focus heavily on functional testing [18]. Thus, non-functional testing with LLMs is a gap in the current research [18]. Based on our findings, this research gap was not well addressed for mobile application testing either, although non-functional testing was slightly covered by two papers that focused on vulnerability assessment. However, vulnerability assessment of mobile applications is still in early stages and more research is needed. Therefore, different non-functional testing activities, such as security testing and performance testing, could be explored.

5.6 Limitations of the Study

To improve the validity of this research, we want to be transparent with the possible limitations of this study.

The main concern for validity is the relatively high number of non-peer-reviewed papers. This can partly be explained by the novelty of the thesis topic, as well as how recently many of the papers have been published, so it is possible that they have not yet gone

through the peer review process. We aimed to minimize this risk by being transparent of the papers that are not peer-reviewed. We also analyzed the results of peer-reviewed and non-peer-reviewed both together, as well as separately, to see if there are differences between these two types of papers. The main differences that were found are highlighted throughout this thesis.

It is also possible that due to limiting the Google Scholar results into the top 200 first results, some relevant results might have been excluded outside of these 200 papers. In the ideal case, every article should have been checked, and if similar research is repeated in the future, it would be advisable to go through every result. However, the risk of missing relevant papers was mitigated by the fact that Google sorts the results by relevance, and as the majority of the selected papers were already in the top 100 results, the risk of missing relevant papers due to this limitation is low.

6 Conclusions

In this thesis, we conducted a systematic literature review (SLR) on the use of LLMs in mobile application testing, highlighting both the possibilities as well as challenges and limitations. The review consisted of 20 research papers from the years 2023-2024. Based on the findings, two research questions were answered:

RQ1: How are LLMs used in mobile application testing?

LLMs are used in seven different mobile application testing categories. These categories include e.g., defect management, GUI testing, text input generation and test generation and maintenance. The role of the LLM in these tasks varies. For example, the LLM might generate test scripts, simulate human testers in their thinking or detect vulnerability issues from code snippets. Most commonly used LLMs were ChatGPT and GPT-4.

RQ2: What kind of limitations or challenges are there for using LLMs in mobile application testing?

The use of LLMs in mobile application testing has many challenges and limitations. These include the inherent randomness of LLMs, the variability across different models, and the lack of domain-specific knowledge, all of which cause unaccuracy and unpredictability in the testing. Additionally, LLMs struggle with specific testing tasks, such as managing app view scrolling. Other, less frequently discussed issues include problems in LLM training data and the tendency of LLMs to lose contextual information during testing.

Despite these limitations, this study highlights the promising potential of LLMs in certain mobile application testing areas, such as GUI testing and text input generation. However, some other areas, like vulnerability assessment, are still are still in early stages and require more research.

While some of the known challenges and limitations of using LLMs can be tackled by model fine-tuning, certain limitations and challenges are still likely to exist even years from now. For example, due to the inherently random nature of LLMs, human supervision will probably still be needed since LLMs can produce results that are not accurate. It is crucial to overcome these challenges in order to successfully integrate LLMs into mobile application testing and software testing as a whole.

The future prospects of using LLMs are bright: LLMs are developing fast and their ca-

pabilities are continuously increasing. As LLMs get even more powerful and mainstream, their use in software testing and specifically in mobile application testing will become more common. LLMs hold contextual understanding capabilities which traditional automated tools are lacking, and that enables them to be used in areas of testing that typically rely heavily on manual efforts.

Although these initial findings are promising, the use of LLMs in mobile application testing is still in early stages and more research is needed to assess their real-world applicability. Nevertheless, our findings indicate that LLMs have significant potential to enhance mobile application testing processes, as they can address some of the limitations in current automated testing tools and reduce the reliance on manual work, which has been a long-standing challenge in the field.

Bibliography

[1] K. S. Arif and U. Ali. "Mobile Application testing tools and their challenges: A comparative study". In: *2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*. 2019, pp. 1–6. DOI: 10.1109/ICOMET.2019.8673505.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. "Language models are few-shot learners". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. Red Hook, NY, USA, 2020, pp. 1877–1901.

[3] H. Du, D. Niyato, J. Kang, Z. Xiong, P. Zhang, S. Cui, X. Shen, S. Mao, Z. Han, A. Jamalipour, H. V. Poor, and D. I. Kim. "The Age of Generative AI and AI-Generated Everything". In: *IEEE Network* 38.6 (2024), pp. 501–512. DOI: 10.1109/MNET.2024.3422241.

[4] M. Fazzini. "Automated support for mobile application testing and maintenance". In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Lake Buena Vista, FL, USA, 2018, pp. 932–935.

[5] S. Feuerriegel, J. Hartmann, C. Janiesch, and P. Zschech. "Generative AI". In: *Business & Information Systems Engineering* 66 (2024), pp. 111–126. DOI: https://doi.org/10.1007/s12599-023-00834-7.

[6] R. Gozalo-Brizuela and E. C. Garrido-Merchán. *A survey of Generative AI Applications*. 2023. URL: https://arxiv.org/abs/2306.02781.

[7] IBM. *What are LLMs?* URL: https://www.ibm.com/think/topics/large-language-models.

[8] IBM. *What is NLP?* URL: https://www.ibm.com/think/topics/natural-language-processing.

[9] T. Mahmud, M. Che, A. H. H. Ngu, and G. Yang. "An Empirical Investigation on Android App Testing Practices". In: *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. 2024, pp. 355–366. DOI: 10.1109/ISSRE62328.2024.00042.

[10] N. Mireshghallah, H. Kim, X. Zhou, Y. Tsvetkov, M. Sap, R. Shokri, and Y. Choi. *Can LLMs Keep a Secret? Testing Privacy Implications of Language Models via Contextual Integrity Theory*. 2024. URL: https://arxiv.org/abs/2310.17884.

[11] M. Motan and S. Zein. "Android App Testing: A Model for Generating Automated Lifecycle Tests". In: *2020 4th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. 2020, pp. 1–11. DOI: 10.1109/ISMSIT50672.2020.9254285.

[12] F. Mulla. "Modern Mobile Testing Tools: A Comprehensive Guide to Quality Assurance and Automation". In: *International Journal of Scientific Research in Computer Science Engineering and Information Technology* 10 (2024), pp. 1577–1584. DOI: 10.32628/CSEIT24106120.

[13] OpenAI. *GPT-4 is OpenAI's most advanced system, producing safer and more useful responses*. URL: https://openai.com/index/gpt-4/.

[14] OpenAI. *What are LLMs?* URL: https://openai.com/index/chatgpt/.

[15] M. A. K. Raiaan, M. S. H. Mukta, K. Fatema, N. M. Fahad, S. Sakib, M. M. J. Mim, J. Ahmad, M. E. Ali, and S. Azam. "A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges". In: *IEEE Access* 117 (2024), pp. 26839–26874. DOI: https://doi.org/10.1016/j.jss.2016.03.065.

[16] Statista. *Number of mobile phone messaging app users worldwide from 2019 to 2025*. URL: https://www.statista.com/statistics/483255/number-of-mobile-messaging-users-worldwide/.

[17] Statista. *Number of smartphone mobile network subscriptions worldwide from 2016 to 2023, with forecasts from 2023 to 2028*. URL: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.

[18] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang. "Software Testing with Large Language Models: Survey, Landscape, and Vision". In: *IEEE Transactions on Software Engineering* (2024), pp. 1–27. DOI: 10.1109/TSE.2024.3368208.

[19] J. D. Weisz, M. Muller, J. He, and S. Houde. *Toward General Design Principles for Generative AI Applications*. 2023. DOI: https://doi.org/10.48550/arXiv.2301.05578.

[20] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T. Liu. "On Layer Normalization in the Transformer Architecture". In: *Proceedings of the 37th International Conference on Machine Learning*. 2020, pp. 10524–10533. URL: https://proceedings.mlr.press/v119/xiong20b.html.

[21] S. Zein, N. Salleh, and J. Grundy. "A systematic mapping study of mobile application testing techniques". In: *Journal of Systems and Software* 117 (2016), pp. 334–356. DOI: https://doi.org/10.1016/j.jss.2016.03.065.

[22] X. Zhang, Y. Zeng, Q. Li, G. Chen, Q. Xu, X. Hu, and Z. Peng. "DesignWatch: Analyzing Users' Operations of Mobile Apps Based on Screen Recordings". In: *Adjunct Proceedings of the 26th International Conference on Mobile Human-Computer Interaction*. New York, NY, USA, 2024, pp. 1–7.

SLR Bibliography

[S1] C. Cui, T. Li, J. Wang, C. Chen, D. Towey, and R. Huang. *Large Language Models for Mobile GUI Text Input Generation: An Empirical Study*. 2024. URL: https://arxiv.org/abs/2404.08948.

[S2] M. Fazzini. "Testing the Limits: Unusual Text Inputs Generation for Mobile App Crash Detection with Large Language Model". In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. New York, NY, USA, 2024, pp. 1–12.

[S3] S. Feng and C. Chen. "Prompting Is All You Need: Automated Android Bug Replay with Large Language Models". In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. New York, NY, USA, 2024, pp. 1–13.

[S4] Y. Hu, H. Jin, X. Wang, J. Gu, S. Guo, C. Chen, X. Wang, and Y. Zhou. "AutoConsis: Automatic GUI-driven Data Inconsistency Detection of Mobile Apps". In: *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*. New York, NY, USA, 2024, pp. 137–146.

[S5] Y. Hu, X. Wang, Y. Wang, Y. Zhang, S. Guo, C. Chen, X. Wang, and Y. Zhou. *AUITestAgent: Automatic Requirements Oriented GUI Function Testing*. 2024. URL: https://arxiv.org/abs/2407.09018.

[S6] Y. Huang, J. Wang, Z. Liu, Y. Wang, S. Wang, C. Chen, Y. Hu, and Q. Wang. "CrashTranslator: Automatically Reproducing Mobile Application Crashes Directly from Stack Trace". In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. New York, NY, USA, 2024, pp. 1–13.

[S7] B. Ju, J. Yang, T. Yu, T. Abdullayev, Y. Wu, D. Wang, and Y. Zhao. *A Study of Using Multimodal LLMs for Non-Crash Functional Bug Detection in Android Apps*. 2024. URL: https://arxiv.org/abs/2407.19053.

[S8] V. Kouliaridis, G. Karopoulos, and G. Kambourakis. *Assessing the Effectiveness of LLMs in Android Application Vulnerability Analysis*. 2024. URL: https://arxiv.org/abs/2406.18894.

[S9] Y. Ling, S. Yu, C. Fang, G. Pan, J. Wang, and J. Liu. *Redefining Crowdsourced Test Report Prioritization: An Innovative Approach with Large Language Model*. 2024. URL: https://ssrn.com/abstract=4741001.

[S10] Z. Liu, C. Chen, J. Wang, X. Che, Y. Huang, J. Hu, and Q. Wang. "Fill in the Blank: Context-aware Automated Text Input Generation for Mobile GUI Testing". In: *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. Melbourne, Australia, May 2023, pp. 1355–1367.

[S11] Z. Liu, C. Chen, J. Wang, M. Chen, B. Wu, X. Che, D. Wang, and Q. Wang. "Make LLM a Testing Expert: Bringing Human-like Interaction to Mobile GUI Testing via Functionality-aware Decisions". In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. New York, NY, USA, 2024, pp. 1–13.

[S12] Z. Liu, C. Li, C. Chen, J. Wang, B. Wu, Y. Wang, J. Hu, and Q. Wang. *Vision-driven Automated Mobile GUI Testing via Multimodal Large Language Model*. 2024. URL: https://arxiv.org/abs/2407.03037.

[S13] Z. Liu, Y. Tang, M. Li, X. Jin, Y. Long, L. F. Zhang, and X. Luo. *LLM-CompDroid: Repairing Configuration Compatibility Bugs in Android Apps with Pre-trained Large Language Models*. 2024. URL: https://arxiv.org/abs/2402.15078.

[S14] N. S. Mathews, Y. Brus, Y. Aafer, M. Nagappan, and S. McIntosh. *LLbezpeky: Leveraging Large Language Models for Vulnerability Detection*. 2024. URL: https://arxiv.org/abs/2401.01269.

[S15] M. Taeb, A. Swearngin, E. Schoop, R. Cheng, Y. Jiang, and J. Nichols. "AXNav: Replaying Accessibility Tests from Natural Language". In: *Proceedings of the CHI Conference on Human Factors in Computing Systems*. New York, NY, USA, May 2024, pp. 1–16.

[S16] D. Wang, Y. Zhao, S. Feng, Z. Zhang, W. G. J. Halfond, C. Chen, X. Sun, J. Shi, and T. Yu. "Feedback-Driven Automated Whole Bug Report Reproduction for Android Apps". In: *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. New York, NY, USA, 2024, pp. 1048–1060.

[S17] Z. Wang, W. Wang, Z. Li, L. Wang, C. Yi, X. Xu, L. Cao, H. Su, S. Chen, and J. Zhou. *XUAT-Copilot: Multi-Agent Collaborative System for Automated User Acceptance Testing with Large Language Model*. 2024. URL: https://arxiv.org/abs/2401.02705.

[S18] J. Yoon, R. Feldt, and S. Yoo. “Intent-Driven Mobile GUI Testing with Autonomous Large Language Model Agents”. In: *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*. 2024, pp. 129–139.

[S19] S. Yu, C. Fang, Y. Ling, C. Wu, and Z. Chen. “LLM for Test Script Generation and Migration: Challenges, Capabilities, and Opportunities”. In: *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*. Chiang Mai, Thailand, 2023, pp. 206–217.

[S20] Y. Zhang, W. Zhang, D. Ran, Q. Zhu, C. Dou, D. Hao, T. Xie, and L. Zhang. “Learning-based Widget Matching for Migrating GUI Test Cases”. In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. New York, NY, USA, 2024, pp. 1–13.